Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Návrh a implementace univerzální architektury pro sběr senzorických dat a automatizaci

Disertační práce

Autor: Ing. Jan Štěpán
Studijní program: Aplikovaná informatika
Studijní obor: Aplikovaná informatika

Školitel: doc. Ing. Vladimír Soběslav, Ph.D.
Pracoviště školitele: Katedra informačních technologií

Hradec Králové Březen 2021

Prohlášení

Prohlašuji, že jsem disertační práci zpracoval samostatně a s použitím uvedené literatury a pramenů.

V Hradci Králové dne Ing. Jan Štěpán

Abstrakt

Disertační práce se zabývá v současnosti stále diskutovanějším tématem Internetu věcí (Internet of Things, IoT). V úvodní části práce je definován samotný pojem *Internet věcí* jako systém vzájemně propojených zařízení, jsou diskutovány různé možnosti využití a jeho přínosy. Následně je popsán doménový a funkční model Internetu věcí a jsou stanoveny základní prvky IoT platforem. Tedy zařízení, která integrují senzory a aktuátory, dále bran provádějících překlad komunikačních protokolů pro zařízení nevybavených IP protokolem, a integračního middlewaru. Podstatou middlewaru je poskytovat data klientským aplikacím, se kterými poté interagují další služby a uživatelé. Na úvodní část navazuje rešerše technologií využitelných v IoT a dekomponováním problematiky z různých hledisek. Dále se práce zabývá prezentací současných výzkumných problémů v oblasti Internetu věcí, jmenovitě na synchronizaci času v systémech, schopností protokolových bran, poskytování dat v reálném čase, edge computingem a virtuálními a externími zařízeními. Po analýze existujících komerčních a open-source řešení je zjištěno, že platforma, která by tyto problémy souhrnně řešila, v současné době neexistuje. Z tohoto důvodu je v další kapitole navržena architektura nového řešení s cílem vyřešit výše uvedené problémy. Ta je v částečně implementována a je ukázáno její využití napříč vědeckými projekty. V závěrečné diskuzi je ukázáno, že předkládané řešení uvedené požadavky splňuje.

Klíčová slova: Internet věcí, computing, automatizace, senzory, protokoly

Abstract

The dissertation thesis deals with the currently increasingly discussed topic of the Internet of Things (IoT). The introductory part of the thesis defines the very concept of the Internet of Things as a system of interconnected devices and discusses its various uses and benefits. The domain and functional model of the Internet of Things is described and the basic elements of IoT platforms are presented. Devices integrate sensors and actuators, gateways are used for translating communication protocols for devices not equipped with an IP protocol, and integration middleware. Its essence is to provide data for client applications, which are used for other services and user interactions. The introductory part is followed by presenting relevant technologies usable in IoT and by decomposing the topic from various perspectives. Furthermore, the work deals with the presentation of current research problems, namely the synchronization of time in platforms, the capabilities of protocol gateways, real-time data provision, edge computing and virtual and external devices. After analyzing the existing commercial and open-source solutions, it is found that a platform that would comprehensively solve these problems does not currently exist. For this reason, the next chapter proposes the architecture of the new solution in order to solve the above problems. It is partially implemented and its use across scientific projects is shown. The final discussion shows that the presented solution successfully implements all requirements.

Keywords: Internet of Things, computing, automation, sensors, protocols

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

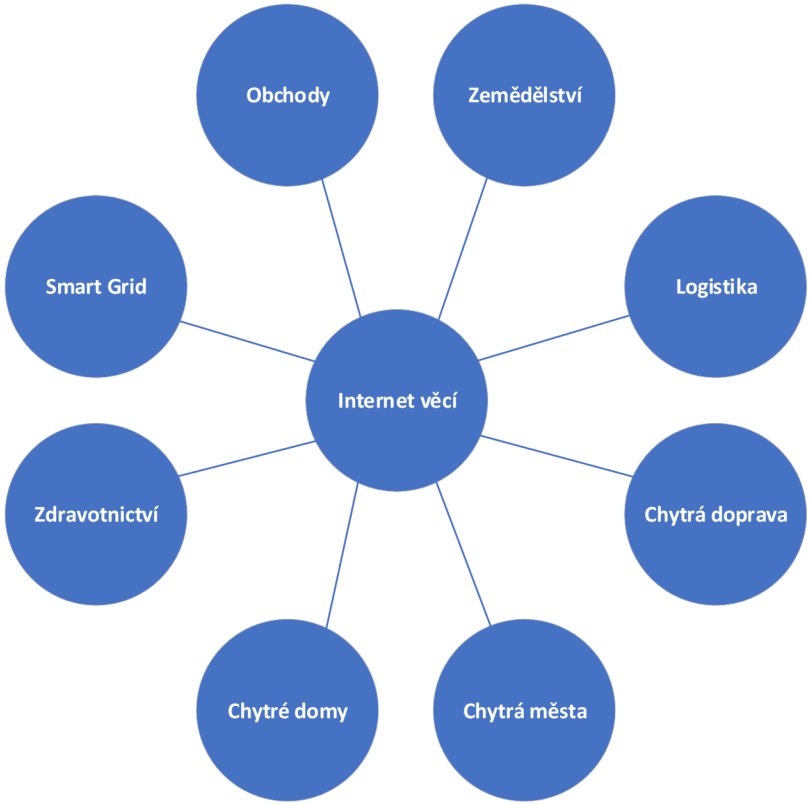
6LoWPAN ... IPv6 over Low Power Wireless Personal Area Networks
... IPv6 pro bezdrátové osobní sítě s nízkou spotřebou energie
API ... Application Programming Interface
... Rozhraní pro programování aplikací
BLE ... Bluetooth Low Energy
CPU ... Central Processing Unit
... Centrální výpočetní jednotka
CRC ... Cyclic redundancy check
... Cyklický redundantní součet
CRUD ... Create Read Update Delete
... Vytvořit Číst Editovat Smazat
EEPROM ... Electrically Erasable Programmable Read-Only Memory
... Elektronicky Vymazatelná Paměť pouze pro čtení
CHOPN ... Chronická obstrukční plicní nemoc
I^2C ... Inter-Integrated Circuit
... Sběrnice pro komunikaci integrovaných obvodů
IaaS ... Infrastucture as a Service
... Infrastruktura jako služba
IoT ... Internet of Things
... Internet věcí
IoTIM ... IoT Integration Middleware
... IoT integrační middleware
IIoT ... Industrial Internet of Things
... Průmyslový Internet věcí
LED ... Light emitting diode
... Dioda emitující světlo
PaaS ... Platform as a Service
... Platforma jako služba
PM ... Particulate matter
... Pevné částice

PWM ... Pulse width modulation
... Pulzně šířková modulace
REST ... Representational State Transfer
... Architektura rozhraní
SaaS ... Software as a Service
... Software jako služba
SDK ... Source development kit
... Vývojové zdrojové kódy
SOA ... Service Oriented Architecture
... Servisně orientovaná architektura
SPI ... Serial peripheral interface
... Sériové periferní rozhraní
SQL ... Structured query language
... Strukturovaný dotazovací jazyk
TA ČR ... Technologická agentura České republiky
TSDB ... Time Series Databases
... Databáze pro ukládání časových řad
VPS ... Virtuální privátní server
UART ... Universal Asynchronous Receiver Transmitter
... Univerzální asynchronní přijímač a vysílač
UML ... Unified Modeling Language
... Unifikovaný modelovací jazyk

1

Úvod

Koncept Internetu věcí (angl. Internet of Things, IoT) se datuje do roku 1982, kdy skupina výzkumníků modifikovala výdejní automat na nápoje a připojila ho k internetu [144]. Přes velmi jednoduché rozhraní bylo možné zjistit počty dostupných nápojů a zda jsou vychlazeny na správnou teplotu. V roce 1991 byla Markem Weiserem představena moderní vize Internetu věcí ve formě všudypřítomných výpočtů (ubiquitous computing) [127]. V roce 1999 pak navrhl Kevin Ashton termín *Internet věcí*, popisující systém vzájemně propojených zařízení [13].



Obrázek 1.1: Vybrané oblasti využití internetu věcí [autor]

Problematika Internetu věcí se objevuje téměř ve všech oborech, které nějakým způsobem zahrnují elektronická zařízení. Existují různé definice Internetu věcí, např. že Internet věcí je „otevřená a komplexní

síť inteligentních objektů, které jsou schopny automaticky organizovat a sdílet informace, data a zdroje, reagovat a jednat s ohledem na situace a změny prostředí“ [70]. Případně lze říci, že „Internet věcí lze také považovat za globální síť, která umožňuje komunikaci mezi lidmi a lidmi, lidmi a věcmi či věcmi a věcmi, což může být cokoliv na světě s tím, že každému objektu poskytuje jedinečnou identitu.“ [5] Lze tedy říci, že pojem Internet věcí není zcela jednoznačně definován. Obecně řečeno se tedy jedná o jakékoliv uživatele či jejich zařízení, která jsou připojeny do sítě Internet, která mohou komunikovat s dalšími zařízeními či uživateli a současně i jednat autonomně.

Internet věcí ve své obecné koncepci tedy nezahrnuje pouze různé věci (zařízení), ale může se jednat i o lidi či zvířata. Například díky nositelným senzorům lze získávat a dále zasílat informace o aktuálním vitálním stavu jedinců a reagovat na něj. V současné době se rozvíjí moderní trend využití různých metod přenosu vitálních dat v rámci telemedicíny. S přibývajícím množstvím různých senzorů a možností zjišťování vitálních dat bude na toto odvětví kladen stále větší důraz. Velice často se ve spojitosti s IoT mluví o oborech vyznačených na obrázku 1.1. Nejedná se o vyčerpávající výčet, jelikož elektronická zařízení se v současné době využívají téměř ve všech oborech lidské činnosti. Tyto vybrané oblasti jsou ale poměrně typické a využití prvků IoT je zde časté. Ve všech těchto oblastech umožňuje využívání principu IoT zefektivnění vykonávaných činností. Možnost komunikovat, tedy předávat či přijímat data na velkou vzdálenost, umožňuje optimalizovat činnosti napříč obory. Současně aktuální trend miniaturizace a extrémního zlevňování komunikačních prvků podporuje rozvoj řešení využívající IoT. Pro zdůraznění důležitosti a využitelnosti IoT budou nyní bodově popsána vybraná odvětví a případy užití, které je možné v současné době nebo blízké budoucnosti realizovat [66].

- Domácnosti
  - Chytrá zařízení jako hlasoví asistenti, dveřní zámky, kamery, termostaty atd.
  - Nositelná elektronika pro monitoring vitálních funkcí.
  - Lokační a monitorovací zařízení pro zvířecí společníky.
- Maloobchody, finančnictví a marketing
  - Cílené reklamy na základě lokace zákazníků a poskytování informací o slevách.
  - Detekce vzorů chování zákazníků.
  - Sledování zboží, kontrola inventur a ztrát, optimalizace zásobovacích řetězců.
  - Monitoring mrazicích boxů a rychle se kazícího zboží, prediktivní analýza selhání.
  - Navádění návštěvníků kulturních akcí.
- Zdravotnictví
  - Vzdálený monitoring pacientů z domova.
  - Učení modelů prediktivní a preventivní péče.
  - Monitoring seniorů a pacientů s vážnými nemocemi.
  - Dohled nad nemocničním vybavením a zásobovací logistikou.
  - Sledování a zabezpečení léčiv.

  - Detekce pádu pacientů.
  - Trasování nakažlivých osob během pandemické situace.
- Logistika
  - Trasování a zvyšování povědomí o poloze firemních flotil.
  - Identifikace a trasování přepravních vozíků.
  - Preventivní údržba vozidel.
- Zemědělství a životní prostředí
  - Chytré zavlažování a hnojení pro zvýšení úrody.
  - Automatická preventivní údržba farmářské techniky výrobcem.
  - Chytré osvětlení drůbežích farem pro zvýšení výnosů.
  - Průzkum složení terénu pomocí dronů.
  - Robotické či vertikální farmaření.
  - Monitorování vulkanické aktivity pro predikci katastrof.
- SmartGrid
  - Vzdálený monitoring a údržba solárních panelů.
  - Chytré elektroměry pro monitoring využívání energie v měřítku měst.
  - Úprava náklonu rotorových listů větrných elektráren dle počasí.
  - Analýza senzorů na ropných plošinách pro zvýšení účinnosti.
- Města a doprava
  - Regulace znečištění díky senzorickým měřením.
  - Predikce změn mikroklimatu díky senzorům po celém městě.
  - Zlepšování propustnosti dopravy chytrým řízením semaforů.
  - Parkoviště hledající nejlepší parkovací místa dle požadavků uživatele.
  - Zvyšování efektivity svozu odpadu skrze měření zaplněnosti kontejnerů.
- Vláda a armáda
  - Real-time sledování vojenského personálu a techniky.
  - Detekce leteckých hrozeb pomocí dronů.
  - Měření stavu naplněnosti přehrad a vodních toků pro predikci povodní.

Na trhu již existuje mnoho IoT řešení a malé i velké technologické firmy se zabývají tvorbou IoT systémů. Často je ale problémem vzájemná nekompatibilita těchto systémů a zkratka IoT by se v takových případech dala s nadsázkou vykládat jako "Intranet věcí". Při návrhu jakéhokoliv systému v rámci internetu věcí je nutné přemýšlet nad interoperabilitou, tedy možností komunikace a interakce s dalšími zařízeními.

Nevýhodou mnoha řešení v oblasti informačních technologií i IoT je jejich uzavřenost a chybějící podrobná dokumentace, která by usnadnila jejich nasazení v dalších oblastech. Většina řešení je připravena pro použití ve vybraných případech většinou komerčního charakteru, ale pokud by měly by být využity i pro jiné účely, např. ve výzkumných a vývojových projektech, není to vůbec možné nebo je to obtížné.

V této práci je popsáno řešení, jehož architektura splňuje výše uvedené definice IoT a zároveň je možné její využití ve speciálních (zejména výzkumných) projektech díky možnosti realizovat senzorický výzkumný systém od sběru surových dat až po tvorbu edge computing algoritmů (viz dále).

S neustále se zvyšující populací planety, stále dostupnějšími elektronickými zařízeními a rostoucí ekonomikou vznikl v informačních technologií problém: Každý rok je lidmi a stroji generováno stále větší množství dat. Je odhadováno [145], že v roce 2018 bylo vytvořeno 33 zettabajtů (10^{21} bajtů) nových dat. S tím přichází problémy v podobě limitů komunikačních technologií, archivace i analýzy (tzv. big data problem). *Edge computing* je přístup, který posouvá výpočty od datových center blíže k místům kde data vznikají [102]. Díky neustále rostoucímu výkonu mikroprocesorů i mikrokontrolérů mají koncová zařízení dostatečné prostředky i k vykonávání komplexních algoritmů. Principy edge computingu jsou stále častěji využívány v oblasti internetů věcí, ovšem nejsou exkluzivní pouze pro tuto oblast. Využívají se v mobilních aplikacích, 5G sítích a multimédiích. Zařízení, které provádí výpočty místo serveru, se nazývá edge node a může se jednat například o senzory, videokamery, chytré telefony, routery nebo jednodeskové počítače.

Druhým pojmem, který je často spojován internetem věcí, je také *Fog computing* (též nazývaný jako *Fog networking* či *fogging*). Myšlenkou tohoto přístupu je snížit zátěž serverů či celých datových center, ovšem pomocí distribuování výpočtů mezi několik horizontálních jednotek [18]. Ty spolu komunikují koordinovaně a sdílí relevantní data.

2

Cíle práce a metodologie

Z úvodu práce je již zřejmé, že internet věcí je natolik obsáhlá a všudypřítomná oblast, že není možné navrhnout jedno zcela univerzální řešení, které by pokrývalo všechny oblasti. IoT architektury se dle rozsahu využití liší dle požadavků na administraci, oprávnění uživatelů, škálovatelnost, zabezpečení, počtu zařízení atd. Z tohoto důvodu cílí práce na IoT architektury malého rozsahu, tedy v rozsahu od lokálních výzkumných experimentů v rámci laboratoře až po jednotky domů a budov.

Hlavním cílem práce je navrhnout open-source architekturu nového IoT řešení pro sběr senzorických dat a automatizaci, která přidává oproti existujícím systémům specifické funkcionality. Jmenovitě se jedná o rozšíření schopností protokolových bran o offline funkcionalitu v podobě pravidel a logování, synchronizaci času mezi integračním middlewarem a branami, podporu přenosu senzorických dat v reálném čase, edge computingu na bráně a koncových prvcích a podpory virtuálních či externích senzorů. Realizace takové architektury vyplní mezery v nedostatcích existujících řešení a bude představovat jeden z možných způsobů řešení vybraných výzkumných problémů. Kompletní implementace navržené architektury je časově velmi náročná a vyžaduje celý vývojářský tým. Z toho důvodu budou implementovány pouze vybrané části této architektury, zejména ty, které potvrdí vyřešení zvolených výzkumných problémů.

Dílčí cíle disertační práce:

- Vymezení základní terminologie a obecného modelu senzorických IoT architektur a různé pohledy na oblast IoT.
- Představení technologií využitelných v oblasti internetu věcí.
- Popis existujících IoT platforem.
- Provedení rešerše současného stavu výzkumu v oblasti internetu věcí a nalezení řešených problémů.
- Porovnání existujících řešení z hlediska nalezených problémů.
- Návrh architektury nového řešení.
- Implementace vybraných softwarových a hardwarových částí nového systému.
- Diskuze a představení výzkumných projektů využívajících nové řešení.

V práci je z metodologického pohledu vyžito množství analytických metod, zejména funkční, klasifikační, systémová a srovnávací analýza. Ty slouží k prozkoumání a rozboru celé problematiky softwarových a hardwarových architektur řešení využívaných v oblasti internetu věcí a sběru senzorických dat.

Text práce je rozdělen do devíti kapitol včetně úvodu a tohoto vymezení cílů. Ve třetí kapitole jsou nejprve definovány základní pojmy používané v IoT architekturách, včetně představení obecného doménového a funkčního modelu a typické fyzické podoby IoT řešení. Poté je navázáno rešerší technologií využitelných a využívaných při realizaci IoT řešení. Práce se dívá na IoT z hlediska nasazení a ekonomického. Dále jsou v následující kapitole představeny existující komerční a open-source řešení z hlediska jejich architektur a vlastností. Následně jsou popsány vybrané současné výzkumné problémy řešené v oblasti internetu věcí, sběru senzorických dat a automatizace. V závěru kapitoly jsou všechny porovnány z hlediska výzkumných problémů.

Čtvrtá kapitola navrhuje celkovou architekturu nového řešení z hlediska hardwarových komponent, datového modelu a jednotlivých vrstev. Pátá kapitola popisuje části ukázkové implementace nejvyšší vrstvy řešení – IoT integračního middleware, šestá kapitola pak subsystém realizující bránu a sedmá kapitola nody, realizující koncová zařízení. Osmá kapitola se zabývá diskuzí možností alternativních implementací architektury, splnění stanovených cílů práce a krátkým představením projektů, které využívají popsané řešení. Závěrečná kapitola obsahuje koncizní shrnutí obsahu práce.

Disertační práce obsahuje množství cizích výrazů a anglicismů, které jsou v některých případech těžko přeložitelné. Pojmy jsou vysvětleny při prvním výskytu v textu a jsou uvedeny také v seznamu zkratek a pojmů s cílem usnadnit čtenáři čtení textu.

3

Analýza současného stavu

V této kapitole je nejprve definován pojem Internet věcí a jeho architektura, následně je navázáno představením funkčního modelu Internetu věcí. Poté je představena typická fyzická architektura IoT platforem a představeny vybrané technologie využitelné či využívané pro jejich realizaci. Dále je ukázáno, jak lze IoT integrační vrstvy klasifikovat a oblast IoT je dekomponována z hlediska nasazení a ekonomie. Další část kapitoly představuje existující IoT platformy a je na ni navázáno uvedením současných vybraných výzkumných problémů řešených v IoT. V závěru jsou platformy srovnány z hlediska výzkumných problémů a toho, jaké jsou přínosy vyřešení těchto problémů.

3.1 Základní pojmy

V úvodu práce bylo řečeno, že pojem „Internet věcí"není zcela jednoznačně definován. Mimo definic z úvodní kapitoly lze zmínit například tyto:

„Internet věcí je paradigma, ve kterém objekty vybavené senzory, aktuátory a procesory komunikují mezi sebou, aby mohly sloužit smysluplnému účelu" [101]

„Internet věcí je nově vznikající síťová nadstavba, která spojuje fyzické zdroje a lidi společně se softwarem." [62]

„Senzory nebo aktuátory, které vykonávající specifickou funkci a které jsou schopné komunikovat s jiným zařízením. Jsou součástí infrastruktury umožňující přenos, skladování, zpracování a přístup ke generovaným datům uživatelům nebo jiným systémům." [29]

„Internet věcí je scénář, ve kterém jsou věci připojeny k internetu prostřednictvím zařízení pro snímání informací za účelem inteligentní identifikace a správy." [118]

„Pojem Internet věcí se používá jako zaštiťující klíčové slovo pro pokrytí různých aspektů souvisejících s rozšířením internetu a webu do fyzické sféry prostřednictvím rozmístění prostorově distribuovaných zařízení se zabudovanou identifikací, schopnosti snímání anebo aktuace." [79]

Další pohled na pojem Internet věcí je, že se jedná o tzv. buzzword, viz publikace [48, 63]. Pojem buzzword definuje Websterův slovník jako „důležitě znějící technické slovo nebo frázi, často malého významu, používané hlavně k tomu, aby zapůsobilo na laiky" [146]. Mnohdy jsou tak v informatice

označovány dlouhodobě využívané přístupy a koncepty, které ovšem neměly konkrétní pojmenování. Za vznikem buzzwordů často stojí média. Mezi současné buzzwordy tak lze zařadit například také strojové vidění, umělou inteligenci, blockchain či quantum computing.

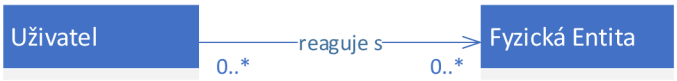
3.1.1 Architektura

Pojem architektura je znám především z oboru stavitelství jako umění či zkušenost navrhování a budování struktur [147]. Softwarovou architekturu lze definovat jako soubor relevantních návrhových rozhodnutí ovlivňující vlastnosti celkové funkčnosti systému [36]. Na architekturu softwaru lze nahlížet z několika pohledů. Software je možné popsat z hlediska jeho struktury, chování, interakce, bezpečnosti, nasazení, ekonomických faktorů atd. Architektura může být vyjádřena několika různými způsoby, od prostého textového popisu, přes tabulky po diagramy. Velmi často je využíván grafický modelovací jazyk UML (Unified Modeling Language). Specifikuje dvě kategorie diagramů: diagramy struktury a diagramy chování [12]. Strukturální diagramy popisují prvky softwarové architektury a patří mezi ně diagramy tříd, komponent, nasazení, objektů (instancí), balíčků, profilů a kompozitní struktury. Diagramy chování popisují, jakým způsobem se má navrhovaný systém chovat. Tyto diagramy lze rozdělit na diagramy případů užití, aktivit, stavů, sekvencí, komunikací, interakcí a časování. Granularita diagramů popisujících architekturu se může lišit dle cílového zaměření. Zatímco softwarový vývojář využije diagram tříd včetně jednotlivých metod a členských proměnných, Dev Ops si vystačí pouze s diagramem síťových interakcí, kde celá komplexní aplikace nebo její část může být zastoupena pouze jako jeden blok.

Oblast Internetu věcí lze dekomponovat a popsat z několika různých pohledů. Nejprve bude představen doménový model, popisující fyzické entity, virtuální entity, uživatele, služby, prostředky a zařízení. Dále bude IoT podrobně rozebrán z hlediska funkcionality, nasazení a ekonomie.

3.2 Doménový model Internetu věcí

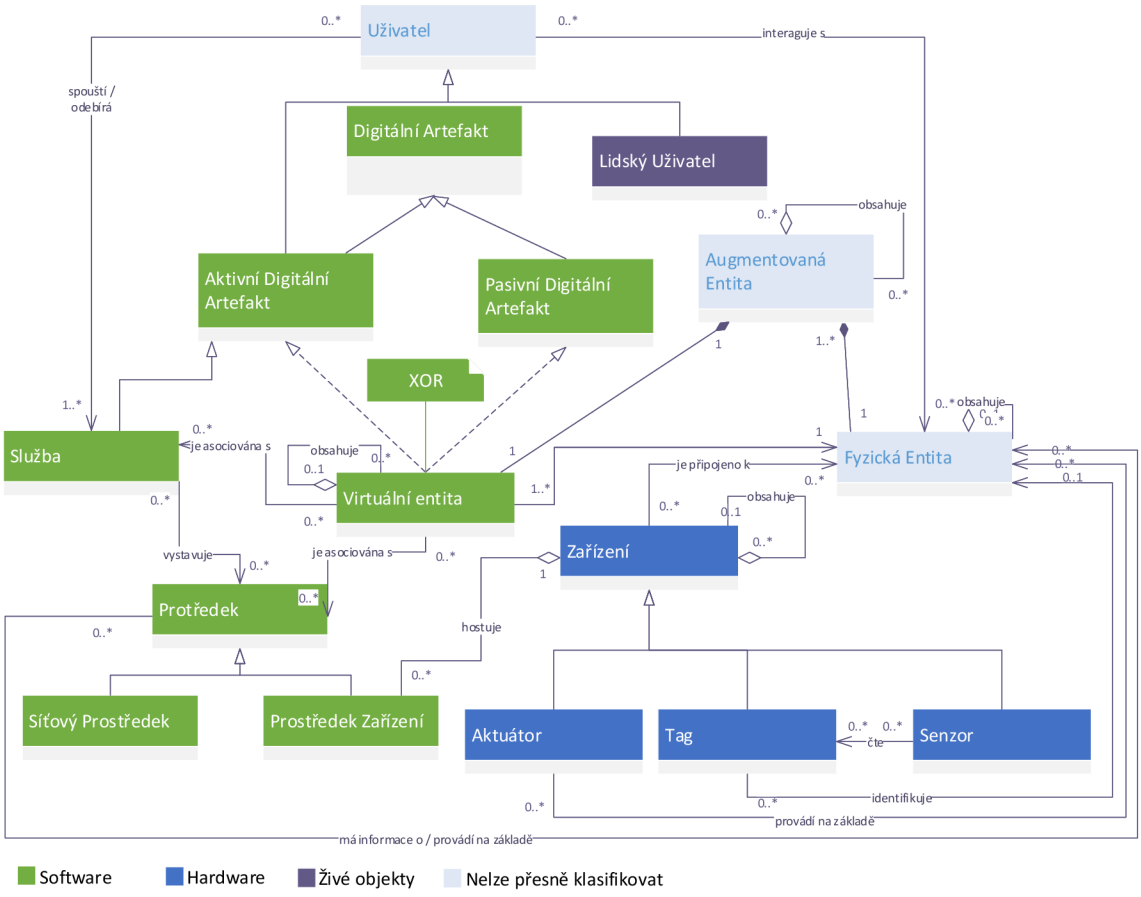
Nejvíce obecný scénář v IoT lze chápat tak, že *Uživatel* potřebuje interagovat s *Fyzickou Entitou* (pravděpodobně vzdálenou), viz obrázek 3.1. Již v tomto krátkém popisu došlo k představení dvou klíčových entit v IoT: *Uživatel* je člověk nebo druh *Digitálního Artefaktu* (například služby, aplikace nebo softwarového agenta) který vyžaduje interakci s *Fyzickou Entitou*. *Fyzickými Entitami* mohou být v podstatě jakékoliv objekty v prostředí, od lidí přes zvířata až k automobilům, od obchodů přes prvky logistických řetězců po počítače, od elektronických spotřebičů po oblečení a podobně.



Obrázek 3.1: Obecná abstrakce IoT interakce (překresleno z [16])

Ve hmatatelném prostředí mohou interakce nastávat přímo (například přesun zboží z lokace A do lokace B). V IoT je ovšem potřebné provádět interakci buď nepřímo nebo skrze prostředníka, tedy zavoláním *Služby*, která buď poskytne informaci o *Fyzické Entitě* nebo bude jednat podle ní. Když je ke službě přistupováno *Lidským uživatelem*, děje se tak skrze klientský software, tedy programu vybaveného uživatelským rozhraním. Z důvodu lepší čitelnosti není klientský software v diagramu 3.2 znázorněn.

V rámci doménového modelu IoT jsou interakce obvykle charakterizovány cílem, který vyžaduje *Uživatel*. *Fyzická Entita* je identifikovatelná část skutečného prostředí a je zájmem *Uživatele*, protože mu umožňuje dosažení jeho cíle.



Obrázek 3.2: UML reprezentace doménového modelu IoT (překresleno z [16])

Fyzické Entity jsou v digitálním světě reprezentovány *Virtuálními Entitami*. Tento termín je také často označován jako virtuální protějšek [97], ovšem využitím slova entita je v obou termínech usnadněno pochopení vzájemných vztahů. Existuje mnoho druhů digitálních reprezentací *Fyzických Entit* – 3D modely, avataři, záznamy v databázi, objekty (nebo instance tříd v objektově orientovaných programovacích jazycích), dokonce i na účty uživatelů sociálních sítí může být nahlíženo jako na reprezentace, protože digitálně reprezentuje určité aspekty vlastníka účtu (fotografie, seznam koníčků, osobní údaje, atd.). Nicméně v kontextu IoT musí *Virtuální Entity* splňovat dvě základní vlastnosti:

- Jsou *Digitálními Artefakty*. *Virtuální Entity* jsou spojeny s jednou *Fyzickou Entitou* a *Virtuální Entita* reprezentuje právě tuto *Fyzickou Entitu*. I když pro každou *Virtuální Entitu* existuje obecně pouze jedna *Fyzická Entita*, je možné, že stejná *Fyzická Entita* může být spojena s několika *Virtuálními Entitami*, například v různých reprezentacích pro každou aplikační doménu. Každá *Virtuální Entita* musí mít přiřazený jednoznačný identifikátor (ID). *Virtuální Entity* jsou *Digitálními Artefakty*, které mohou být klasifikovány jako aktivní či pasivní. Na *Aktivní Digitálním Artefaktu* běží software nebo *Služba*, která může přistupovat k dalším *Službám* nebo *Prostředkům*. *Pasivní Digitální Artefakt* je

nečinným softwarovým elementem, jako jsou například záznamy v databázi, sloužící k digitální reprezentaci *Fyzické Entity*.

- V ideálním případě jsou *Virtuální Entity* synchronizovanými reprezentacemi dané sady aspektů (nebo vlastností) *Fyzické Entity*. To znamená, že příslušné digitální parametry představující vlastnosti *Fyzické Entity* jsou aktualizovány při každé změně této entity. Stejně tak se mohou změny, které ovlivňují *Virtuální Entitu*, projevit ve *Fyzické Entitě*. Například ruční zamknutí dveří může mít za následek změnu stavu dveří v softwaru pro domácí automatizaci na „zamčeno“ a opačně nastavení dveří v automatizačním softwaru na „odemčeno“ vede k odemčení fyzických dveří.

Nyní je vhodné poznamenat, že z diagramu 3.1 může na první pohled vyplývat, že pouze *Lidští Uživatelé* provádí interakci s *Fyzickými Entitami*. Diagram ovšem pokrývá i interakci mezi dvěma stroji, v tomto případě je řídící software prvního stroje *Aktivním Digitálním Artefaktem*, a tedy i *Uživatelem*, a druhý stroj – nebo zařízení ve smyslu doménového modelu IoT – lze modelovat jako *Fyzickou Entitu*. Představený koncept je konceptem tzv. *Augmentované Entity* (rozšířené entity) jako kombinace jedné *Virtuální Entity* a *Fyzické Entity*, s níž je spojena, aby byla zdůrazněna skutečnost, že tyto dva pojmy patří k sobě. *Augmentovaná Entita* je to, co ve skutečnosti umožňuje každodenním objektům, aby se staly součástí digitálních procesů, a tak lze *Augmentovanou Entitu* považovat za „věc“ v Internetu věcí.

Mělo by také být zdůrazněno, že může existovat větší množství typů uživatelů. *Lidský Uživatel* je specializace obecného konceptu. Možné jsou však i jiné druhy uživatelů, například údržbáři, majitelé nebo bezpečnostní pracovníci. Do doménového modelu IoT nejsou zahrnuty různé role, a to ze stejného důvodu, jako nejsou zahrnuty různé typy *Uživatele*. Při vývoji konkrétních architektur je velmi pravděpodobné, že *Uživatelé* převezmou různé role a ty budou odpovídajícím způsobem modelovány, nebo alespoň popsány. Protože se základní taxonomie bude lišit podle řešených případů použití, není zde žádná konkrétní předepsána. Zejména v podnikové sféře, kde jsou role zabezpečení zásadní pro prakticky každý aspekt IoT architektury, lze jako jednu z možností pro modelování rolí nalézt v [94].

Vztah mezi *Virtuální Entitou* a *Fyzickou Entitou* je obvykle dosažen „vložením do“, „připojením“ nebo jednoduše „umístěním“ v těsné blízkosti *Fyzické Entity*, jednoho nebo více *Zařízení* s komunikační technologií, která poskytuje potřebné rozhraní pro interakci nebo získávání informací o *Fyzické Entitě*. Tímto přístupem *Zařízení* rozšiřuje *Fyzickou Entitu* a umožňuje jí, aby byla součástí digitálního světa. *Zařízení* tedy zprostředkovává interakce mezi *Fyzickými Entitami* (které nemají projekce v digitálním světě) a *Virtuálními Entitami* (které nemají projekce ve fyzickém světě), čímž vytváří pár, který se vzájemně rozšiřuje, tj. vzniká výše zmíněná *Augmentovaná Entita*. *Zařízení* jsou tedy technickými artefakty pro přemostění skutečného světa *Fyzických Entit* s digitálním světem internetu poskytováním možností monitorování, snímání, spouštění, výpočtu, ukládání a zpracování. Samotné *Zařízení* může být také *Fyzickou Entitou*, zejména v souvislosti s určitými aplikacemi. Příkladem takové aplikace je management *Zařízení*, jejímž hlavním zájmem jsou samotná *Zařízení* a nikoli entity nebo prostředí, která tato *Zařízení* monitorují.

Z pohledu IoT domény lze *Zařízení* klasifikovat na tři typy:

- *Senzory* poskytují informace, znalosti nebo data o *Fyzické Entitě*, kterou monitorují [148]. Mohou být připojeny nebo zabudovány přímo ve *Fyzické Entitě*, případně umístěny v prostředí a *Fyzickou Entitu* monitorovat nepřímo. Příkladem posledního přístupu může být kamera s podporou rozpo-

znávání tváře. Informace ze *Senzoru* mohou být zaznamenány pro pozdější analýzu (například v úložišti *Prostředku*).

- *Tagy* se využívají k identifikaci *Fyzických Entit*, ke kterým jsou *Tagy* obvykle fyzicky připojeny. Identifikační proces se nazývá „čtení“ a provádí ho specifické senzory, které se obvykle nazývají čtečky. Hlavním účelem *Tagů* je usnadnit a zvýšit přesnost procesu identifikace. Tento proces může být optický (jako v případě čárových kódů a QR kódů) nebo může být založen na radiovém přenosu (jako v případě systémů RFID [82]). Fyzikální princip procesu, stejně jako typy *Tagů*, nejsou pro model domény IoT relevantní, protože tyto technologie se v průběhu času mění a vyvíjejí. Výběr vhodných technologií je ovšem důležitý při implementaci konkrétního systému.

- *Aktuátory* (též *aktory*) mohou měnit stav jednoduché *Fyzické Entity* změnou například rotace, translace, zapnutím nebo vypnutím, případně aktivací či deaktivací funkcionality komplexnějších entit.

Zařízení mohou býti sama o sobě agregacemi dalších *Zařízení* odlišných typů. Například požární hlásič se skládá jak ze *Senzoru* (chemický senzor reagující na přítomnost kouře) tak i *Aktoru* (relé spínající sirénu). V některých případech může *Virtuální Entita* související s rozměrnou *Fyzickou Entitou* záviset na několika (pravděpodobně heterogenních) *Prostředcích* a *Zařízeních* aby bylo možné poskytnout smysluplnou reprezentaci *Fyzické Entity*.

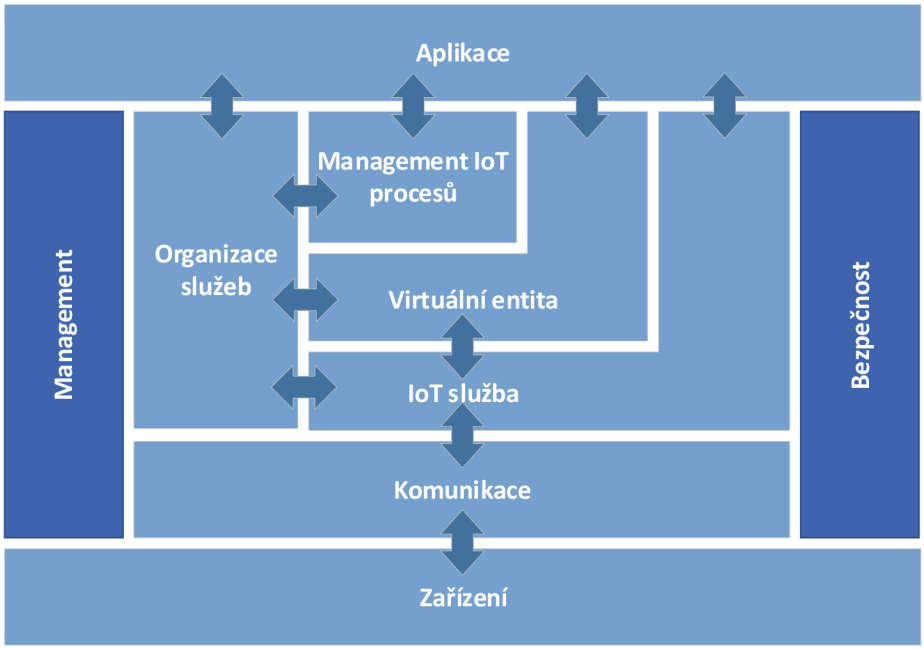
Prostředky jsou softwarovými komponentami, které buď poskytují data, nebo nastavují *Fyzické Entity*. Lze je rozlišit na *Prostředky Zařízení* a *Síťové Prostředky*. Jak naznačuje název, *Prostředky Zařízení* se nacházejí přímo v zařízení, nejčastěji v podobě softwaru *Zařízení* asociovaného s *Fyzickou Entitou*. Zahrnují spustitelný kód používaný k přístupu, zpracování a uložení informací *Senzorů*, nebo kód pro ovládání *Aktuátorů*. *Síťové Prostředky* jsou pak dostupné v síti, například na serverovém back-endu nebo v cloudové databázi. *Virtuální Entita* může být také asociována s *Prostředkem* umožňujícím interakci s *Fyzickou Entitou*, kterou *Virtuální Entita* reprezentuje.

V kontrastu s heterogenními *Prostředky* (implementacemi, které jsou silně závislé na hardwaru *Zařízení*) *Služby* poskytují standardizované a otevřené rozhraní s veškerou nezbytnou funkcionalitou nutnou pro interakci s *Prostředky* či *Zařízeními* asociovanými s *Fyzickými Entitami*. Interakce se *Službou* probíhá přes datovou síť. Na nejnižší úrovni, tedy na rozhraní s *Prostředky* a blíže skutečnému hardwaru *Zařízení*, *Služba* vystavuje funkcionalitu *Zařízení* prostřednictvím hostovaných *Prostředků*. Další *Služby* mohou volat jiné *Služby* nižší úrovně pro poskytování funkcionality vyšší úrovně, například spuštěním aktivity business procesu.

Jelikož právě *Služby* umožňují dostupnost *Prostředků*, výše zmíněné relace mezi *Prostředky* a *Virtuálními Entitami* jsou modelovány jako asociace mezi *Virtuálními Entitami* a *Službami*. S každou *Virtuální Entitou* může být asociováno více odlišných *Služeb* s různou funkcionalitou, jako je získávání informací nebo umožnění provádění aktuačních úkonů. *Služby* mohou být také redundantní, tedy *Služba* daného typu je poskytována v odlišných instancích, například redundantní *Služba* poskytující teplotu od odlišných *Zařízení*. V tomto případě může existovat vícenásobná asociace se stejnou *Virtuální Entitou*.

3.3 Funkční model IoT

Funkční model znázorněný na obrázku 3.3 není přímo svázán s konkrétní technologií, aplikační doménou nebo implementací. Funkční bloky *Aplikace*, *Virtuální entity*, *IoT služba* a *Zařízení* jsou odvozeny z doménového modelu IoT. Požadavkům zúčastněných stran na možnost budování služeb a aplikací nad IoT se zabývají funkční bloky *Management IoT procesů* a *Organizace služeb*. Problém důvěry, bezpečnosti a soukromí je řešen v funkčním bloku *Bezpečnost* a funkční blok *Management* je vyžadován pro správu nebo interakci mezi jednotlivými bloky funkcionalit.



Obrázek 3.3: Funkční model IoT [62]

Funkční model obsahuje sedm vodorovných funkčních bloků (světle modrá), které doplňují dva svislé funkční bloky (*Management* a *Bezpečnost*, tmavě modrá). Tyto svislé bloky poskytují funkcionality vyžadované každou vodorovnou funkční skupinou. Zásady týkající se vodorovných bloků se aplikují nejen na samy na sebe, ale také patří do svislých bloků. Příkladem může být efektivnost bezpečnostních zásad, kde musí být zajištěno, že neexistuje komponenta, která dokáže danou zásadu obejít a poskytnout tak neautorizovaný přístup. V modelu jsou šipkami znázorněny interakce jednotlivých funkčních bloků mezi sebou, s výjimkou svislých funkčních bloků z důvodu přehlednosti. Nyní budou jednotlivé funkční bloky popsány, vyjma funkčních bloků *Aplikace* a *Zařízení*, jelikož jejich vlastnosti jsou velmi obecné a nepřinášejí tak z hlediska architektury informační hodnotu.

3.3.1 Management IoT procesů

Funkční blok *Managementu IoT procesů* se týká integrace business procesů s IoT. Dodržováním standardů a běžných postupů mohou podniky efektivně využívat IoT systémy a vyhnout se problémům při integraci do jejich existující infrastruktury. Blok tedy řeší aspekty specifické pro podnikové business procesy, jako spolehlivost senzorických dat poskytováním informací o *Virtuálních entitách* nebo vyžadovaných schopností zpracování *Zařízení* poskytujících určité *Prostředky* relevantní pro skutečný svět. Aplikace,

které interagují s tímto funkčním blokem, jsou tak odstíněny od IoT specifických nižších vrstev funkčního modelu, což značně redukuje náklady na integraci a tedy zvyšuje adopci IoT systémů [75].

3.3.2 Organizace služeb

Organizace služeb je centrálním funkčním blokem, který se chová jako prostředník pro další funkční bloky. Funkční blok skládá a orchestruje *Služby* s různou úrovní abstrakce, jelikož primárním komunikačním konceptem v IoT jsou právě *Služby*. V rámci IoT architektury spojuje požadavky na *Služby* od funkčních bloků vyšší úrovně jako *Management IoT procesů* či externích aplikací se základními službami vystavujícími *Prostředky* (například služby poskytované bránou). Tím umožňuje vzájemné provázání těchto entit se službami využitím funkčního bloku *Virtuální entity* a provádět překlad požadavků vyšších vrstev (například „Jaká je teplota v obývacím pokoji?“) dolů směrem ke konkrétním službám (například „senzorická Služba 25“). Aby bylo možné dotazovat se *Virtuálních entit* nebo *IoT služeb*, musí být blok *Organizace služeb* zodpovědný za organizování *IoT služeb* a také se zabývat kompozicí a choreografií *Služeb*. Kompozice služeb je centralizovaný koncept v rámci architektury IoT, jelikož *IoT služby* mají často velmi omezené schopnosti z důvodu omezeného výpočetního výkonu a životnosti na baterii, typické pro bezdrátové senzorické sítě nebo embedded *Zařízení*. Kompozice služby pak pomáhá kombinovat více takových základních *Služeb*, aby bylo možné odpovídat na požadavky na vyšší úrovni abstrakce *Služby*[1]. Choreografie služeb je koncept zprostředkovávající *Služby* takovým způsobem, aby mohly komunikovat mezi sebou.

Jak bylo uvedeno v předchozí části, blok *Organizace služeb* je úzce svázán s blokem *Managementu IoT procesů*, jelikož umožňuje (business) procesům nebo externím aplikacím nalézt a alokovat *Služby*, které pak mohou provádět procesní kroky nebo být různými způsoby integrovány s externími aplikacemi. *Organizace služeb* funguje jako základní aktivátor *Managementu IoT procesů*.

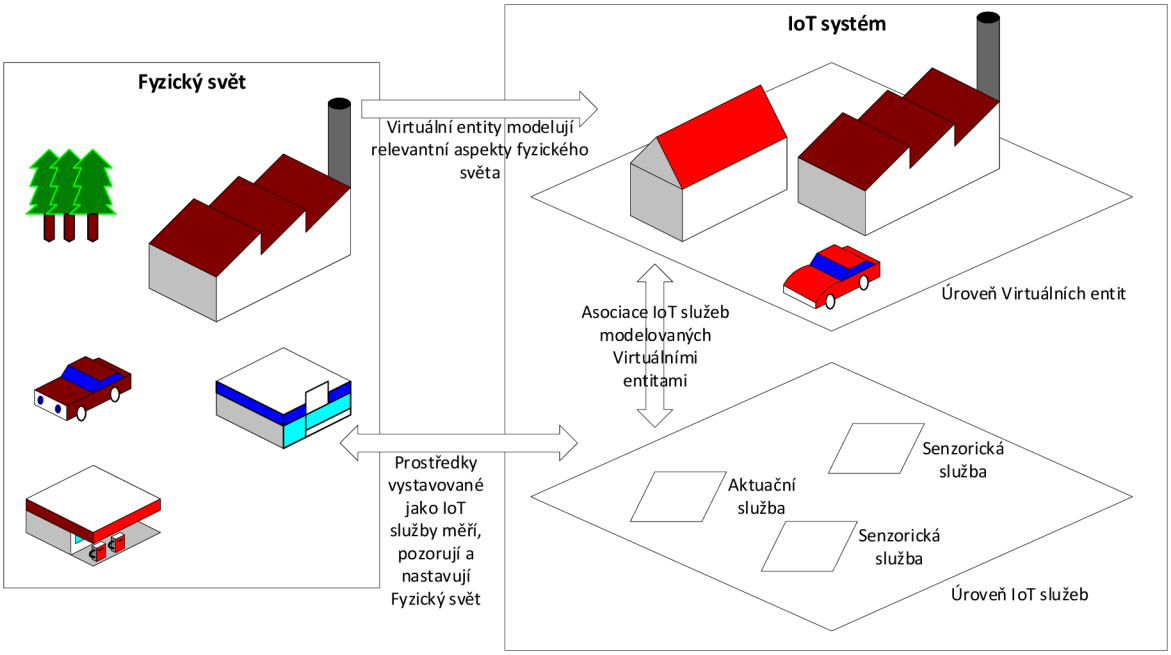
3.3.3 Virtuální entita a IoT služba

Funkční bloky *Virtuální entity* a *IoT služby* zahrnují funkce, které se vztahují k interakcím a úrovním abstrakce. Obrázek 3.4 ukazuje úrovně abstrakce bloků a jak spolu souvisí. Na levé straně je vyobrazen fyzický svět, ve kterém je množství *Senzorů* a *Aktuátorů* zachycujících a umožňujících změny určitých aspektů fyzického světa. *Služby* asociované se *Senzory* a *Aktuátory* jsou vystavovány jako *IoT služby* na úrovni stejnojmenného funkčního bloku. Interakcemi mezi aplikacemi a *IoT službami* na této úrovni abstrakce mohou být například: „Sděl hodnotu Senzoru 42“ nebo „Nastav Aktuátor 15 na stav vypnuto“. Aplikace mohou s těmito *Službami* komunikovat smysluplným způsobem, pokud již znají sémantiku hodnot. Tedy když Senzor 42 vrátí hodnotu 22, aplikace musí být naprogramována nebo nakonfigurována takovým způsobem, aby věděla, že se jedná o teplotní senzor s jednotkami stupňů Celsia. Na této úrovni není žádná sémantika zakódována v samotné informaci, ani IoT systém tyto informace nemá, musí tedy být předem sdíleny mezi *Senzorem* a aplikací.

Zatímco interakce na úrovni *IoT služeb* jsou užitečné pro určité sady aplikací, které jsou programovány nebo konfigurovány pro konkrétní prostředí, existují jiné aplikace, které chtějí příležitostně využívat vhodné *Služby* v případně měnícím se prostředí. Pro tyto typy aplikací a především *Lidské Uživatele* jsou k dispozici *Virtuální entity* modelující vyšší úrovně aspektů fyzického světa, a tyto aspekty mohou být

[1] Například kombinace *Služby* měřící relativní vlhkost prostředí a *Služby* měřící teplotu mohou sloužit jako vstup pro kontrolu mikroklimatu.

využity k objevování *Služeb*. V tomto případě by ukázková interakce aplikace s IoT systémem vypadala například takto: „Sděl mi teplotu v obývacím pokoji“ a „Vypni klimatizaci v obývacím pokoji“. K podpoře interakcí s úrovní *Virtuálních entit* musí být mezi *IoT systémem* a *Virtuálními entitami* modelována vazba ve formě asociací, například asociace obsahující informaci, že teplota v obývacím pokoji je poskytována senzorem s identifikátorem 42.



Obrázek 3.4: Vztah fyzického světa, virtuální entity a IoT služby (překresleno z [16])

Funkční blok *Virtuálních entit* obsahuje funkcionality pro interakci s IoT systémem na základě samotných *Virtuálních entit* a také nezbytné funkcionality pro detekci a vyhledávání *Služeb* poskytujících informace o entitách nebo umožňují interakci. Navíc blok musí obsahovat prostředky pro správu asociací, dynamické vyhledávání nových asociací a monitorování jejich validity. Tyto funkce mohou být spouštěny změnou mobility *Fyzických entit* a nebo *Zařízeními*. Funkční blok *IoT služeb* obsahuje veškerou funkcionalitu související s úrovní *IoT služeb*.

3.3.4 Komunikace

Funkční blok *Komunikace* abstrahuje různorodá interakční schémata odvozená z množství komunikačních technologiích využitelných v IoT a poskytuje jednotné rozhraní pro funkční blok *IoT služeb*. Poskytuje prostředky pro instancování a správu toku informací na vyšší úrovni. V potaz se musí brát reprezentace dat, řešení dostupnosti spojení, identifikace, správa sítě a specifické vlastnosti jednotlivých zařízení a komunikačních technologií.

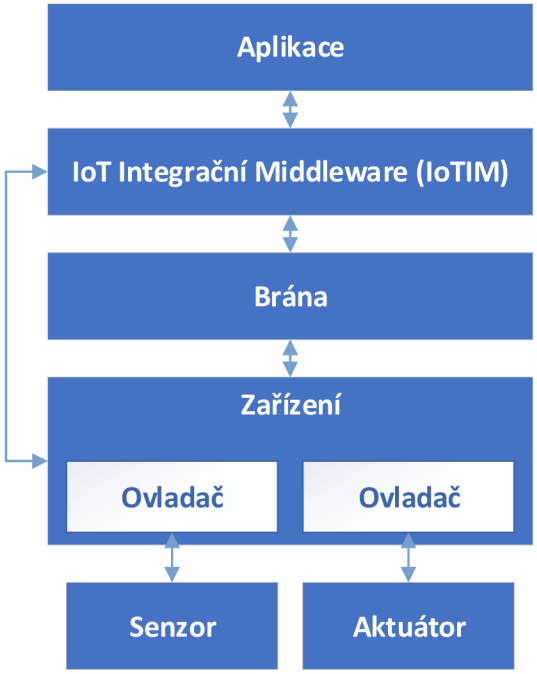
3.3.5 Management

Všechny funkcionality potřebné k řízení IoT systému jsou zahrnuty ve funkčním bloku *Management*. Obsahuje informace o konfiguraci ostatních bloků, nastalých chybových stavech, reportech, členech a stavů.

3.3.6 Bezpečnost

Funkční blok *Bezpečnost* je zodpovědný za dodržování zabezpečení a soukromí v IoT systému. Zajišťuje prvotní registraci klienta do systému bezpečným způsobem, čímž je zajištěno, že pouze ověření klienti mohou přistupovat k službám poskytovaných systémem. Blok musí být také zodpovědný za ochranu privátních parametrů uživatele, tedy zajišťuje to, že identita uživatele zůstane skryta, když je přistupováno k *Prostředku* nebo *Službě*. Tento blok zároveň provádí detailní logování pro možnost auditů. Bezpečnostní blok dále zajišťuje, že k legitimní interakci dochází pouze mezi bloky, které jsou oprávněny vzájemně komunikovat, nebo které si navzájem důvěřují. Toho je zajištěno buď pomocí sady autorizačních funkcí, nebo pomocí modelu důvěry a reputace, který dokáže identifikovat důvěryhodné bloky.

3.4 Typická architektura IoT řešení



Obrázek 3.5: Bloková IoT architektura [45]

Na obrázku 3.5 je blokově znázorněna typická architektura systému Internetu věcí. Skládá se z šesti odlišných částí, a to senzoru, aktuátoru, zařízení, brány, integračního middleware a aplikace.

3.4.1 Senzor a aktuátor

Senzor a aktuátor byl již definován v doménovém modelu IoT. Oproti němu si lze povšimnout, že IoT platformy nerozlišují mezi senzorem a tagem. Z datového pohledu jsou totiž tag a senzor identické. Obojí poskytuje informace směrem od nejnižší vrstvy zařízení k nejvyšší, tedy aplikacím.

Jedna veličina může být měřena několika různými typy senzorů. Praktickým příkladem může být měření teploty prostředí. Nejednodušším senzorem je termistor, tedy součástka měnící svůj elektrický odpor v závislosti na teplotě. Termistor může být připojen jedním vývodem k zemi a v sérii připojen přes

pevný rezistor k napájení. Následně je snímáno napětí na odporovém děliči, a dle hodnoty odporu rezistoru a charakteristiky termistoru přepočteno na teplotu. Druhou součástkou může být analogový teplotní senzor, který výše zmíněný termistor a rezistor integruje do jednoho pouzdra. Po přivedení napájení lze již měřit napětí a dle technické specifikace senzoru jej přepočítat na teplotu prostředí. Dalším senzorem pak může být digitální teploměr, který obsahuje termistor, převodník a řadič sběrnice. S takovým senzorem se pak komunikuje pomocí sběrnice dle povelů definovaných ve specifikaci senzoru.

Tabulka 3.1: Vybrané typy senzorů a aktuátorů [autor]

Fyzikální princip	**Senzor**	**Aktuátor**
Elektrický	Ampérmetr Voltmetr Wattmetr Fotorezistor Fototranzistor	Tranzistor SSR
Mechanický	Spínač Přepínač Tlačítko Potenciometr Vibrační senzor Anemometr Bimetal	
Elektromechanický	Akcelerometr Gyroskop Magnetometr Mikrofon	Relé Motor Solenoid Reproduktor
Elektrochemický	CO CO_2	
Elektrooptický	Prachové částice CO_2 Optobrána	

V tabulce 3.1 jsou vypsány vybrané typy senzorů a aktuátorů, které mohou být relevantní z hlediska IoT projektů. Výpis se nesnaží být kompletní, jelikož je v současné době k dispozici několik desítek až stovek typů senzorů a aktorů, které jsou pro IoT relevantní, jelikož jako senzor může být považován například mobilní telefon, automobil nebo člověk.

3.4.2 Zařízení

Zařízení (z anglického výrazu Device) je hardwarová komponenta, která je spojena metalicky či bezdrátově se senzory nebo aktory, případně mohou být senzory či aktory integrovány jako přímá součást zařízení. Zařízení musí být dále vybaveno procesorem a úložištěm pro běh softwaru. Úkolem softwaru je navazování a komunikace buď s IoT integrační vrstvou nebo s bránou. Druhým úkolem je vyčítání stavů senzorů nebo nastavování aktorů pomocí ovladačů. Ovladač je také softwarem běžícím na zařízení a poskytuje uniformní přístup k heterogenním senzorům či aktorům.

Dle komplexnosti samotného senzoru jsou zařízení vybavena buď mikrokontrolérem nebo mikroprocesorem. Mikrokontrolér je jednočipový počítač se zabudovanou centrální procesní jednotkou (CPU), operační pamětí pro data a non-volatilní pamětí pro program. Programuje se nejčastěji v jazyce C a kód se

provádí přímo na hardwaru. Pokud je využit operační systém, jedná se o operační systém reálného času se základní podporou úloh. Výhodou mikrokontrolérů je nízká cena, determinismus prováděného kódu a integrace různých sběrnic a komunikačních technologií. Současné mikrokontroléry obsahují CPU s taktem až stovky MHz, stovky kilobajtů operační paměti a jednotky megabajtů non-volatilního úložiště. Naopak mikroprocesor obsahuje pouze CPU, případně vyrovnávací paměti cache. Operační paměť i non-volatilní paměť je realizována jako externí součástka nebo soustava součástek propojených sběrnicemi. I když je možné zajistit běh kódu na mikroprocesoru bez operačního systému, tak v reálném prostředí je operační systém vždy přítomen. Typickým výrobcem mikroprocesorů je například společnost AMD a běžným operačním systémem jsou pak Windows nebo GNU/Linux. Výhodou mikroprocesorů je řádově vyšší výkon i kapacita pamětí, nevýhodou pak v kontextu IoT zařízení komplikovanější hardware, čas vývoje a vyšší provozní náklady a spotřeba. Většina IoT zařízení využívá mikrokontrolér, ovšem existují i výjimky, zejména multimediální zařízení jako kamery, automobily nebo radary.

Na obrázku 3.6 je znázorněno zařízení společnosti nke Watteco využívající mikrokontrolér, vybavené kombinovaným senzorem teploty a relativní vlhkosti prostředí a komunikující s IoTIM využitím bezdrátové technologie SigFox.

Obrázek 3.6: IoT zařízení se zabudovaným senzorem teploty a relativní vlhkosti [134]

3.4.3 Brána

Pokud zařízení obsahuje technologie a protokoly pro přímou komunikaci s IoT integračním middlewarem, není brána vyžadována. V praxi se jedná o přenosové technologie založené na IP protokolu. Pokud zařízení takové technologie neobsahuje, musí být v systému přítomna brána. Jejím cílem je provádět obousměrný překlad komunikačního protokolu. Když IoT integrační middleware vyžaduje zjištění stavu senzoru, je tento pokyn odeslán vybraným IP protokolem bráně. Ta požadavek zpracuje a vyšle pokyn zařízení, zpracuje odpověď a odešle jej. Případně může brána obsahovat aplikační logiku, která dokáže stavy senzorů vyčítat autonomně a odesílat vyšší vrstvě nové stavy, a to buď při jakékoliv změně nebo dle konfigurovatelných prahových limitů. Obdobně probíhá i nastavení aktoru. IoT integrační vrstva vyšle požadavek IP protokolem, brána jej přeloží a nastavuje aktor. IoT vrstva může být zpětně informována o úspěšném nebo neúspěšném výsledku této operace. Fyzická podoba IoT brány je znázorněna na obrázku 3.7 a konkrétně se jedná o výrobek společnosti Aaeon, model SRG-3352 vybavený ARM procesorem s taktem 800 MHz, 1 GB operační paměti a konektivitou zahrnující Ethernet, LTE, NB-IoT či USB a RS-485.

Obrázek 3.7: IoT brána [135]

3.4.4 IoT integrační middleware

IoT integrační middleware (dále v práci zkráceno jako IoTIM) slouží jako integrační vrstva pro odlišné typy senzorů, aktuátorů, zařízení, bran a aplikací. Provádí čtyři základní operace, a to přijímaní dat z připojených zařízení, zpracování přijatých dat, poskytování přijatých dat připojeným aplikacím a správu zařízení [20]. Příkladem zpracování přijatých dat může být vyhodnocování pravidel a nastavování aktuátorů na základě jejich výsledku. IoTIM se nemusí omezovat pouze na tuto nezbytnou funkcionalitu, může například dále obsahovat databázi pro perzistenci naměřených dat, nástroje pro grafickou reprezentaci dat a také vystavovat aplikační rozhraní (API) pro komunikaci s vyššími vrstvami.

Zařízení komunikuje buď přímo s IoTIM (pokud disponuje vhodnou komunikační technologií, jako je IP skrze Ethernet nebo WiFi) a korespondujícím transportním protokolem, například MQTT nebo HTTP. Tyto technologie budou detailně představeny v následující podkapitole. Pokud zařízení těmito technologiemi nedisponují, musí využívat prostředníka v podobě brány.

3.4.5 Aplikace

Posledním a nejvyšším blokem IoT architektury je aplikace, tedy software využívající aplikační rozhraní IoTIM pro získání přehledu o fyzickém stavu prostředí. Po IoTIM vyžaduje stavy senzorů a může na ně reagovat nastavením aktoru. Typickým příkladem takové aplikace je grafické uživatelské rozhraní, které slouží k vizualizaci aktuálního stavu měřeného prostředí a k jeho managementu. Dalším příkladem může být komplexní systém pro regulaci teploty prostředí. Další kategorií mohou být takové aplikace, které jsou určeny k výzkumu. Nad získanými daty ze senzorů provádějí například statistickou analýzu či aplikují algoritmy strojového učení. Často podceňovanou kategorií jsou aplikace pro monitoring stavu systému, tedy takové, které upozorňují na problémy s branami, zařízeními či na chybné stavy senzorů. Tyto aplikace jsou obzvláště důležité při přechodu z kontrolovaného laboratorního prostředí do testovací provozu v reálném prostředí a pomohou identifikovat řadu chyb, které v laboratorním prostředí nenastanou [110].

3.5 Technologie využitelné v IoT

Předtím, než budou představeny existující IoT platformy a současné výzkumné problémy, je vhodné představit technologie, protokoly a standardy které se při realizaci IoT řešení využívají. Při analýze byly využity publikace [43, 6, 78, 7, 21]. Jsou představeny pouze vybrané technologie, kompletní vyčerpávající popis vydal na několik knižních publikací.

3.5.1 Metalické komunikační technologie

Komunikační technologie se využívají pro sdílení informací mezi dvěma a více zařízeními. Lze je použít i pro komunikaci mezi komponentami v zařízení, ale nejsou k tomu primárně určeny. I přes nevýhody kabelové komunikace v podobě ceny kabeláže, nutnosti instalace, krimpování a testování kabelů, mají tyto technologie na trhu stále své místo. Zatímco některé bezdrátové technologie lze velmi snadno rušit, průmyslové kabelové technologie umožňují komunikaci na stovky metrů spojů se signifikantně vyšší spolehlivostí, a to i ve výrobních halách, kde je úroveň elektrického rušení zvýšená kvůli výrobním strojům.

RS-232

Technologie RS-232 byla představena již v roce 1960 a její poslední iterace, standard RS-232C , pak v roce 1969 [35]. Specifikováno je pouze diferenciální napětí v rozsahu ±5 V až ±25 V a komunikace bod-bod. Komunikační protokol a přenosová rychlost mohou být voleny libovolně. V osobních počítačích se dříve RS-232 využívalo k připojení modemů, tiskáren, myší a terminálů. Dnes je již u PC kompletně nahrazen univerzálním rozhraním USB, ale stále se využívá u celé řady IR přijímačů, pokladních systémů, ovládání multimediálních zařízení a v průmyslových strojích. Ke komunikaci využívá buď 3 vodiče při asynchronní komunikaci nebo 5 vodičů pro řízenou komunikaci master-slave.

RS-485

Stejně jako RS-232, rovněž RS-485 [77] nespecifikuje typ komunikačního protokolu, pouze napěťové úrovně, typ kabeláže a strukturu sítě. Komunikace probíhá jako master-slave po fyzické sběrnicové topologii. Ke komunikaci se využívá kroucený pár při poloduplexu a dva páry při full-duplexu. V obou případech je mezi páry diferenciální buzení ±12 V. Při polovičním duplexu vyšle master příkaz a přepíná transceiver do přijímacího režimu. Všechna slave zařízení jsou trvale v přijímacím režimu a to, kterému je zpráva určena, přepíná svůj transmitter do vysílacího režimu a odesílá odpověď. Ve full-duplex režimu je jedna linka trvale nastavena pro příjem příkazů pro slave zařízení a druhá pro odpovědi. Na jedné sběrnici může být připojen pouze jeden master a až 31 slave, při maximální délce kabeláže 1 200 metrů při rychlosti 100 kbit/s. Když komunikace probíhá maximální přenosovou rychlostí 10 Mbit/s, klesá vzdálenost na desítky metrů.

CAN bus

Sběrnice CAN (Controller Area Network) se od výše zmíněných odlišuje tím, že není pouze přenosovou specifikací, ale komunikačním protokolem s definovanou fyzickou a přístupovou vrstvou [105]. Sběrnice vznikla pro komunikaci v automobilech, kde se využívá úspěšně dodnes. Dále se začlenila mezi automatizační technologie. CAN nod se skládá z řadiče a kontroléru, přičemž může jít o dva oddělené obvody.

Řadič obsahuje fyzický výstup pro komunikaci a kontrolér řeší obalování uživatelských dat do rámců, detekci kolize a prioritizaci. Fyzicky je CAN zapojena jako sběrnice a logicky neobsahuje žádný řídící prvek, všechny nody jsou si rovny. Každý kontrolér musí mít od uživatelské části přidělen identifikátor. Datová zpráva může být dlouhá maximálně 8 bajtů. Zpráva s nízkou prioritou bude mezi dvěma nody doručena až po doručení všech zpráv s vysokou prioritou. Zprávy se odesílají všem nodům, a ty na ně v uživatelské části mohou, ale i nemusí reagovat.

Ethernet

Prvotní verze ethernetu využívaly sběrnicovou topologii skrze koaxiální kabeláž při rychlosti 10 Mbit/s. Od devadesátých let se ustanovil standard hvězdicové topologie s přepínacím prvkem, konektory RJ-45 a čtyřmi kroucenými páry metalického kabelu. Poslední specifikace stanovuje po metalickém médiu přenosovou rychlost 10 Gbit/s. Ethernet využívají stolní počítače, notebooky, routery, síťová úložiště, embedded zařízení, multimediální prvky a mnoho dalších. Ethernet je implementací linkové vrstvy TCP/IP modelu.

DALI

Technologie Digital Addressable Lighting Interface [149] (DALI) slouží výhradně pro ovládání osvětlení. Využívá dva vodiče s diferenciálním napěťovým buzením a přenosovou rychlostí 1 200 bps. Podrobnou specifikaci komunikačního protokolu definuje norma IEC 62386. Komunikaci orchestruje kontrolér, který umožňuje podružné prvky připojit buď ho sběrnicové nebo hvězdicové topologie. DALI se kvůli omezení pouze na svítidla používá nejčastěji v kombinaci s dalšími technologiemi.

KNX

KNX je otevřený standard [150], který přímo vznikl pro účely průmyslové automatizace. Na trhu jsou dostupné moduly pro osvětlení, termostaty, stykače, ovladače rolet, environmentální senzory atd. Podporuje čtyři různé způsoby komunikace:

- **TP1** – komunikace pomocí krouceného páru s diferenciálním buzením, poloviční duplexem a přenosovou rychlostí 9 600 bps.
- **PL 110** – komunikace přes síťové rozvody modulované na frekvenci 110 kHz a přenosovou rychlostí 1 200 bps.
- **RF** – bezdrátová komunikace v pásmu 868,3 Mhz.
- **IP** – obalení KNX telegramů (zpráv) do IP paketů s využitím technologií Ethernet nebo WiFi.

Jednotlivé komunikační technologie lze kombinovat přes konvertory. Každé zařízení musí mít přidělenou 16-bitovou adresu a jedna KNX instalace může obsahovat až 57 375 zařízení. Doporučená topologie využívá na nejvyšší vrstvě IP komunikaci ve stromovém zapojení. Jednotlivé místnosti jsou vybaveny moduly s protokolem TP1. Při návrhu topologie je nutné brát zřetel na limit 50 telegramů za sekundu při využití TP1, PL110 a RF komunikace. Při použití IP komunikace jsou limitem schopnosti zvolených konvertorů.

3.5.2 Bezdrátové technologie

Během poslední dekády došlo k rapidní proměně bezdrátových technologií. Stále existují technologie snažící se o co nejvyšší přenosové rychlosti, ale také vzniklo mnoho nových technologií s jiným cílem. Tyto technologie slouží především pro senzorické IoT aplikace, kdy je objem dat velmi malý (v řádu jednotek bajtů), a je vyžadována možnost dlouhodobého provozu na baterii, za cenu výrazně snížené propustnosti.

Sigfox

Sigfox [151] je francouzská společnost a zároveň také název bezdrátové přenosové technologie, kterou společnost vyvíjí. Přenosová rychlost se přesně neuvádí, ale z jednoho koncového prvku je možné odeslat k základové stanici zprávu o délce až 12 bajtů každých deset minut. Pokud by se zařízení pokoušelo data odesílat častěji, bude automaticky odpojeno od sítě. Směrem od základové jednotky jsou limity k zařízení ještě přísnější, lze poslat pouze čtyři zprávy denně a jejich velikost je maximálně 8 bajtů.

Tyto údaje se mohou zdát jako velmi limitující, ale pro některé projekty z oblasti Internetu věcí mohou být naprosto dostatečné. Jedná se o senzorické aplikace, příkladem budiž odesílání informace o teplotě, vlhkosti nebo například úrovni oxidu uhličitého. Zde je frekvence měření jednou za deset minut více než dostatečná. Díky těmto omezením a také faktu, že technologie pracuje ve frekvenčním pásmu 868 MHz (Evropa) a 902 MHz (USA), jsou Sigfox zařízení velmi úsporná a dokáží fungovat z baterie po dobu několika let. Tento údaj je samozřejmě také závislý na spotřebě připojených senzorů. Vzdálenost koncového prvku od základové stanice může být v otevřeném prostranství až 50 kilometrů.

Zatímco vývojové kity pro koncové prvky je možné pro vývoj zakoupit, základové stanice vždy v konkrétním státě poskytuje partnerský operátor. V české republice se jedná o společnost SimpleCell, která využívá páteřní linky T-Mobile. Data z každého koncového prvku putují přes tyto stanice do cloudového úložiště Sigfox. Odtud je poté může vývojář získat. Datové zprávy v sítí Sigfox jsou zpoplatněny, jedná se ovšem o ceny v řádech desetin haléře za datovou zprávu.

LoRaWAN

Technologie LoRaWAN [152] (Long Range Wide Area Network) sdílí s technologií Sigfox stejnou základní myšlenku. Celý standard spravuje LoRa aliance. Jedná se také o nízkopříkonové koncové prvky navržené pro levnou a zabezpečenou komunikaci pro Internet věcí. Několik rozdílů zde ovšem existuje. Na rozdíl od Sigfoxu počet zpráv není omezený a přenosová rychlost je volitelná od 300 b/s až po 50 kb/s [3]. Je třeba ale brát na vědomí, že při maximální přenosové rychlosti se výdrž zařízení na baterii rapidně sníží z řádů let na řády týdnů. Dosah v terénu je až 40 km, v příměstských oblastech pak klesá na 15 km a v hustě zastavěných městských prostředí se pohybuje v rozsahu 2 až 5 km. Dále není omezená komunikace směrem od brány (u technologie Sigfox nazývána jako od základové stanice ke koncovým prvkům). Stejně jako u technologie Sigfox existuje v České republice poskytovatel bran, jsou jím České radiokomunikace. Rozdílem je, že je možné zakoupit a využít svou vlastní bránu, která bude přijímat data z koncových prvků místo těch od Českých radiokomunikací. Při použití vlastní brány je ovšem doporučeno brát na zřetel sdílení pásma a omezit vysílací výkon, jinak může dojít k degradaci signálu. Koncové prvky s rádiem LoraWAN se dělí do tří tříd, u všech je komunikace obousměrná:

Třída A – Koncový prvek může přijmout zprávu poté, co nějakou odeslal směrem k bráně.

Třída B – Krom přijímání zpráv stejně jako v třídě A mají tato zařízení specifikovány intervaly, kdy mohou přijmout zprávu od brány.

Třída C – Příjem zpráv je otevřen téměř nepřetržitě, nelze přijímat pouze během vysílání k bráně.

NB-IoT

Zkratka NB-IOT [153] znační NarrowBand Internet of Things, neboli úzkopásmový Internet věcí. Záměrem této technologie je tedy obdobný jako u LoRaWAN a Sigfox. Za technologií stojí projekt 3GPP, tedy autoři vysokorychlostní technologie LTE, která bude popsána níže. S tímto standardem sdílí několik vlastností jako jsou přenosová pásma, kterých je celkem 76 ve frekvenčním rozsahu od 7 do 900 MHz. Přenos mezi koncovým prvkem a základnovou stanicí je obousměrný, ale s polovičním duplexem. Šířka přenosového pásma je pouze 180 kHz a dokumentace uvádí přenosovou rychlost až 50 kbps. Maximální vzdálenost je velmi obdobná jako u předchozích technologií – na otevřeném prostranství se jedná až o 50 km. Stejně tak je to s výdrží koncového prvku na baterie, kde se při střídmé frekvenci odesílání v řádů jednotek minut pohybuje výdrž v řádu let.

Stejně jako u Sigfoxu není možné zakoupit vlastní základní stanice, ovšem celostátní provoz není jako v síti Sigfox vázán na jednoho monopolního poskytovatele. V České republice v současné době nabízí 100% plošné venkovní pokrytí a 94% vnitřního pokrytí mobilní operátor Vodafone. Vlastní síť také poskytují v ČR T-Mobile a O2. Pro provoz NB-IoT je nutné využít buď standardní nebo virtuální SIM karty. Velkou výhodou této technologie ve srovnání se SigFox a LoRaWAN [65, 121] je, že počet downlink a uplink zpráv není omezen a je možné si vybrat zařízení, které umožňuje data odesílat neustále. Limitem je pak samozřejmě datový balíček poskytovaný operátorem a je třeba brát na zřetel výdrž baterie. Při neustálé komunikaci bude výdrž velmi nízká.

Bluetooth Low Energy

Technologie Bluetooth je spravována skupinou Bluetooth Special Interest Group (SIG) [154], kterou založily v roce 1998 společnosti Ericsson, IBM, Intel, Toshiba a Nokia. Jejich prvotním záměrem bylo vytvořit bezdrátovou přenosovou technologii s hvězdicovou topologií (tedy jeden centrální prvek a několik klientů) pro digitální přenos zvuku. Do roku 2011 platilo pravidlo, že každá nová iterace této technologie přinesla navýšení přenosové rychlosti. U verze 4.0 došlo ke změně a spolu s původní technologií Bluetooth (nyní nazývána jako Bluetooth Classic) vznikla druhá technologie Bluetooth Low Energy (BLE; také nazývána Bluetooth Smart). S původní technologií sdílí pouze přenosové pásmo 2,4 GHz. Kódování, šířka pásma a přenosová rychlost jsou odlišné. Zatímco Bluetooth Classic cílí na vyšší přenosové rychlosti a výdrž na baterii není brána jako velmi omezující faktor, BLE se zaměřuje na senzory a nositelnou elektroniku. Senzorická data potřebují řádově nižší přenosové rychlosti [106]. Dosah obou technologií je obdobný a činí přibližně 15 metrů. Maximální přenosová rychlost BLE je 250 kbit/s, ovšem protože je použit velmi propracovaný systém služeb, charakteristik a deskriptorů, pohybuje se reálná propustnost na úrovni 19 kbit/s. Technologie definuje centrální a periferní zařízení. Centrální zařízení dovoluje prohledávat okolí přijímače, hledat a navazovat spojení s periferiemi. Periferie pak odesílají nebo přijímají data komunikací s centrálním zařízením. Centrální zařízení může být například chytrý telefon a periferie pak chytré hodinky nebo fitness náramek.

U poslední verze standardu 5.0 byla uvedena velmi očekávaná vlastnost, a to mesh topologie [25], kdy již neplatí striktně hvězdicová topologie. Centrální prvek je stále jediný, ale pokud od libovolného klienta neexistuje přímá cesta, může se zpráva doručit přes n dalších klientů [41], pokud je mezi nimi cesta. Mesh protokol pracuje dynamicky a zlepšuje tak pokrytí i pro mobilní periferie, ovšem snižuje maximální propustnost.

Wi-Fi

Jedná se o nejrozšířenější technologii pro komunikaci v lokálních sítích s cílem poskytnout bezdrátovou alternativu místo ethernetu, detailně popsanou IEEE standardy 802.11.X [155]. Implementuje linkovou vrstvu TCP/IP modelu. V minulosti využívala pouze pásmo 2,4 GHz, v dnešní době také pásma 5 GHz, 5,9 GHz, 60 GHz a také 900 MHz. Wi-Fi v pásmu 2,4 GHz dnes trpí zejména v hustě zastavěných oblastech problémem rušení a překryvu pásem. Při základní rychlosti 11 Mbit (neboli standardu 802.11b s kódováním DSSS) je totiž možné využít pouze 4 nepřekrývající se kanály, protože Wi-Fi specifikuje celkem 14 kanálů. U novějšího standardu 802.11g je počet nepřekrývajících se kanálů také 4, protože je ale použito efektivnější OFDM kódování, přenosová rychlost je až 54 Mbit. V posledním standardu, který využívá 2,4 GHz pásmo při přenosové rychlosti až 600 Mbit se jedná pouze o dva nekolidující kanály. Tyto limity se zdály při návrhu technologie jako dostatečné, protože ale zejména v panelových bytech má každá domácnost vlastní router (i když sdílejí centrální konektivitu), nastává zde velká degradace výkonu. Proto nová koncová zařízení čím dál častěji komunikují v pásmu 5 GHz.

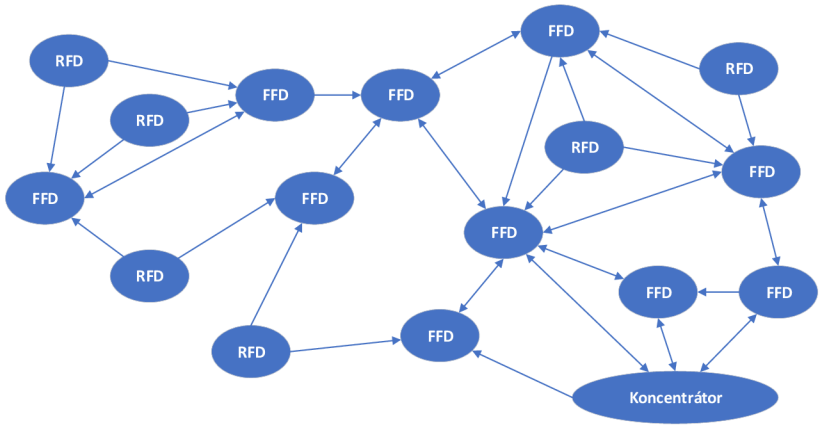
LTE

Long-term Evolution [153] (zkráceno jako LTE) označuje standard pro vysokorychlostní bezdrátovou komunikaci mobilních zařízení se základní stanicí. Využívají se buď základy technologie GSM/EDGE nebo nyní stále častěji UMTS/HSPA. Stejně jako NB-IoT je spravován 3GPP uskupením, ovšem zaměření LTE je velmi odlišné. Cílem je poskytnout co nejvyšší přenosové rychlosti pro chytré telefony, laptopy, tablety, případně nositelnou elektroniku a zpřístupnit stejné možnosti jako při připojení skrze Wi-Fi. Prvotní specifikace LTE byla často mylně označována jako 4G připojení, i když kladeným požadavkům standardu nevyhovovala. Standard byl představen v roce 2007 a nabízel přenosové rychlosti 144 MBit/s downlink a 50 MBit uplink. V současné době probíhá globální přechod na verzi 5G, s přenosovými rychlostmi v řádu stovek megabitů za sekundu.

ZigBee

ZigBee [156] patří do kategorie PAN (Personal Area Network) a spadá pod IEEE standard 802.15.4 od roku 2004. Do množiny 802.15 spadá mimo jiné i Bluetooth. Pracuje v bezlicenčním pásmu 868 MHz, případně v pásmech 902–908 MHz, dokonce také v pásmu 2,4 GHz. Dle konfigurace činí přenosové rychlosti 20, 40 nebo 250 kbit/s. Na vývoji a evoluci technologie se podílí desítky firem, nejznámějšími z nich jsou Motorola, Philips, Samsung, Honeywell, ABB a například Siemens. Všechny jsou členy ZigBee aliance, jejímž cílem je inovace protokolu i pro využití v průmyslové automatizaci a Internetu věcí. Aliance také funguje jako certifikační autorita. Zařízení, které projdou certifikačním procesem, pak mohou využívat ZigBee ochranou známku. Certifikace se vztahuje pouze na aplikační vrstvy, žádný specifický hardware není vyžadován a proto existuje na trhu několik výrobců, kteří dodávají mikrokontroléry s hardwarovou podporou standardu.

Standard dělí zařízení na plně funkční (Fully Functional Device; FFD) a s omezenou funkčností (Reduced Functionality Device; RFD). Ty s plnou funkčností implementují plný protokol standardu ZigBee, zatímco zařízení s omezenou funkčností pouze části protokolu nezbytné k zajištění komunikace. Důvodem je šetření programové paměti na jednočipových mikrokontrolérech. Oba dva typy implementují na fyzické vrstvě šifrování AES s délkou klíče 128 bitů. Zařízení mohou využít v rámci jedné sítě 64-bitovou nebo zkrácenou 16-bitovou adresu. Poté je počet zařízení v jedné síti omezen na 65 535. Každá síť je dále identifikována svým PAN ID, které pak umožní existenci více logicky oddělených ZigBee sítí, i když se mohou signálem vzájemně překrývat. Co se týče síťové topologie, je základní hvězdicová s centrálním prvkem (koordinátorem), který přijímá a orchestruje komunikaci mezi všemi ostatními prvky. Druhá topologie je pak stromová, kdy s koordinátorem komunikují pouze plně funkční zařízení jako podružní koordinátoři. Ti plní funkci routerů. S těmi pak komunikují přímo libovolná koncová koncová zařízení. Pokud se mezi koncovými prvky vytvoří přímé spojení, vzniká třetí typ sítě, mesh. Je tak možné rozšiřovat pokrytí signálem, ovšem dva prvky s omezenou funkčností mezi sebou komunikovat nemohou. Ukázka topologie je znázorněna na obrázku 3.8.



Obrázek 3.8: Ukázka síťové topologie mesh [autor]

ZigBee sítě podporují dva typy komunikace, s vysílacími rámci (beacons) a bez nich (non-beacons). Bez vysílacích rámců mají routery a koordinátor trvale zapnuté transmittery a koncové prvky mohou naprosto libovolně vysílat. Tento způsob klade minimální požadavky na strukturu přenášených dat a plánování topologie. Zároveň je však kvůli neustále zapnutým transmitterům výdrž na baterii velmi nízká a tento způsob komunikace je vhodný pro zařízení napájená ze sítě, případně pro prvotní prototypování. V režimu s vysílacími rámci má každý router uživatelsky definovaný časový interval, kdy je transmitter uspán. Po jeho probuzení je vyslán komunikační rámec (beacon) a koncové prvky, které mají data k delegovaní, na něj mohou reagovat nebo zůstat uspány a komunikaci ignorovat. Tento přístup výrazně spoří baterii, zároveň ale při příliš vysoké periodě není možné v síti provádět rychlé reakce. V prostředí automatizace pak reakční čas může být naprosto kritický. Koncové prvky nemohou mimo komunikační rámce odesílat asynchronní data.

IQRF

IQRF [157] je technologická platforma kompletně vyvinutá v České Republice firmou IQRF Tech s.r.o. Primárním cílem pro využití IQRF jsou senzorické sítě, Internet věcí a průmyslová automatizace, ale

i domácí automatizace. Technologie pracuje jako mnoho dalších v bezlicenčních pásmech 868, 916 a 433 MHz. Pro každé pásmo je nutné zakoupit přímo uzpůsobený transceiver, nejsou mezi pásmy vzájemně kompatibilní. Dosah technologie je v řádu desítek metrů se zabudovanou anténou, při návrhu vlastní směrové antény lze pak dosáhnout vzdálenosti až stovek metrů (při přímé viditelnosti). Síťová topologie je implementována jako mesh, kde musí být přítomen alespoň jeden hlavní koordinátor [100]. Následně může být až 239 nodů (koncových prvků) v jedné sítí. Libovolný nod se může stát podružným koordinátorem a řídit další síť také s 239 prvky. Pokud jsou podružní koordinátoři nakonfigurování do stejné sítě, oba v různých časových intervalech předávají datové pakety od nodů centrálnímu koordinátorovi a naopak. Díky tomu je paket doručen i když přestane existovat velké množství cest v síti. Nodů může existovat díky podsítím několik tisíc, jelikož jeden paket může mít až 240 přeskoků. Protože čas jednoho přeskoku závisí na stavu koordinátora (počet nodů zbývajících k obsluze a kvalita sítě), pro zachování přijatelné odezvy sítě[2], je doporučeno nesnažit se cílit na maximální limity technologie. Maximální velikost uživatelských dat v paketu činí až 64 bajtů a transceivery mezi sebou komunikují přenosovou rychlostí 19 200 baud/s.

Síťová a linková vrstva operačního systému není při vývoji programátorsky přístupná, výsledný firmware se sestavuje vůči hotovému binárnímu kódu. Existují dva přístupy, jak vyvíjet pro IQRF transceivery: 1) dvouvrstvý s operačním systémem, a 2) třívrstvý s DPA frameworkem. Při dvouvrstvém vývoji má programátor k dispozici API vůči linkové vrstvě a obsluhu senzorů přes různé sběrnice jako I^2C nebo SPI musí implementovat sám. Pokud je k IQRF trasmitteru ale připojen pouze senzor a není vyžadována další speciální činnost, lze využít DPA framework, kdy se přes síťový protokol nastaví požadovaná sběrnice, úvodní konfigurace senzoru a registry, ze kterých se mají vyčítat data. Tento přístup rapidně urychluje vývoj a nasazení technologie. Pro vývoj je dostupné kompletní open-source SDK.

Z-Wave

Komunikační technologie Z-Wave [158] byla navržena přímo pro účely domácí automatizace již v roce 2001 společností Zensys. V roce 2005 bylo zformováno sdružení Z-Wave Alliance, jehož členy jsou jak společnosti vyrábějící prvky pro domovní instalace (Danfos, Leviton, GE, Honeywell, atd.), tak i technologické společnosti jako Samsung, Intel nebo LG. Z-Wave pracuje v přenosovém pásmu 865 až 922 MHz v závislosti na lokalitě a může tak nastat regionální problém, kdy žádané zařízení není k dispozici pro daný trh. V současné době lze zakoupit LED svítidla, spínače, dimery, zásuvkové moduly, sondy měření spotřeby, detektory zaplavení, dveřní zámky, termostaty, alarmy, kouřové hlásiče a mnoho dalších typů zařízení. Celkem existuje více než 600 výrobců, kteří poskytují přes 2 100 různých produktů. Každý výrobce musí před uvedením výrobku na trh projít certifikačním procesem, jinak mu není umožněno výrobek prodávat jako Z-Wave produkt. Každé zařízení má přesně definovaný komunikační protokol a ovládací třídy, které musí respektovat. Díky tomu je každé zařízení plně kompatibilní s jinými a libovolnou řídící jednotkou nebo transceiverem.

První verze specifikace podporovaly přenosovou rychlost 9 600 baud/s. Ta byla v roce 2013 zrychlena až na 250 kbaud/s, tedy na obdobnou jako u technologie Bluetooth Low Energy. Zařízení využívající nový standard jsou zpětně kompatibilní s původními specifikacemi. Pro odlišení od původní specifikace nesou označí Z-Wave Plus. I přes šifrování přenosu protokolem AES není možné zaručit stoprocentní

[2]Ta se bude odlišovat v závislosti na scénáři použití. Mesh senzorů teploty a vlhkosti nevyžaduje pohotové doručení paketu ke koordinátorovi, ovšem systém osvětlení budovy musí mít odezvy co nejkratší.

bezpečnost, například proti vyčtení identifikátorů z non-volatilní paměti [15]. Komunikační vzdálenost v otevřeném prostranství činí až 100 metrů.

Původně byla celá specifikace komunikačního protokolu proprietární a plně k dispozici pouze pro členy Z-Wave Alliance. V září 2016 ovšem Sigma Designs, nynější správce hardwarové specifikace vydala značnou část pro veřejnost. To umožnilo vznik open-source knihoven a Z-Wave produkty jsou nyní podporovány v různých automatizačních softwarech, bez nutnosti platit licenční poplatky.

Co Z-Wave odlišuje od ostatních technologií je přístup k meshování. Zatímco IQRF a Bluetooth spoléhají na dynamické meshovaní, kde je cesta neustále přepočítávána, optimalizována a měněna, u Z-Wave jsou cesty vytvořeny staticky během prvotní instalace infrastruktury. Z-Wave výrobky mohou být napájeny buď ze sítě nebo i z baterií, ovšem každé síťově napájené zařízení se chová jako repeater datových zpráv. U síťových prvků se také nepředpokládá jejich přesouvání z místa na místo. Naopak u bateriových zařízení, jako jsou dálkové ovladače nebo přenosné senzory, je toto očekáváno. V momentě, kdy je domovní instalace síťových komponent finální, je na primární řídící jednotce spuštěna diagnostika sítě. Při tomto procesu jsou stanoveny statické cesty mezi repeatery k řídící jednotce. Přenosná zařízení pak detekují každý repeater tak, jakoby přímo komunikovaly s řídící jednotkou. Díky tomuto přístupu se pak během běžné komunikace nepočítají nové cesty a odezvy sítě jsou mnohem rychlejší než při dynamickém meshování. V jedné Z-Wave síti může být 127 prvků, jeden z musí být primární řídící jednotka, pak neomezeně koncových prvků a podružných kontrolérů. Jednotlivé sítě se mohou signálově překrývat, protože řídící jednotka používá unikátní ID sítě a další prvky přiřazené do této sítě komunikují pouze s tímto ID.

Shrnutí

V tabulce 3.2 jsou jednotlivé komunikační technologie zhodnoceny z hlediska využitelnosti v IoT platformách. Je zřejmé, že existují technologie vhodné pro komunikaci brány se zařízeními jak metalicky tak bezdrátově, technologie pro komunikaci IoT integračního middleware a bran či přímo se zařízeními.

3.5.3 Technologie pro komunikaci mezi IoTIM a branami

Tato skupina technologií je využitelná pro komunikaci mezi integračním middlewarem a branami, ovšem některé technologie jsou využitelné i pro vystavování aplikačního rozhraní klientským aplikacím.

TCP a UDP protokoly

Protokol IP leží na třetí (síťové) vrstvě referenčního ISO/OSI modelu a je využíván všemi dalšími protokoly, které pracují přes ethernet, WiFi nebo LTE. Nad tímto protokolem jsou ve čtvrté transportní vrstvě implementovány protokoly TCP a UDP. TCP pracuje jako spojový protokol, kdy je nejprve na serveru na specifikovaném portu otevřen socket a nastaven na naslouchání. K serveru se připojují klienti a mohou si obousměrně sdělovat zprávy, jejichž formát není definován a je aplikačně specifický. U TCP je řešeno pořadí paketů a jedná se o spolehlivou komunikaci za cenu nižší datové propustnosti. Datagramový protokol UDP nevyžaduje navazování spojení, je sestaven paket a odeslán, bez kontroly zdali byl úspěšně doručen a v jakém pořadí. Nad protokolem UDP jsou nejčastěji implementovány protokoly a technologie upřednostňující propustnost nad spolehlivostí.

Tabulka 3.2: Zhodnocení komunikačních technologií ve vztahu k IoT [autor]

Technologie	**Typ**	**Topologie**	**Využitelnost v IoT platformách**
RS-232	metalická	bod – bod	zařízení – senzor
RS-485	metalická	sběrnice	brána – zařízení
CAN Bus	metalická	sběrnice	brána – zařízení
Ethernet	metalická	hvězda	brána – zařízení IoTIM – brána
DALI	metalická	sběrnice	brána – zařízení
KNX	metalická	sběrnice hvězda	brána – zařízení
Sigfox	bezdrátová	zařízení – cloud	IoTIM – zařízení
LoRaWAN	bezdrátová	zařízení – cloud	IoTIM – zařízení
NB-IoT	bezdrátová	zařízení – cloud	IoTIM – zařízení
Bluetoot Low Energy	bezdrátová	hvězda mesh	brána – zařízení
Wi-Fi	bezdrátová	hvězda	brána – zařízení IoTIM – brána
LTE	bezdrátová	klient - internet	IoTIM – zařízení IoTIM – brána
ZigBee	bezdrátová	mesh	brána – zařízení
IQRF	bezdrátová	mesh	brána – zařízení
Z-Wave	bezdrátová	mesh	brána – zařízení

I když je možné realizovat IoTIM, bránu a zařízení s využitím pouze TCP a UDP, je vhodnější využít protokoly z vyšších vrstev ISO/OSI modelu, které řeší například formát zpráv, strukturu komunikace, znovunavazování spojení a znovudoručování zpráv.

REST

REST (Representational state transfer) [73] je způsob jakým lze vytvořit, číst, editovat nebo smazat informace na serveru s využitím HTTP volání. Rozhraní REST je využitelné pro přístup ke zdrojům serveru, kde je každý identifikován pomocí URI. Jedná se o bezstavový protokol a proto musí každý požadavek serveru obsahovat všechny nezbytné informace k jeho vykonání. Využívá HTTP metodu *GET* pro získání seznamu či informace, *PUT* pro výměnu, upravení (či vytvoření pokud záznam neexistuje), *POST* pro vytvoření a *DELETE* pro odstranění záznamu.

MQTT

MQTT (dříve: Message Queuing Telemetry Transport, dnes MQ Telemetry Transport) je protokol pro předávání dat mezi dvěma a více klienty [47]. Klienti nekomunikují přímo mezi sebou, ale využívají prostředníka, tzv. brokera. Klient se připojí k brokeru a řekne mu, jaká témata zpráv (tzv. topics) jej zajímají. Témata jsou hierarchická a využívají lomítka pro zanoření. Například:

Výpis 3.1: Struktura MQTT témat

```
1 budova-a/mistnost-a4/teplota
2 budova-a/mistnost-a4/vlhkost
```

```
3 budova-j/mistnost-343/teplota
4 budova-j/mistnost-343/vlhkost
```

Na těchto tématech pak jeden či více klientů publikují zprávy. Pokud nějaký klient zprávu publikuje, klient, který se o dané téma zajímá je jeho odběratel (anglicky subscriber), broker se postará o doručení zprávy. Myšlenka protokolu MQTT tedy je, že klient nemusí řešit ostatní klienty a pouze publikuje zprávy na tématech, případně odebírá pouze ta témata, jež jej zajímají. Klient může činnosti kombinovat, tedy publikovat i přijímat zároveň.

MQTT vznikl jako protokol s ohledem na Internet věcí a je implementován tak, aby jej mohly využívat i mikrokontroléry s omezenými prostředky. Přenos probíhá pomocí TCP s možností SSL šifrování. Každý klient si může zvolit QoS level pro dané téma. V prvopočátku bylo MQTT spravováno společností IBM, nyní jde o Eclipse Foundation a celý protokol je kompletně otevřený.

V případě tohoto řešení jsou pak jednotlivými klienty IoTIM a aplikace bran. Brány publikují témata, která jsou odebírána IoTIM a naopak. Aktivně vyvíjených implementací MQTT brokeru je v současné době několik a liší se zejména v použitém implementačním jazyce, schopností škálování a clusteringu a možnostmi monitoringu a auditingu. Díky standardizaci je možné vybrat si libovolný MQTT broker a mít jistotu kompatibility.

CoAP

Constrained Application Protocol (CoAP) [159] je specializovaný protokol popsaný standardem RFC 7252, vytvořený primárně pro zařízení s omezenými prostředky. Nejčastěji se využívá nad UDP protokolem, ovšem lze nalézt i implementace pro 6LoWPAN sítě. Každá CoAP zpráva musí mít přítomnou verzi protokolu, typ zprávy (požadavek s nutností potvrzení, požadavek bez nutnosti potvrzení, acknowledge zpráva a resetovací zpráva). Dále pak token, identifikátor zprávy a vlastní volitelná data. Architekturou se jedná o protokol typu request/reply.

3.5.4 Standardy pro přenos informací

Výše popsané protokoly jsou agnostické vůči datům, která jsou odesílána a přijímána. Nyní budou popsány vybrané technologie, které se soustředí naopak na strukturu zpráv a ignorují komunikační technologie.

JSON

Jedním z nejvyužívanějších textových formátu je JSON [160] (JavaScript Object Notation), původně vyvinutý pro programovací jazyk JavaScript. Je zcela nezávislý na platformě a existují knihovny pro serializaci a deserializaci pro v podstatě jakýkoliv aktivně používaný programovací jazyk. Vstupem jsou datové datové struktury (číslo, textový řetězec, boolean), které se mohou dále vnořovat a vytvářet tak pole nebo objekty. Každá proměnná, pole nebo objekt musí být identifikovatelná textovým klíčem. Komplexnost hierarchie není teoreticky omezena, ovšem lze narazit na praktické limity při využívání komplexních a rozsáhlých struktur.

XML

XML (eXtensible Markup Language) [161] patří mezi obecné značkovací jazyky, a opět jej podporuje široká škála programovacích jazyků. Umožňuje rychlé vytváření vlastních značek pro realizaci odlišných

případů užití. XML dokument je vždy textový s kódováním Unicode. Každý XML dokument musí mít právě jeden kořenový dokument. Jednoduchý XML dokument může mít následující podobu:

Výpis 3.2: XML dokument

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Uzivatelsky komentar -->
<zarizeni identifikator="42" lokace="Obyvaci pokoj">
  <titulek>Senzor vlhkosti</titulek>
  <data>
    <jednotka>%RH</jednotka>
    <hodnota>43</hodnota>
  </data>
</zarizeni>
```

FlatBuffers a ProtocolBuffers

Největší výhoda formátů, jako je JSON nebo XML, spočívá v lidské čitelnosti, což minimálně v prvopočátku vývoje velmi usnadňuje ladění. Stačí si zprávu vypsat do vývojářské konzole nebo uložit do souboru a analyzovat některým z desítek dostupných nástrojů. Také existují knihovny pro parsování a skládání takových zpráv. V jazycích vyšší úrovně lze využít i knihovny, které objekty mapují přímo na zprávy. Jejich největší nevýhoda spočívá ve velikosti přenášených dat, protože se jedná o čistý text.

FlatBuffers jsou open-source knihovna vyvíjená společností Google [162]. Oproti textovým formátům se jedná o formát binární, který není uživatelsky čitelný. Oproti JSON nemá tak širokou podporu programovacích jazyků, ovšem v posledním roce se seznam rozšířil na C++, C, Go, Java, C#, JavaScript, PHP, Python, Rust a Lua. Výhodou je několikanásobně nižší čas na sestavení zprávy oproti JSON formátu [163]. Prvním krokem je definice schématu využívajícím vlastní IDL (Interactive Data Language) syntaxi. K převedení schématu ze souboru se využívá program `flatc`, neboli FlatBuffer compiler, kterým se vytvoří soubor pro daný programovací jazyk. Ten obsahuje konstruktor, gettery a settery pro vytvoření binárního FlatBufferu a jeho parsování.

Přidanou hodnotou FlatBufferů je možnost parsovat již serializovaný FlatBuffer bez nutnosti deserializace, což snižuje náročnost manipulace s přijatými daty. Vedle FlatBuffers existují i další knihovny pro binární serializaci s využitím IDL. Například samotný Google poskytuje ještě ProtocolBuffers [164], jejichž největší odlišností je nutnost data deserializovat, což vede ke zvýšení režie. Výhodou oproti FlatBuffers je pak jednotné programátorské API ve všech podporovaných jazycích[3]. Existují i další knihovny jako gRPC [165], Cap'N'Proto [166] nebo Apache Thrift [167]. Jejich náročnost či efektivnost může být v některých případech lepší než FlatBuffers, ovšem za FlatBuffers stojí technologický gigant a jsou využívány i velkými společnostmi, jako je Facebook [168].

3.5.5 Technologie pro perzistenci dat

Vyjma velmi jednoduchých systémových utilit a minimalistických nástrojů potřebují aplikace ukládat informace do perzistentního úložiště, jakým mohou být magnetické pevné disky, polovodičové SSD, Flash

[3] FlatBuffers mají pro každý podporovaný jazyk odlišné API knihovny, což zvyšuje obtížnost integrace.

paměti atd. Existuje několik přístupů, které se odlišují několika faktory jako je komplexnost, vhodnost užití, škálovatelnost nebo například cena a liší se také podporovanými operačními systémy.

Soubory

Nejednodušším přístupem je využít obyčejné soubory. Práce se soubory se liší dle vybraného programovacího jazyka a operačního systému, ale obecně lze buď se soubory pracovat na nízké úrovni jako s proudem bajtů, nebo jako s textem. Aplikace si do souboru může zapisovat libovolná data, ovšem nejčastěji se u moderních serverových nebo démonizovaných aplikací[4] využívají soubory pouze k nezbytným konfiguračním údajům jako konfigurace portů, přístupové údaje k databázovým serverům nebo úrovně informačních výpisů. Vývojáři si mohou implementovat vlastní strukturu dat, ať již binární nebo textovou, případně využít široké množství standardů, jako například výše zmíněný JSON, XML nebo FlatBuffer a desítky dalších. Pro tyto formáty jsou často dostupné knihovny s open-source licencemi poskytující aplikační rozhraní pro konkrétní programovací jazyky.

Relační databáze

Pojem relační databáze byl zaveden již v roce 1970 a jedná se o technologie založené na tabulkách (relacích), tedy dvourozměrných strukturách tvořených záhlavím a tělem [90]. Sloupci jsou atributy, každý s definovaným datovým typem a řádky jsou záznamy. Každý záznam musí mít atribut jednoznačného primárního klíče. Na relační databáze se pokládají dotazy ve standardizovaném strukturovaném dotazovacím jazyku (SQL). SQL příkazy se dělí na příkazy pro definici dat (tvorba, úprava a mazání tabulek), příkazy pro manipulaci s daty (vložení, vybrání, úprava a mazání), příkazy pro řízení transakcí, přístupová práva a příkazy specifické pro konkrétní databázové implementace. Moderní programovací jazyky využívají struktury nebo objekty bez primárních klíčů a často se tak využívají relačně mapovací frameworky, které mapují objekty na záznamy v tabulkách a opačně a provádějí CRUD operace. Relační databáze využívají buď přístup klient-server a v této kategorii patří mezi nejpopulárnější MariaDB, PostgreSQL, MS SQL a desítky dalších. Existují také souborové relační databáze, kde není přítomen server, tabulky jsou obsaženy s jediném souboru a s databází se manipuluje pomocí SQL dotazů skrze programové API. V této kategorii je velmi populární sqlite.

Relační databáze mají v oblasti Iinternetu věcí praktické využití. Databáze na IoT middlewaru může nést informace o zařízeních, branách, senzorech, uživatelích atd. Brána samotná může využívat relační databázi pro perzistenci informací o lokálně připojených zařízeních. I když je možné do relační databáze ukládat i časově řady, tedy stavy senzorů, existují novější přístupy, které jsou pro tuto povahu dat považovány za vhodnější [81].

NoSQL databáze

NoSQL databáze využívají odlišné prostředky pro datové úložiště i zpracování dat, než je relační přístup s tabulkami [119]. Důvodem může být například lepší vertikální či horizontální škálování, případně kompromitování výkonu jedné operace za ceny zvýšení výkonu jiné operace. Příkladem může být rychlejší vkládání dat za cenu pomalého třídění dat. Jednou z možností implementace je úložiště typu klíč-hodnota, kdy je k unikátnímu klíči přiřazena téměř libovolná hodnota (například databáze memcached [169]). Dále

[4] Do této kategorie spadá jak aplikace IoTIM, tak aplikace bran.

pak existují databáze dokumentové, kdy jsou do databáze ukládány dokumenty v textovém formátu JSON nebo binárním BSON a každý dokument může mít odlišnou strukturu. Zde jsou typickými představiteli databáze MongoDB [170] nebo Elasticsearch [171]. Dále se můžeme setkat s databázemi sloupcovými, kde je ke každému klíči možné uložit více hodnot, jež odpovídají příslušnému sloupci a každý klíč může mít vyplněné jiné sloupce. Typickém představitelem je Apache Hadoop [172]. Grafové databáze ukládají uzly, jejich vlastnosti a hrany mezi uzly. Velkým přínosem je řádově vyšší rychlost vyhledávání oproti relační databázi. Představitelem grafové databáze je neo4j [173]. Poslední kategorií jsou databáze multimodelové, které kombinují více výše zmíněných možností do jedné databáze, například Redis [174].

Implementace IoTIM nebo brány může namísto relační databáze využívat například dokumentovou nebo klíč-hodnota databázi pro perzistenci informací o zařízení, senzorech, klientech apod.

Databáze pro ukládání časových řad

Databáze pro ukládání časových řad (Time series databases, TSDB) jsou specifickou subkategorií NoSQL databází. Jsou optimalizovány výhradně pro měření hodnot měnících se v čase, tedy pro časové řady. Každý zdroj dat (v případě IoT domény tedy senzor) je umístěn v odděleném měření a jednotlivé vzorky jsou identifikovány časovou známkou [81]. Předností databází pro ukládání časových řad je výkon při vkládání nových měření a rychlost získání souvislého úseku časové řady. Tento typ databází naopak není optimalizován pro řazení. Časové řady lze samozřejmě ukládat i v relační databázi či odlišné kategorii NoSQL databází. Není ovšem zaručeno dosažení optimálního výkonu. Typickými představiteli databází pro ukládání časových řad je InfluxDB [175], Prometheus [176] nebo Graphite [177].

3.6 Možné klasifikace IoT integrační vrstvy

Dle [95, 101] lze architektury IoTIM klasifikovat do následujících kategorií dle jejich interního návrhu a implementace:

Založené na událostech
Všechny komponenty mezi sebou interagují prostřednictvím událostí. Každá událost má svůj definovaný typ a parametry. Události jsou generovány producenty a přijímány konzumenty. Na tento návrh lze nahlížet také jako na návrhový vzor publish/subscribe, kde entity odebírají (subscribe) konkrétní typy událostí a obdrží notifikaci při vzniku nové události.

Orientované na databáze
IoTIM spadající do této kategorie považují IoT zařízení za virtuální relační databázový systém. Klientské aplikace se pak na dotazují databáze pomocí dotazovacího jazyka. Dále mohou být poskytována rozhraní pro snadnou extrakci dat. Tento přístup může mít problémy se škálováním při zvyšujícím se počtu zařízení, jelikož je jedná o centralizovanou architekturu.

Sémantické
Sémantický middleware se soustředí na interoperabilitu různých typů zařízení, které komunikují pomocí odlišných komunikačních protokolů. Kombinuje odlišná zařízení a ontologie do unifikovaného frameworku, který je využíván pro výměnu dat mezi odlišnými zařízeními [88]. Pro jednotný sémantický formát je nutné mít N adaptérů pro komunikaci s N zařízeními, protože pro každé odlišné zařízení musí být vyvinut adaptér k mapování konkrétního komunikačního protokolu na protokol abstraktní.

U sémantického IoTIM musí být přítomna sémantická vrstva, ve které je prováděno mapování prostředků pro konkrétní softwarové služby [109]. Ty pak nekomunikují přímo mezi sebou, ale využívají vzájemně srozumitelný jazyk (založený na sémantickém webu). Tento interní návrh umožňuje integraci fyzicky odlišných zdrojů a jejich vzájemnou komunikaci, i když nevyužívají kompatibilní protokoly.

Aplikačně specifické

Tento typ IoTIM se využije pro velmi specifickou aplikační doménu a celá jeho architektura je navržena a odladěna na míru konkrétních požadavků. Klientské aplikace jsou pak těsně spjaty s tímto middlewarem a nejsou zamýšleny pro obecné použití. Příkladem může být IoTIM pro analýzu vitálních funkcí ze senzorů umístěných v nemocničních lůžkách. Takový IoTIM bude mít specifické požadavky na spolehlivost, management i zabezpečení.

3.7 Rozdíl monolitické aplikace a architektury mikroslužeb

Monolitické aplikace či monolitické služby jsou klasickým přístupem vývoje. Typicky je taková aplikace rozdělena do tří vrstev – uživatelského rozhraní (front-end), business logiky a datové vrstvy. Pro většinu monolitických aplikací platí, že jejich programový kód je napsán v jediném programovacím jazyce, je uložen v jediném repozitáři a také je spravován jako monolit. Vnitřně jsou monolitické aplikace strukturovány a modularizovány pomocí knihoven, jmenných prostorů, tříd, funkcí a programových bloků [38].

Zejména v případě webových aplikací, respektive aplikací využívající model klient-server, je reálná aplikace realizovaná komplikovaněji. Vlastní realizace front-endu je skrytá za mezivrstvou realizovanou HTTP serverem, který deleguje požadavky od klientů. Front-end, business logika a datová vrstva pak můhou běžet v aplikačním serveru či kontejneru. Aplikace může být implementována v širokém množství programovacích jazyků, využívat frameworky či sady knihoven. Aplikace většinou využívají k perzistenci dat databázi s abstrahovaným přístupem přes datovou vrstvu.

Častým problémem monolitických aplikací je jejich příliš velká granularita a tím pádem větší nároky na hardware či na cloudové prostředky. Dále může být obtížné vybalancovat požadavky na výkon procesoru, rychlost I/O operací, velikost operační paměti či kapacitu síťového připojení. To samé platí pro horizontální nebo vertikální škálování, pokud aplikace využívá jedinou databázi. Monolitické služby a aplikace se používají již desítky let a postupně byly zjištěny limity této architektury z hlediska rychlosti vývoje, kvality služby, udržovatelnosti i rychlosti. Rostoucí komplexnost aplikací vedla ke vzniku nových konceptů architektur služeb s cílem odstranit největší problémy klasických monolitických služeb, ale zároveň nepřinést problémy nové. Sada konceptů, idiomů a principů byla pojmenována jako mikroslužby.

Základní myšlenka mikroslužeb je v podstatě jednoduchá, celá aplikace se skládá z většího množství (jednotky až desítky) mikroslužeb, tedy malých aplikací, kde každá má jasně definovanou roli a běží v samostatném procesu s limitovaným počtem vláken [30]. Mikroslužby komunikují s ostatními mikroslužbami buď přímo nebo nepřímo, často s využitím nějakého standardního protokolu jako REST přes HTTP, nebo pomocí specializovaných technologií pro messaging.

Ve skutečnosti není tato myšlenka nijak nová, jelikož operační systémy vycházející z OS Unix jsou založeny na stejných principech. Unixové systémy obsahují mnoho nástrojů a každý z nich má přísně omezenou roli, kterou musí zvládat dobře. I myšlenka komunikace mikroslužeb mezi sebou není nová,

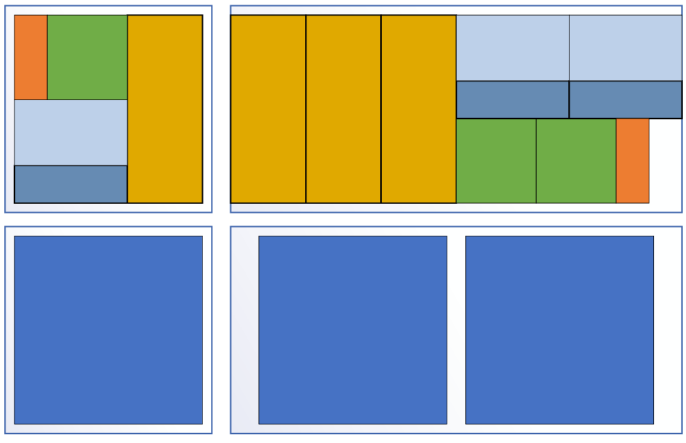
Unix jí podporuje mnoho let pomocí signálů, rour, socketů a zprávových front. Mikroslužby tento ověřený koncept rozšiřují do systémů s větším množstvím počítačů nebo kontejnerů [10].

Tabulka 3.3: Srovnání monolitické aplikace a architektury mikroslužeb [autor]

	Monolitická architektura	**Mikroslužby**
Výhody	Jednoduché debugování a testování Snadné nasazení Rychlejší vývoj Méně podpůrných	Nezávislost komponent Rychlejší nasazení Škálovatelnost Flexibilnost při výběru technologií Vysoký stupeň agility Redundance
Nevýhody	Pomalejší pochopení zdrojového kódu Provádění změn Škálovatelnost Bariéra pro nové technologie Nižší redundance	Přidaná komplexnost Distribuce systému Obtížné testování

V tabulce 3.3 jsou shrnuty základní výhody a nevýhody monolitické aplikace a architektury mikroslužeb. Největší přidanou hodnotou mikroslužeb je možnost odděleného vývoje aplikací, kdy každá obsahuje svoje interní datové úložiště a může být realizována v odlišném programovacím jazyce než ostatní mikroslužby. Nevýhodou tohoto přístupu je obtížnější vývoj, nutnost využít nástroje pro automatizované nasazení a schopnost zvolit správnou úroveň granularity vyvíjené aplikace.

Přidanou hodnotou mikroslužeb je možnost lepšího škálování prostředků oproti monolitickým aplikacím [123]. Na obrázku 3.9 je znázorněno využití prostředků monolitické aplikace (vlevo) a mikroslužeb. Ohraničení lze chápat jako plochu různých prostředků (operační paměť, výkon procesoru, atd.). Prázdná plocha je pak nevyužitý prostředek. Jak je vidět, v případě monolitické aplikace lze škálování provést spuštěním druhé instance aplikace, zatímco mikroslužby lze škálovat s větší granularitou a lépe využít prostředky.



Obrázek 3.9: Škálování monolitické aplikace a mikroslužeb [autor]

3.8 Pohled na IoT z hlediska nasazení

Proces uvedení provozu IoT systému lze rozdělit na nasazení IoTIM, kde se jedná o běžnou softwarovou aplikaci, a na nasazení bran a zařízeních samotných.

3.8.1 Nasazení IoTIM

Nezávisle na na kategorii IoTIM se vždy jedná o aplikaci. Při návrhu takové aplikace je nutné například rozhodnout, zda-li bude realizována jako monolitická aplikace, či jako shluk vzájemně komunikujících mikroslužeb. Dále se mohou odlišovat vybranými programovacími jazyky, komunikačními technologiemi, jsou-li nativní nebo zda vyžadují webový nebo aplikační server, licencí, aplikační doménou nebo plánovaným rozsahem. V případě mikroslužeb mohou být nasazeny přímo na operačním systému, v kontejnerech nebo v clusteru kontejnerů.

IoTIM middleware může být nasazen několika odlišnými způsoby, přičemž každý z nich má pro poskytovatele IoTIM výhody i nevýhody.

Vlastní dedikovaný server

V této variantě má poskytovatel IoTIM zakoupen dedikovaný serverový hardware, umístěný v ním vlastněné lokalitě s dostupnou veřejnou IP adresou. Výhodou tohoto přístupu je maximální kontrola nad hardwarem, která přináší ovšem i nevýhodu v podobě nutného dohledu a servisu, aktualizací a nutností investice do dohledového řešení. Tento přístup je vhodný zejména pro velmi malé projekty s nízkou garancí dostupnosti služeb, často zahrnovanou ve smlouvě mezi uživatelem a poskytovatelem. Tato smlouva je nazývaná jako Service-level agrement (SLA) a její součástí je často procentuální garance dostupnosti služeb v roce. SLA s garancí 99,9 % znamená maximální možný výpadek služeb po dobu osmi hodin a 45 minut za rok, 99,5 % pak maximálně jeden den a 19 hodin. Tato varianta je totiž cenově úporná pouze jedná-li se o jednotky serverů. Při vzrůstajícím počtu serverů rostou také požadavky na redundanci, síťovou infrastrukturu, záložní zdroje a například i na klimatizaci a vytápění místnosti.

Pronajímaný dedikovaný server

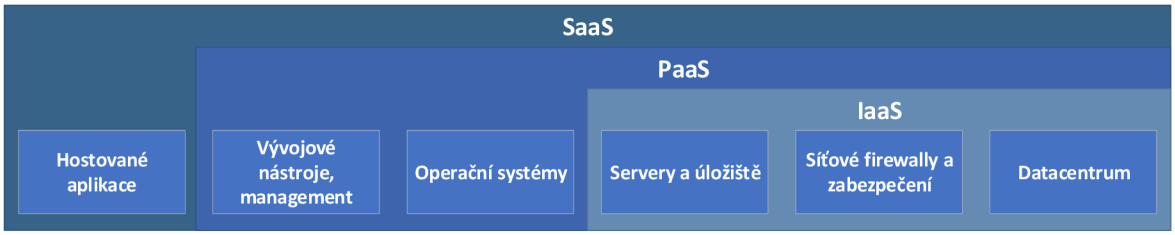
V tomto případě využívá IoTIM také celý server, ovšem hardware serveru je pronajímán za poplatky a servis je zajišťován pronajímatelem, včetně síťové infrastruktury, konektivity a zálohování. V České Republice je takovým poskytovatelem například společnost Wedos [178]. Nájemce dostává kompletní přístup k serveru v podobě vzdálené klávesnice, myši a monitoru. Tento přístup se nejčastěji používá pouze pro prvotní instalaci operačního systému, poté je využíván přístup přes SSH.

Virtuální privátní server

Virtuální privátní server (VPS) využívá principů virtualizace hardwaru. Na serveru je nainstalován hypervizor, který dle svého plánovače poskytuje prostředky virtuálním počítačům [49]. Díky tomu lze na výkonném serveru provozovat několik od sebe kompletně izolovaných virtuálních počítačů. Tyto počítače mají přidělenou část operační paměti, část diskového prostoru a definován počet procesorových jader, která mohou využívat. Jedno procesorové jádro může být sdíleno v určitém poměru mezi více virtuálními počítači.

Cloud computing

Cloud computing je model vývoje a používání softwaru. Jeho principem je poskytování výpočetního výkonu serverů uživatelům [96]. Mezi atributy cloud computingu patří škálovatelnost a elasticita, které umožňují měnit uživatelům výši dostupných prostředků operativně. Jedná se o přístup *pay as you go*, kdy uživatel platformy platí pouze za ty prostředky, které skutečně spotřebuje. Dalším atributem je pak aktuálnost, kdy poskytovatel garantuje, že podpůrný software a operační systém je vždy aktuální.



Obrázek 3.10: Typy cloud computingu (překresleno z [136])

Rozdílné distribuční modely cloud computingu jsou znázorněny na obrázku 3.10 a lze je rozdělit na tři kategorie:

- IaaS – infrastruktura jako služba (Infrastructure as a Service). Poskytovatel služeb poskytuje infrastrukturu. Může se jednat pouze o dedikovaný hardware nebo a virtualizovanou platformu s hypervizorem. Výhodou tohoto přístupu oproti vlastnímu dedikovanému serveru je, že o hardware se stará poskytovatel. Oproti pronajímanému dedikovanému serveru a VPS jsou možnosti škálování signifikantně vyšší.
- PaaS – platforma jako služba (Platform as a Service). V tomto případě poskytovatel nabízí kompletní prostředky pro podporu celého životního cyklu aplikací. Nevýhodu představuje možné proprietární uzamčení, kdy aplikace musí využívat specifické prostředky poskytovatele. Podpora nejnovějších technologií může být také omezena. PaaS konkrétního poskytovatele může například nabízet běhové prostředí pro aplikace v jazyce Java EE a relační databázi MS SQL, ale nepodporovat nativní aplikace v jazyce Go a databáze pro ukládání časových řad.
- SaaS – software jako služba (Software as a Service). Aplikace jsou poskytovány uživatelům jako služby a uživatelé si platí přístup k aplikaci, nikoliv aplikaci samotnou. Typickým představitelem takových aplikací jsou Google Workspace [179], zahrnující emailovou schránku GMail a sadu kancelářských aplikací.

Cloud computing platformy dále poskytují čtyři možné modely nasazení:

- Veřejný – model, ve kterém je služba poskytnuta široké veřejnosti a všichni klienti obdrží velmi obdobnou funkcionalitu.
- Soukromý – cloud je provozován pouze pro organizaci, buď třetí stranou nebo samotnou organizací.
- Hybridní – kombinace veřejného a soukromého modelu, kdy některé aplikace jsou poskytovány veřejnosti a další jsou pouze interní.

- Komunitní – infrastruktura je sdílena mezi několika organizacemi, které jí využívají. Organizace může spojovat například stejný obor zájmu.

Doporučení

Z předchozího popisu je zřejmé, že IoT middleware lze nasadit několika různými způsoby, z nichž každý má své výhody a zápory. Nelze ovšem zcela jasně doporučit, který přístup je ten nejlepší, jelikož to závisí na povaze projektu, počtu uživatelů a požadavcích organizace [22, 78]. U systému pro domácí automatizace bude preferována nízká latence, aby reakce na změnu senzorů nebo požadavky uživatelů proběhly co nejrychleji, tedy lokálně umístěný server bude dobrou volbou. U projektu chytrého města s aplikací pro občany a dohledovým systémem pro autority bude vhodný hybridní PaaS cloud. Systém monitorující stav výrobních strojů firmy s více pobočkami bude nejspíše postaven na privátním IaaS cloudu. Dalším faktorem je samotná architektura IoTIM. Pokud je realizována jako monolitická aplikace, tedy jeden proces v operačním systému, jsou možnosti nasazení v cloud platformách omezené.

3.8.2 Nasazení bran a zařízení

Poznatky z této podkapitoly vycházejí z publikace [99], kde autoři představují komplexní architekturu chytrého města, a dále z publikací shrnujících poznatky z provozu v reálném prostředí [110, 64].

Komerčně dostupné brány jsou často dodávány s předinstalovaným operačním systémem i softwarem brány a je pouze nutné provést nastavení připojení k IoTIM a následně již začleňovat nová zařízení. V případě open-source projektů využívajících běžně dostupné embedded počítače jako Raspberry Pi je často k dispozici instalační balíček, který je nutné v operačním systému nainstalovat, případně celý obraz operačního systému.

Častým problémem výzkumných projektů v IoT je testování systému s branami a zařízeními pouze v ideálních laboratorních podmínkách [4], případně kompletně virtuálně [124]. V laboratorní místnosti může být komunikace mezi branou a zařízeními bezproblémová díky absenci překážek v prostředí a externích rušivých vlivů.

V případě brány poskytující bezdrátovou technologii pro připojení zařízení je nutné zvážit při nasazení její umístění. Pokud je brána umístěna daleko od zařízení, nemusí být přenos dostatečně spolehlivý a může docházet k opakování přenosu paketů a tedy i snížení maximální propustnosti sítě. Tento problém je částečně eliminován u technologií podporující topologii mesh. Brány provádějí překlad mezi IP technologiemi (Ethernet, WiFi nebo LTE), které jsou energeticky náročné, a proto vyžadují trvalé napájení z elektrické sítě. Umístění brány musí reflektovat i tento požadavek.

Praktickým problémem při nasazení zařízení jsou různé požadavky a možnosti napájení. Na první pohled existují pouze dvě kategorie zařízení, a to ty trvale napájené z elektrické sítě a ty bateriové. Ovšem při plánování nasazení systému chytrého města bylo zjištěno, že existuje pět odlišných kategorií napájení zařízení:

- Zařízení instalovaná na veřejných budovách, tedy bez jakéhokoliv omezení napájení (provoz 24/7).
- Zařízení, která jsou instalována do veřejné dopravy, jako autobusy či služby taxi, kontinuálně napájená z baterie automobilu. Tato zařízení mohou pracovat pouze když je vozidlo nastartováno.

- Zařízení připojená k veřejnému osvětlení, které je napájeno pouze v noci. Jedná se hlavně o fasády budov a pouliční lampy. Takové zařízení musí obsahovat baterii k provozu přes den a v noci je napájeno z přívodu a dobíjí baterii.
- Zařízení bez dostupné sítě, které musí vystačit pouze s baterií. Jde například o parkovací senzory, které jsou pod vrstvou asfaltu vozovky.
- Pasivně napájené externími zdroji. Nejčastěji se jedná o různé QR kódy a NFC či RFID tagy.

Zařízení se také dají klasifikovat na dvě kategorie dle opravitelnosti. Jde o zařízení přístupná, s otevíratelným pouzdrem, se kterými lze manipulovat a vyměnit vadnou komponentu na místě provozu. Dále pak jde o zařízení, které se na místě musí celá vyměnit, rekonfigurovat nastavení brány a IoTIM a vadné zařízení opravit v servisním místě. Některá zařízení, například průjezdové či parkovací senzory, jsou zapuštěna ve vozovce a jejich výměna je nákladná.

Dle [110] je v IoT důležité neustále monitorovat stav zařízení i kvalitu přenosu. Nejsou-li zařízení dosažitelná nebo jsou-li jejich měření špatné kvality, není možné na základě takových dat provádět dobrá rozhodnutí. Autoři dále zmiňují několik praktických poznatků při nasazení bran a zařízení v městské infrastruktuře:

- Instalační krabice s garantovaných krytím IP67 i přes toto krytí nezabránily vniknutí dešťové vody přes závity šroubů. Krabice doporučují instalovat dnem vzhůru.
- Přes použití komerčně dostupných modulů i hotových zařízení bylo od technických správců města vyžadována instalace jističe a proudové ochrany. Proto bylo nutné většinu zařízení napájených z elektrické sítě přepracovat.
- Zařízení je nutné umisťovat mimo dosah lidí z důvodu vandalismu.
- Některé komerční budovy obsahují z důvodů bezpečnostních politik bezdrátové rušičky a zařízení nebylo možné v jejich okolí provozovat.
- Zařízení s kombinovaným provozem na baterii a z elektrické sítě nesmí překročit určitý limit vybití baterie, jelikož je pak nelze dobít a musí dojít k servisnímu zásahu.

3.9 Ekonomický pohled na IoT

Je odhadováno, že v roce 2020 je k internetu připojeno 50 miliard zařízení [26]. Mezi ně patří jak běžná spotřební zařízení jako osobní počítače, notebooky, tablety a mobilní telefony, tak i IoT zařízení s připojenými senzory a aktuátory. Z ekonomického hlediska je tedy IoT atraktivním odvětvím s potenciálním velkého růstu a přidané hodnoty. Článek [37] klasifikuje různé IoT aplikace do sedmi různých kategorií podle toho, jakou přinášejí podnikům a uživatelům přidanou hodnotu. Čtyři kategorie využívají komunikace zařízení-zařízení (Machine-to-machine – M2M) a zbylé tři pak interakci uživatelů. Publikace [84] pak kategorie doplňuje o potencionální rizika.

Jednoduché manuální spuštění přiblížením

První kategorie je velmi základní a je součástí mnoha aplikací, například samoobslužného odbavení v obchodě, inventur, kontroly přístupu do budov a fitness zařízení, platebních bran nebo čipování zvířat. Obchodní hodnotou je schopnost sdělit unikátní identifikátor tím, že jsou ručně a vědomě přesunuty do prostoru čtecího senzoru nebo antény. Jakmile je zařízení dostatečně blízko k aktivnímu místu, dojde k automatickému spuštění transakce, například platby, kontroly platnosti nebo aktualizováním záznamu v databázi. Společnosti zahrnují tuto kategorii do svých aplikací, neboť usnadňují práci jejím zaměstnancům (přiložení přístupové karty je pohodlnější než zadávání kódu na klávesnici), umožňuje samoobslužnost zákazníka a v důsledku toho jsou sníženy náklady na pracovní síly. Spotřebitelům tato kategorie pomáhá šetřit čas, zvyšovat nezávislost a pohodlí. Potenciálním rizikem této kategorie je možnost odchytávání identifikátorů a jejich zneužití.

Automatické spuštění přiblížením

Tato kategorie rozšiřuje předchozí o důležitou vlastnost – transakce je spuštěna automaticky když vzdálenost dvou zařízení klesne pod prahovou hodnotu, například když spotřebitel odejde z obchodu s nákupem a zapomene jej zaplatit. Mnoho společností využívá tuto kategorii od správy aktiv po správu zásob. Kdykoliv je zařízení umístěné například na kamionu, vysokozdvižném vozíku, paletě nebo kontejneru vzdáleno od aktivního čtecího senzoru, je spuštěna transakce jako aktualizace účetního záznamu nebo notifikace k naskladnění či spuštění poplachu. Tato kategorie tedy slouží k vytváření nových či zlepšování stávajících obchodních procesů, kdy její implementace vede ke zvýšení rychlosti, přesnosti a pohodlí, které umožňuje společnostem snížit náklady na pracovní sílu, náklady na selhání procesu a náklady na podvody. Mimo toho přináší neustále nová data, která lze využít ke zlepšování procesů v průběhu času.

Spotřebitelé mají z této kategorie užitek například v podobě automatického odemykání dveří automobilu, pokud se k němu přibližuje nositel klíče od vozidla nebo osoba s autorizovanou mobilní aplikací. Firmy dále využívají tuto kategorii v oblasti augmentované reality pro propojení digitálního a fyzické světa, kdy je při přiblížení k danému objektu tento objekt zobrazen například přes brýle [9] nebo mobilní telefon montážní plán a informace nutné k dokončení pracovního úkonu. Mezi rizika této kategorie patří snižování paměťových a vyhodnocovacích schopností lidí, odstranění edukativního a kreativního přístupu „pokus-omyl“, a směřování k neo-Taylorismu [84].

Automatické spuštění senzory

První dvě kategorie přinášejí užitek ručním a automatickým snímáním a sdělováním unikátního identifikátoru. Tato třetí kategorie rozšiřuje schopnosti chytrých zařízení o poskytování údajů ze senzorů, jako je například teplota, zrychlení, jas, vlhkost, orientace či vibrace. Díky tomu mohou zařízení neustále snímat svůj stav a stav prostředí a iniciovat akce na základě předem naprogramovaných statických či dynamických pravidel. Příkladem je inteligentní systém zalévání, který reguluje množství přivedené vody na základě teploty a vlhkosti půdy. Pravidla umožňují místní a rychlé rozhodování a zvyšují tím kvalitu procesů, což má za následek efektivnější výstupy či přístupy. V případě inteligentního zalévání to znamená ekologičtější používání vody a také pravděpodobně kvalitnější plodiny. Oblasti použití jsou rozmanité, od monitorování stavu v celém dodavatelském řetězci po inteligentní detektory kouře, od

monitorování mostních konstrukcí po systémy včasného varování před živelnými katastrofami. Tato kategorie představuje možnosti, které se otevírají díky schopnostem měřit přesně aspekty reálného světa.

Automatická bezpečnost produktu

Do této kategorie patří aplikace jako je kontrola původu (proof-of-origin), ochrana proti padělání (anti-counterfeiting), rodokmen produktu a kontrola přístupu zabezpečení související s produktem. Produkt (věc), která má být zajištěna, nese na sobě zařízení s mikrokontrolérem vybaveným bezpečnostní technologií jako je šifrování. Uživatel nebo prostor samotný může zkontrolovat platnost provedením implementované metody, například dotazem a odpovědí. Takové metody jsou již zavedené a využívané v některých odvětvích. Například jde o interakce platební karty a bankomatu nebo klíče a zámků automobilu. Vyžadují však drahé a obvykle energeticky náročné výpočetní prostředky. Kromě toho často vyžadují složité zacházení s digitálními klíči. Proto je tato metoda omezena na aplikace, kde jsou v sázce vysoké částky a rizika.

U levných, masově vyráběných produktů přichází s IoT nová metoda, kdy zařízení mohou poskytnout určitou úroveň zabezpečení na základě souhry samotného zařízení a jejího digitálního protějšku. Každé zařízení má na IoTIM historii záznamů, která se aktualizuje v momentě provedené akce s produktem. Taková historie pak připomíná životopis či rodokmen a na jejím základě lze odvodit, zda se jedná o autentický produkt.

Zatímco první metoda využívá nákladné bezpečnostní funkce zabudované do hardwaru samotného produktu, druhá metoda přistupuje k problému využitím připojení k internetu a neustálého shromažďování dat o produktu. Na jejich základě pak vyhodnocuje pravost produktu. Nevýhodami integrace těchto metod může být všudepřítomná kontrola a přesun kontroly pravosti z člověka na stroje.

Zjednodušená přímá zpětná vazba uživatelům

Koncová IoT zařízení jsou obvykle velmi malá a je častým cílem aby nebyla okem viditelná. I tak často obsahují mechanismy poskytující zpětnou vazbu uživatelům, kteří s nimi interagují v daném okamžiku. Zpětnou vazbu poskytují například k ujištění zaměstnance o úspěšném aktivovaní manuální nebo automatické spouště. Může se jednat o zvukový signál, jako je pípnutí, nebo o vizuální signál, jako je blikající LED. U zábavních produktů se může dále jednat o haptickou odezvu v podobě vibrujícího herního ovladače. V aplikacích, které se týkají zboží podléhajícímu rychlé zkáze, může jednoduchá automatická spoušť senzoru ukázat výsledek na malém displeji, které spotřebiteli řeknou, zda produkt stále stojí svou původní cenou. Pokročilé klíče od auta dokáží odhadnou směr, kde se auto nachází a indikovat jej. Ve výrobních továrnách, například ve výrobních zařízeních společností vyrábějící mikrokontroléry, mají inteligentní nástroje úsporné displeje, které operátorovi sdělují mimo jiné užitečné informace o dalším cíli, obráběcím stroji nebo poličce. Tato funkce v kombinaci s identifikací, lokalizací a připojením k systému řízení výroby umožňuje novou úroveň téměř bezchybné výroby čipů, která je také flexibilní a nákladově efektivní.

Rozsáhlá zpětná vazba uživatelům

Tato kategorie rozšiřuje výstupy jednoduché zpětné vazby pro zkvalitnění služeb. Typická IoT zařízení jsou výkonově omezená a v tomto případě jsou často zastoupena chytrými telefony samotných uživatelů a

dedikovanými aplikacemi nebo webovými stránkami. Serverové části aplikací jsou pak propojeny s IoTIM. Konkrétní aplikace, které mohou využívat tuto kategorii, jsou rozmanité. Mezi takové aplikace patří například srovnávání cen a hodnocení produktů, politické nákupní poradenství uvádějící, v jaké zemi a jakou společností byl produkt vyroben, či varování před zdravotními dopady produktu. Dále se může jednat o různé chytré spotřebiče jako jsou kávovary, fitness náramky nebo robotické vysavače, které přes mobilní aplikace poskytují pokročilé informace a možnosti nastavení. Podniky těží z tohoto hodnotového faktoru vznikem nových kanálů pro udržování kontaktu se spotřebiteli, nabízením nových služeb a získáváním pozornosti spotřebitelů. Spotřebitelé těží ze skutečnosti, že mají přístup k personalizovaným službám velmi jednoduchým a rychlým způsobem přímo na místě a v požadovaném čase.

Zpětná vazba měnící mysl

Všechny ostatní kategorie využívaly technické přínosy Internetu Věcí, ovšem tato je nevyužívá. Vychází z obavy, že propojení reálného a virtuálního světa může vést k nové úrovni manipulace s lidmi. Lidé existují v reálném světě, ovšem nespornou část volného času tráví sledováním televize, hraním her či sociálními sítěmi. I přesto autor publikace [37] tvrdí, že reálný svět přináší stále největší radost ze života a autor práce se s tímto názorem ztotožňuje. Fyzické zážitky jako je např. dotýkání nebo pobyt v bezútěšných budovách vyvolávají emocionální reakce. Nyní je část této síly přístupná v IoT aplikacích a snad bude využívána výhradně k dobrým záměrům. Mezi ukázky takové aplikace patří měření spotřeby elektrické energie a vody, které je napojeno na aplikaci srovnávající spotřebu s vrstevníky s cílem finanční úspory a upokojení ekologického svědomí. Zde se jedná o pozitivní motivaci, ovšem stejná data je možné využít k penalizaci těch neúsporných. Společnosti z odvětví veřejných služeb a pojišťovnictví mohou tuto kategorii použít k navrhování nových produktů a služeb, které sladí jejich obchodní cíle se spotřebiteli, kteří chtějí zlepšit svůj život a jednat odpovědněji. Příkladem může být vybavení vozidla senzorickým systémem, který bude pracovat podobně jako letový záznamník a v případě nehody jej pojišťovna může využít k rekonstrukci přesného průběhu události. Pojišťovny tak získají zákazníky s averzí k riziku a ti obvykle vytvářejí nadprůměrné marže a řidiči by více dodržovali předpisovou jízdu. Všechny tyto aplikace využívají efektů studovaných v oboru behaviorální ekonomie, která předpokládá, že lidé jednají různými způsoby, ale racionálně [11].

Shrnutí

Zmíněné kategorie jsou v tabulce 3.4 shrnuty z hlediska jejich principů, přidaných hodnot pro společnost a spotřebitele spolu s příklady využití. Tabulka si vzhledem k rozsahu IoT neklade za cíl být kompletní.

Kategorie	Princip	Obchodní hodnota	Hodnota pro spotřebitele	Příklad využití
Jednoduché manuální spuštění přiblížením	Tagy zjednodušují spuštění transakce, čímž zvyšují rychlost transakce, přesnost a pohodlí	Zvýšená spokojenost s prací, umožnění samo obsloužení zákazníka, snížení mzdových nákladů, zvýšení přesnosti dat	Zlepšení samo obsluhy, rychlosti a pohodlí	Samoobsluha v knihovnách, řízení přístupu do budov, značení zvířat
Automatické spuštění přiblížením	Autonomně komunikující zařízení sami spustí transakci při vstupu do cílového prostoru, což vede ke zvýšení rychlosti, přesnosti a pohodlí	Snížení nákladů souvisejících s podvrhy, selháním procesů a ceny práce; Nová vysoce granulární data pro zlepšení procesů	Zvýšení komfortu	Sledování pozice nákladu, prevence krádeže, auto klíče
Automatické spuštění senzory	Inteligentní zařízení monitorují své místní okolí, aplikují data ze senzorů na pravidla procesů a v případě potřeby provádí automatické akce; Umožňují akce založené na lokálních událostech; zvyšují kvalitu procesů	Individuální a rychlé řízení procesu zvyšuje efektivitu; Dodatečná úroveň granularity dat dále zvyšuje procesy	Posun v kvalitě produktů a služeb	Zemědělská produkce, požární bezpečnost, monitoring zboží podléhající rychlé zkáze, elektroměry
Automatická bezpečnost produktu	Integrované šifrování a propojení mezi fyzickými věcmi a jejich digitálními reprezentacemi umožňuje novou úroveň zabezpečení věcí	Snížení nákladů na selhání procesu v důsledku podvodu a snížení nákladů na zabezpečení procesů; Zvýšení zákaznické důvěry	Nové služby založené na důvěře	Řízení přístupu, dokazování originality, prevence padělků
Zjednodušená přímá zpětná vazba uživatelům	Chytré věci poskytují uživatelům přímou zpětnou vazbu ke zvýšení důvěry a ovládání místních procesů	Procesy se stávají přesnější, flexibilnější a rychlejší	Zvýšení pohodlí a zábavnosti	Rychle se kazící zboží, které informuje o svém stavu kvality, digitálně vylepšené hry, klíče od auta ukazující směr
Rozsáhlá zpětná vazba uživatelům	Hmatatelné objekty slouží jako propojení se širokou škálou služeb relevantních pro uživatele a objekty; Uživatel služby ovládá například mobilním telefonem	Získaní kontaktů na nové zákazníky, nové možnosti propagace, dodatečné výnosy ze služeb	Zvýšení komfortu díky přesně individualizovaným informacím	Detailní produktové informace, srovnávání cen, hodnocení produktů, městské a muzejní průvodci, zaznamenávání oprav
Zpětná vazba měnící mysl	Technologie zaměřené na ovlivnění chování uživatelů	Umožňuje nové emotivní funkce produktu a nové služby; umožňuje aktivní výběr atraktivních zákaznických segmentů; pomáhá sladit obchodní cíle se ekologickými cíli	Pomáhá zlepšit kvalitu života a různými způsoby pobízí k zodpovědnému chování	Zlepšování zdraví např. pomocí chytrých zubních kartáčků, úspora el. energie nebo vody

Tabulka 3.4: Ekonomický pohled na oblast Internet věcí (převzato z [37])

3.10 Výzvy a řešené problémy

V této podkapitole budou popsány vybrané problémy, kterými se v současné době výzkumníci zabývají. Jmenovitě se jedna o synchronizaci systémového času bran s IoTIM, nedostatečnými schopnostmi současných implementací bran, zpracováním dat v reálném čase, edge computingem, virtuálními zařízeními a standardizací.

3.10.1 Problematika synchronizace času

Desynchronizace času může vzniknout několika způsoby. Prvním je chybná konfigurace časového serveru, nebo využití odlišných synchronizačních technologií [24] (NTP, Time-Sync, Chrony, AboutTime atd.). Druhým je blokování síťových portů, což je často využíváno ve firemních sítích pro zvýšení bezpečnosti. V tomto případě firmy často poskytují své vlastní servery pro synchronizaci času, ovšem nemusí být kompatibilní se zvolenou platformou [115].

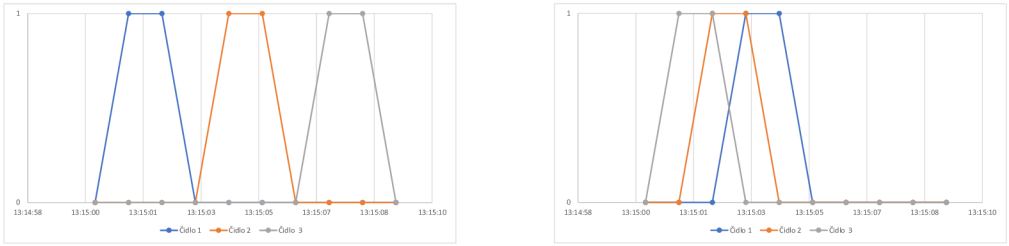
Problém blokovaných serverů pro časovou synchronizaci se prohlubuje u bran, které často využívají jednodeskové minipočítače architektury ARM, jako je například velmi populární Raspberry Pi. Ty na rozdíl od stolních počítačů, notebooků a serverů neobsahují na základní desce baterii. Tato baterie dříve primárně zálohovala nastavení BIOS, nyní slouží k napájení zabudovaného krystalu na základní desce. Pokud je zařízení vypnuto, nedojde ke ztrátě data a času. Bez baterie pak dochází na bráně ke dvou možným stavům. Pokud je systém vypnut bezpečně, je těsně před vypnutím uložen datum a čas na pevné úložiště a poté obnoven. Systém je pak bez synchronizace času oproti skutečnému času pozadu o dobu, která je rovna času vypnutí systému. Když je systém ovšem vypnut náhle, například při výpadku elektrické energie, může nastat situace, kdy se brána nastaví na výchozí čas. Ten může být zvolen výrobcem, nebo také může dojít ke kompletnímu vynulování časové známky a nastavení data na 1. 1. 1970, což je počátek epochy časové známky formátu UNIX.

Nové stavy senzorů jsou pak odesílány na IoTIM se špatnou časovou známkou. Konkrétní implementace IoTIM nemusí brát čas odeslaný branou v potaz a datům nastavovat čas v momentě přijetí. Taková implementace ovšem znemožňuje přesné měření a analýzu senzorických dat, jelikož mezi skutečným časem vyčtení a časem přijetí může vzniknout značná latence v řádu až jednotek sekund [117]. Dalším problémem špatně nastaveného času brány může být i kompletní znemožnění komunikace s IoTIM, pokud je spojení šifrované a spoléhá na certifikáty. Se špatným časem nemusí být certifikát již nebo ještě platný a nedojde ke spojení [108].

Články [71, 72] popisují nově vyvinutý protokol Secure Time Synchronization Protokol, jehož cílem je zpřesnit synchronizaci času IoTIM, bran a IP zařízení a také detekovat falešné časové známky. Detekováním podvrhů časových známek a implementací přesné časové synchronizace se také zabývá článek [92]. Synchronizaci času lze řešit pomocí filtrů a docílit synchronizaci s přesností na jednotky mikrosekund [59]. Problematika synchronizace času je také řešena mezi mobilními zařízeními [32], senzorickými sítěmi [116] i mezi servery [104]. Obecnými radami využitelnými při návrhu IoT řešení je dodržovat stejnou časovou zónu na IoTIM branách a zařízeních, i když se nacházejí v jiných zónách. Dále se uvádí ignorace letního a zimního času. Místo toho je vhodnější ukládat metadata o geolokaci a přepočítávat na místní čas až v klientských aplikacích.

Příklad dopadu špatné synchronizace času mezi branami je znázorněn na obrázku 3.11. Mějme tři brány a ke každé nechť je připojeno binární čidlo. Na obrázku a) jsou brány časově synchronizovány a na

čidlech postupně v čase nastane změna stavu z logické 0 do 1 a zpět. Když jsou naměřené stavy zobrazeny jako časové řady, je zřejmé, že jev nastal nejprve na první čidle, poté na druhém a nakonec na třetím. Je-li ovšem čas bran desynchronizován, kdy je první brána zpožděna o 2 sekundy oproti skutečnému času, druhá předbíhá o 2 sekundy a třetí o 6 sekund, dojde k mylné interpretaci jevů. Nejenom že zdánlivě nastávají v opačném pořadí, není zde zachycena ani prodleva mezi nimi.



a) b)

Obrázek 3.11: Rozdíl v interpretaci dat dle času [autor]

3.10.2 Rozšíření schopností bran

Brána je definována jako prostředník mezi zařízeními nedisponujícími IP protokolem (Bluetooth Low Energy, SigFox, Z-Wave, RS-485 atd.) a IoTIM [85]. Ovšem články [128, 57, 80] ukazují, že lze schopnosti brány rozšířit o určitý stupeň autonomie, jako je například autokonfigurace nových zařízení a jejich automatická detekce [58] či off-line schopnosti brány.

Aby bylo možné bránu rozšířit o off-line funkcionality, musí obsahovat prostředky pro detekci síťového spojení a určitým způsobem detekovat přítomnost serveru [98]. Detekce pouze síťového spojení není dostatečná, zejména při použití publish-subscribe protokolů jako MQTT, kde brána může úspěšně publikovat zprávy, ovšem bez existujícího příjemce. Článek [42] představuje hardware pro multiprokolovou bránu, kde autoři jako budoucí možnosti rozšíření zmiňují možnost implementovat ovládácí pravidla v off-line stavu bez spojení s IoTIM. V článku [67] autoři prezentují softwarovou architekturu brány, která při výpadku spojení s IoTIM provádí autonomní vyčítání zařízení se senzory, aplikuje pravidla a na základě jejich výsledku nastavuje nové stavy zařízení s aktory. Ovšem využití RESP API představuje v reálném nasazení problém, kdy brána musí mít k dispozici v lokální síti predikovatelnou IP adresu a IoTIM musí periodicky volat REST API, aby brána věděla o přítomnosti serveru. Pokud by IoTIM byl nasazen nikoliv lokálně, ale v cloudu, musí být IP adresa veřejná, případně je nutno využít službu třetí strany. Jako vhodnější se jeví využití alternativních komunikačních technologií a obrácení logiky detekce off-line stavu, kdy brána sama odesílá zprávy IoTIM s cílem zjišťovat její prezenci a připravenost přijímat data senzorů.

Samotné vyhodnocování pravidel je vhodné také doplnit o implementaci logování událostí. Pokud brána ztratí spojení s IoTIM a provádí samostatně pravidla, může nastat situace, kdy se po znovunavázání spojení nachází aktory v odlišném stavu, než je zaznamenáno v databázi IoTIM. Pokud IoTIM archivuje historická data a poskytuje rozhraní pro jejich získání, nebudou data v intervalu od počátku do konce výpadku přítomna. Implementováním logování událostí na perzistentní úložiště brány lze ukládat všechny nové stavy senzorů, výsledky pravidel, chybové stavy, informační upozornění a další. Tato data mohou být pak přesunuta a vložena do databáze IoTIM při znovunavázání spojení [74]. V článku [120] je tato funkcionalita využita pro návrh IoT řešení specifického pro zemědělství.

3.10.3 Real-time data pro klientské aplikace

Real-time komunikace znamená, že IoTIM obsahuje takové komunikační technologie, které umožňují klientským aplikacím obdržet nový stav senzoru co nejdříve od jeho přijetí na IoTIM. Druhou podmínkou je asynchronní způsob komunikace, kdy se klientská aplikace nemusí periodicky dotazovat na nová data. Mezi technologie využitelné pro tento způsob komunikace patří například WebSocket, MQTT nebo CoAP. Implementace real-time přístupu umožňuje vznik klientských aplikací, které dokáží mnohem rychleji reagovat na změnu stavu senzorů. Článek [69] ukazuje, že asynchronní doručování zpráv z IoTIM je několikanásobně úspornější než periodické dotazování skrze RESP API. Provedenou změnou může být například nastavení aktoru v nižších vrstvách IoT řešení nebo interakce s jinou službou.

Dle článků [130, 122, 1] lze výzkumné problémy z oblasti tvorby IoT systémů poskytujících data v reálném čase rozdělit do šesti odlišných kategorií: na tvorbu dat, síťování, zpracování a analýzu, správu a prezentaci dat, jejich pochopení a bezpečnost a soukromí. Cílem výzkumu v oblasti tvorby dat je dosáhnout generování toků dat ve sjednocených formátech a současné publikace se soustředí na vývoj formátů metadat [89], alokaci zdrojů a úpravy granularity [132]. Výzkum síťování se soustředí na omezení real-time provozu a škálovatelnost s cílem dosáhnout distribuovaných řešení a dostatečné škálovatelnosti. V oblasti zpracování a analýzy je dlouhodobým cílem agregace a sumarizace dat. Oblast správy obsahu a prezentace se snaží o tvorbu obsahu pomocí integrace mnoha datových toků, tvorby algoritmů a efektivní prezentaci. Výzkum v této oblasti se soustředí na samotný vývoj algoritmů, vizualizaci [87, 52], uživatelská rozhraní a automatizovanou správu toků. Oblast chápání datových toků se snaží podporovat lidskou správu v pochopení závislostí datových toků a výzkum se soustředí převážně na chytré vyhledávání, vizualizace a predikce změn. Hlavním cílem oblasti bezpečnosti a soukromí je zajistit lidem pocit bezpečí při využívání IoT prostředí. Nejčastějšími výzkumnými tématy v této oblasti jsou pak ochrana soukromí, anonymizace a prevence falzifikace dat.

Zároveň je v článku [78] zmíněno, že podpora datových streamů v reálném čase je v současných komplexních komerčních a open-source IoT plaftormách velmi omezená, což představuje mezeru a výzkumný potenciál. Autoři článku [113] využívají technologii WebSocket k vytvoření řešení, které v reálném čase vizualizuje data z venkovních senzorů s cílem vytvořit plně autonomní systém pro zalévání a zavlažování. Princip real-time je také využit v [55] pro vytvoření IoT frameworku specializovaného na zemědělství, či pro monitorování shlukování obyvatel ve městských oblastech [68]. Vytvořením systému se zobrazením informací o prostředí v reálném čase a aplikací pokročilých pravidel pro regulaci chlazení a vytápění se zabývá článek [93]. S rostoucím výkonem mikrokontrolérů lze WebSocket server integrovat přímo do firmware a vytvořit tak minimalistickou platformu pro streaming senzorických dat [86]. Výhody technologie Apache Kafka spočívají v kombinaci message brokera s diskovými frontami, umožňující logování bez přítomnosti databáze popisuje článek [31]. V kombinaci s technologií Apache Spark pro datovou analytiku poskytuje navržený systém platformu pro datovou analýzu. Všechna řešení se dají kategorizovat jako aplikačně specifická.

3.10.4 Edge computing

Článek [126] ukazuje, že princip Edge computingu je velmi dobře aplikovatelný v oblasti IoT a dokáže přinést významné zlepšení IoT platforem z hlediska komunikační latence, úspory energie, mobility zařízení, vyšší přizpůsobivosti řešení a heterogenity sítě. Dále ovšem uvádí, stejně jako [78], že podpora edge computingu v současných řešeních je nízká a jedním z výzkumných směrů je návrh IoT architektur s podporou výpočtů mimo IoTIM.

Příkladem postupné aplikace edge computingu může být prediktivní analýza výrobního stroje. Takový stroj obsahuje velké množství pohyblivých součástek a motorů. Vibrace stroje se zhoršují s časem a při překročení hraniční meze lze předpokládat brzkou poruchu [53, 61]. Implementaci systému, který by obsluhu na tuto skutečnost upozornil, lze rozdělit do tří fází.

V první fázi je nutné vybrat vhodné senzory vibrací a naměřit množství surových dat při dostatečně velké vzorkovací frekvenci z několika strojů identického typu s odlišnou dobou stáří, případně dlouhodobě měřit jeden stroj. Vhodnými senzory mohou být například akcelerometry, gyroskopy, piezosenzory nebo tenzometry a vzorkovací frekvence měření se může pohybovat v rozsahu 150 Hz až 22 kHz [54]. V první fázi jsou senzory připojeny k zařízení, které komunikuje s bránou. Ta všechny měřené vzorky odesílá do IoTIM a data jsou analyzována a mohou být stanoveny podmínky a hraniční úrovně, kdy dojde k poruše stroje. Již v této fázi může dojít k realizaci klientské aplikaci, která bude stroje monitorovat. Aplikace se dotazuje IoTIM na poslední data senzorů, případně je přijímá v reálném čase. Obsahuje vyvinutý algoritmus a buď přes externí službu (například odesíláním denních reportů na email) nebo zpětně na IoTIM notifikuje operátora stroje aktorem. Algoritmus využívající metod strojového učení je popsán v článku [56]. Tento přístup není ovšem dlouhodobě udržitelný, jelikož při vysoké vzorkovací frekvenci budou se vzrůstajícím množství monitorovaných strojů stoupat nároky na provoz IoTIM i klientské aplikace.

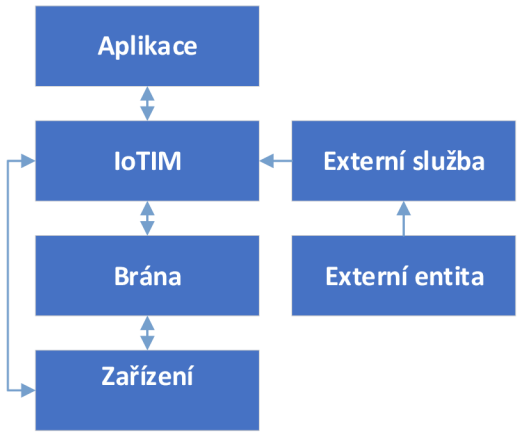
Druhou fází je využití edge computingu, kdy jsou základní matematické operace jako průměrování, vyhlazování nebo převzorkování na nižší frekvenci přesunuty přímo do zařízení. Pokud je algoritmus dostatečně jednoduchý a jeho výpočet není závislý na dalších senzorech, lze ho kompletně přesunout do zařízení. Zařízení pak místo trvalého periodického odesílání nových stavů senzoru již odesílá pouze binární údaj („stroj v pořádku“ nebo „nastane porucha“) či pravděpodobnost poruchy.

Pokud ovšem výpočty vyžadují kooperaci několika zařízení, lze využít edge computing v bráně, kdy jsou surová nebo částečně předzpracovaná data odesílána na bránu. Zde je implementován kompletní algoritmus a do IoTIM jsou pouze odesílány výsledné výpočty. Tento přístup výrazně snižuje zatížení IoTIM a pokud je provozován v cloudu nebo na vzdáleném serveru, tak i přenosové pásmo.

3.10.5 Externí datové zdroje

Na obrázku 3.12 je zobrazena obecná bloková architektura IoT platformy rozšířená o možnost přijímat data nejenom od senzorů, ale i z externích zdrojů. Na taková data lze nahlížet jako na virtuální senzory [78]. Taková funkcionalita je využitelná pro integraci dat poskytovaných třetími stranami. Příkladem mohou být senzory využívající komunikační technologie NB-IoT, Sigfox a LoRaWAN, představené v předchozí části kapitoly. Tyto technologie jsou energeticky úsporné a nabízejí široké portfolio senzorů schopných pracovat i několik let na baterii. Pro získání senzorických dat je nutné využít cloudové aplikace poskytované provozovateli komunikačních sítí. Pokud klientská aplikace vyžaduje data jak z těchto technologií, tak z další IoTI, musí implementovat dvojí aplikační rozhraní a data slučovat interně. Pokud je třeba nad

daty provádět analýzy vyžadující surová historická data v řádech měsíců, znamená to buď pokaždé dotazovat dvě aplikace, nebo data duplikovat v klientské aplikaci. Články [44, 83, 112] shodně tvrdí, že řešením je rozšíření IoTIM vrstvy o aplikační rozhraní, které umožňuje perzistenci senzorických dat z externích zdrojů. Na tato data je pak v systému nahlíženo zcela identicky jako na data pocházející z přímo připojených aktorů a senzorů.



Obrázek 3.12: Bloková IoT architektura rozšířená o externí službu [autor]

3.10.6 Standardizace

Obecnějším problémem řešeným v IoT je nedostatek standardizace. Dle [8] není pravděpodobné, aby nastala podobná situace jako s technologií Wi-Fi, která je dnes standardem vysokorychlostní bezdrátové komunikace nebo DVD, tedy dříve zcela dominantním optickým médiem bez existující konkurence. Je to především ohromným rozsahem IoT projektů a produktů, které sahají od chytrých náramků po chytrá města a autonomní vozidla. V článku jsou zmíněny standardizační dokumenty od ITU, ISO/IEC a IEEE. Každý dokument pohlíží na různé aspekty IoT jinak, často velmi tendenčně. Příkladem budiž IEEE standard doporučující XMPP protokol pro interoperabilní komunikaci zařízení mezi sebou. Tento protokol je postaven na textovém standardu XML, který je díky své povaze nevhodný pro implementaci v jednočipových zařízeních. Publikace [107], která zkoumá přenosové technologie, protokoly a IoT platformy, poukazuje na fakt, že každý přístup má své výhody i nevýhody a nelze jednoznačně vybrat nejlepší, protože to není možné. Publikace [76, 78] také ukazují, že komerční i open-source platformy výše zmíněné standardy z různých důvodů nedodržují.

Jednoduchým příkladem různých přístupů, které mohou být ekvivalentní, je měření teploty prostředí. Jedná-li se o jediné zařízení sloužící k nastavení termostatických hlavic v domě, je vhodná technologie Wi-Fi, kdy je senzor připojen k routeru v domácnosti a informace o teplotě bez přítomnosti brány posílá na IoTIM běžící v PaaS cloudu výrobce mimo dům. Samotný IoTIM může zobrazovat pouze jednoduchou webovou stránku s aktuální teplotou, trendy a možností změnit požadovanou teplotu prostředí. Stejný požadavek na měření teploty prostředí se bude odlišovat u obchodního řetězce, kde je nutné měřit na několika místech. Tehdy se pravděpodobně využije mesh technologii IQRF s bránou a IoTIM běžícím v privátním data centru řetězce. IoTIM musí podporovat role, různé rozsahy pohledů a notifikace o nefunkčním zařízení nebo překročení libovolně nastavitelné prahové hodnoty teploty.

3.10.7 Další vybrané výzkumné problémy

Výše popsané výzkumné problémy jsou pouze malou podmnožinou výzkumných a implementačních otázek v oblasti IoT. Nyní budou bodově popsány další oblasti, vycházející z rešeršních publikací [107, 21, 7, 78], výčet se však nesnaží pokrýt veškeré oblasti. Mezi další výzkumné problémy patří:

- Absence doménově specifických jazyků pro meziplatformní vývoj.
- Návrhy architektur efektivně využívajících principy fog computingu.
- Provádění výkonnostního testování IoT platforem, hledání společných měřitelných metrik, srovnání latence a publikování komparativních studií.
- Optimalizace spotřeby energie pro prodlužování života bezdrátových zařízení, a to např. vývojem nových typů baterií, implementací pokročilých managementů spotřeby nebo vývojem jednodušších protokolů bez zanedbání bezpečnostního hlediska.
- Počty IoT zařízení vzrůstají exponenciálně, tedy škálování IoT sítí bude kritickým požadavkem.
- Bezpečnost jako klíčová vlastnost pro IoT. Kvůli heterogenitě zařízení jsou využívány odlišné bezpečnostní protokoly. Standardizace bude důležitou výzvou. Díky neustále vzrůstající popularitě IoT zařízení se zvyšují snahy o prolomení protokolů s cílem zařízení kompromitovat. Je očekáván vzestup publikací popisující zranitelnosti.
- Správa infrastruktury v reálném čase – různé typy aplikačních domén IoT vyžadují vysokou dostupnost sítě, například zdravotnické aplikace. Výzkumné příležitosti jsou v systémech pro dohled a správu sítě s minimálními režiemi.
- Efektivní slučování časových řad senzorických dat z různých systémů.
- Zpracování dat není dobře integrováno do IoT middlewarů, datové analytické nástroje jsou pouze cloudové, chybějící datové katalogy.
- Katalogy aplikací, zařízení pro IoT nejsou dostupné.

3.11 Existující IoT platformy a jejich architektury

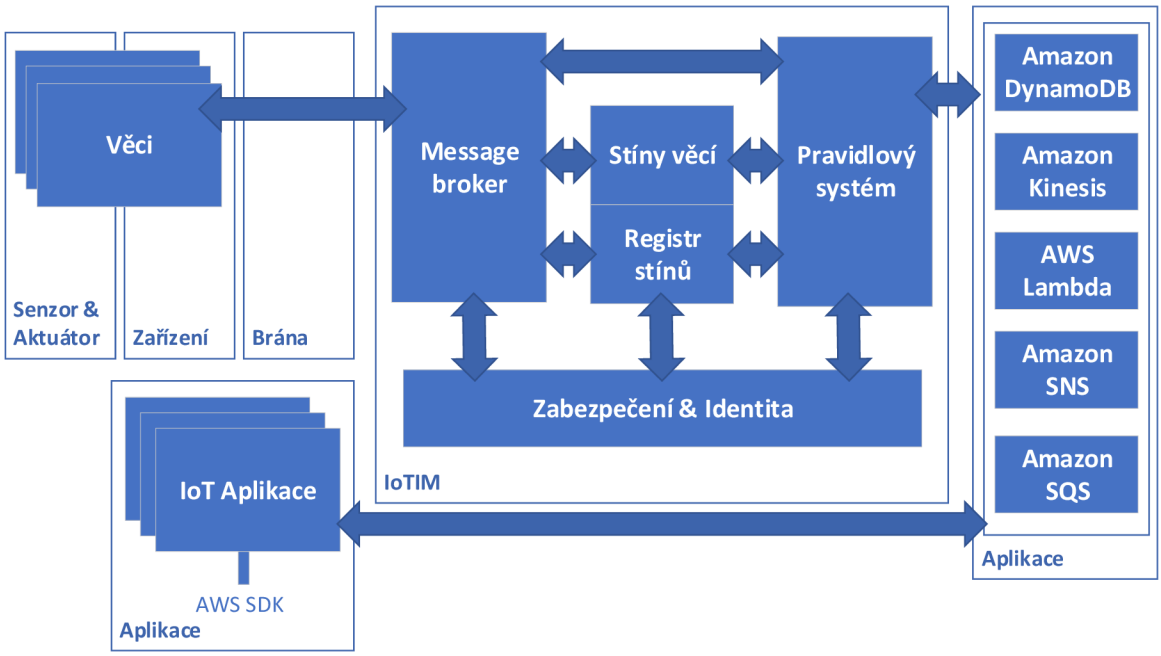
Nyní budou představeny vybrané IoT platformy a krátce popsány jejich interní aplikační architektury. U každé platformy bude zkoumáno, zda nějakým způsobem řeší vybrané výzkumné problémy představené v předchozí kapitole, konkrétně šest detailněji popsaných problémů zabývajících se synchronizací času mezi branami a IoTIM, schopnost bran provádět off-line pravidla a logování událostí, zda IoTIM podporuje přístup k datům z klientských aplikací v reálném čase a podporuje perzistenci externích datových zdrojů a jaký je stav podpory edge computing výpočtů v branách a na zařízeních.

3.11.1 Amazon Web Services IoT Core

Amazon Web Services (AWS) je proprietární cloudový PaaS společnosti Amazon. Dle architektury na obrázku 3.13 je zřejmé, že také využívá referenční architekturu s IoTIM, bránou a zařízeními. AWS interně

nerozlišuje mezi senzorem, aktuátorem a zařízením. Na vše se v systému nahlíží jako na věc (Thing). Každá věc je popsána detailní konfigurací a katalog všech dostupných věcí je umístěn v registru věcí (Things Registry). V systému AWS má každá fyzická věc svůj stín (Device shadows), tedy jeho digitální dvojče. Věci s IP protokolem komunikují s message brokerem (MQTT, HTTP nebo WebSocket protokol) přímo a ostatní přes AWS bránu. Amazon dále nabízí svojí variantu operačního systému FreeRTOS pro mikrokontroléry, který usnadňuje vývoj koncových IP zařízení. V současné době je k dispozici 100 podporovaných zařízení, jedná se ovšem o brány a vývojové kity, které je nutné o senzory či aktory doplnit a vytvořit konfiguraci věcí a zanést je do katalogu.

Uživatelé nemohou přímo přistupovat k aplikačnímu rozhraní IoTIM, ale mohou využít rozhraní dalších Amazon služeb. Mezi ně patří NoSQL databáze Amazon DynamoDB [180], platforma pro kolekci, streamování a analýzu dat Amazon Kinesis [181] nebo Amazon SQS (Simple Queue Service), výpočetní platforma AWS Lambda [182], objektové úložiště Amazon S3 [183] a notifikační služba SNS [184]. Amazon podporuje přístup k datům v reálném čase pomocí služby Kinesis, nicméně se ve skutečnosti nejedná o plně real-time přístup, jelikož data nejsou klientské aplikaci doručována okamžitě, ale je nutné periodicky pokládat dotazy na službu ve formátu podobném SQL. Softwarový vývojový kit pro brány a Amazon implementace FreeRTOS systému pro mikrokonroléry umožňují provádět hraniční (edge) výpočty. Synchronizaci času je nutné řešit externími službami, software brány neumožňuje off-line provoz a IoTIM platforma neumožňuje ukládání dat nepocházejících přímo od zařízení.



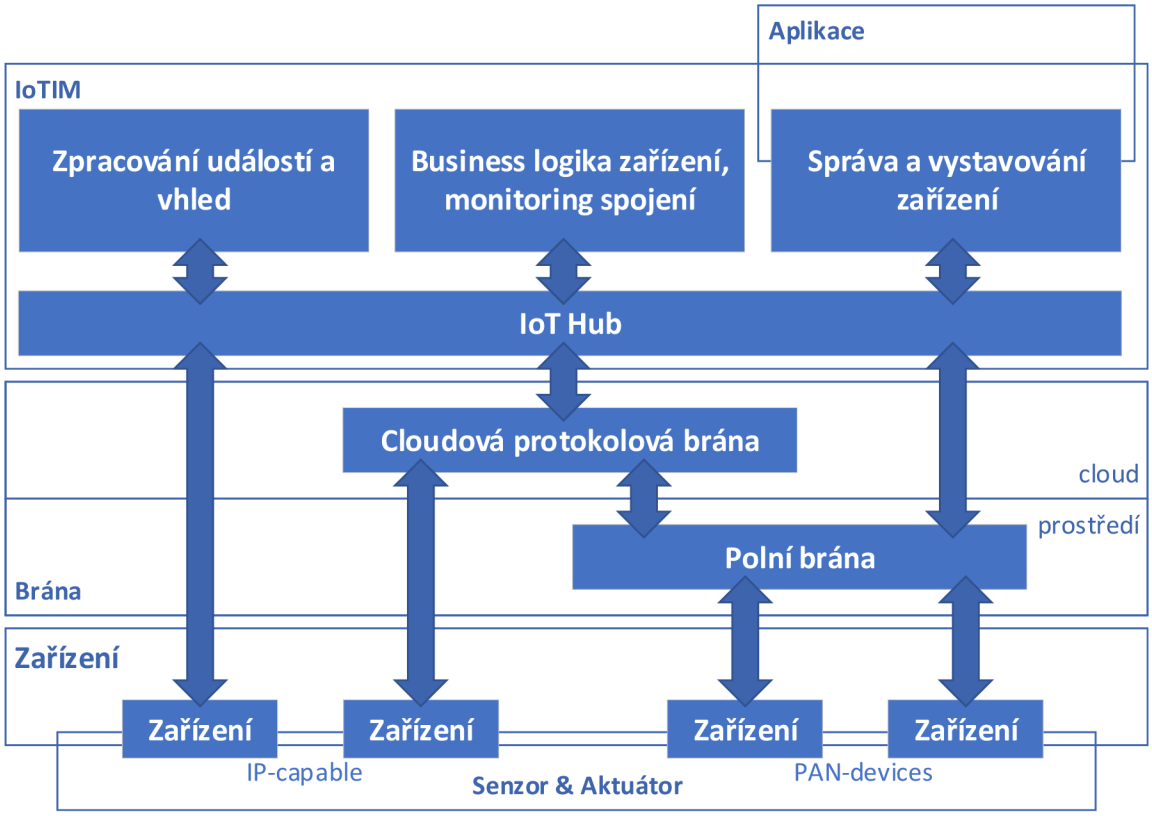
Obrázek 3.13: Architektura Amazon AWS IoT (překresleno z [137])

3.11.2 Microsoft Azure IoT

Azure IoT [138] je cloudový PaaS vyvíjený a spravovaný společností Microsoft. Softwarová architektura zobrazená na obrázku 3.14 plně odpovídá referenčnímu modelu 3.5. Centrálním prvkem Azure IoT je komponenta IoT Hub, ke které se připojují všechny ostatní komponenty architektury. Na IoTIM se vysky-

tují tři další aplikace – Zpracování událostí a vhled (Event Processing and Insight) umožňující perzistenci dat zařízení a jejich analýzu, Business logika zařízení zajišťující monitoring spojení (Device Business Logic, Connectivity Monitoring) a Správa a vystavování zařízení (Application Device Provisioning and Management), která vystavuje aplikační rozhraní pro klientské aplikace.

Zařízení vybavená IP protokolem mohou s hubem komunikovat přímo pomocí HTTP nebo MQTT protokolu, nebo využít prostředníka v podobě cloudové protokolové brány. Jedná se o software, který může pracovat buď na bráně fyzicky nebo na serveru a snižovat zátěž hubu. Zařízení s bez IP protokolu se musí připojit k hardwarové bráně.



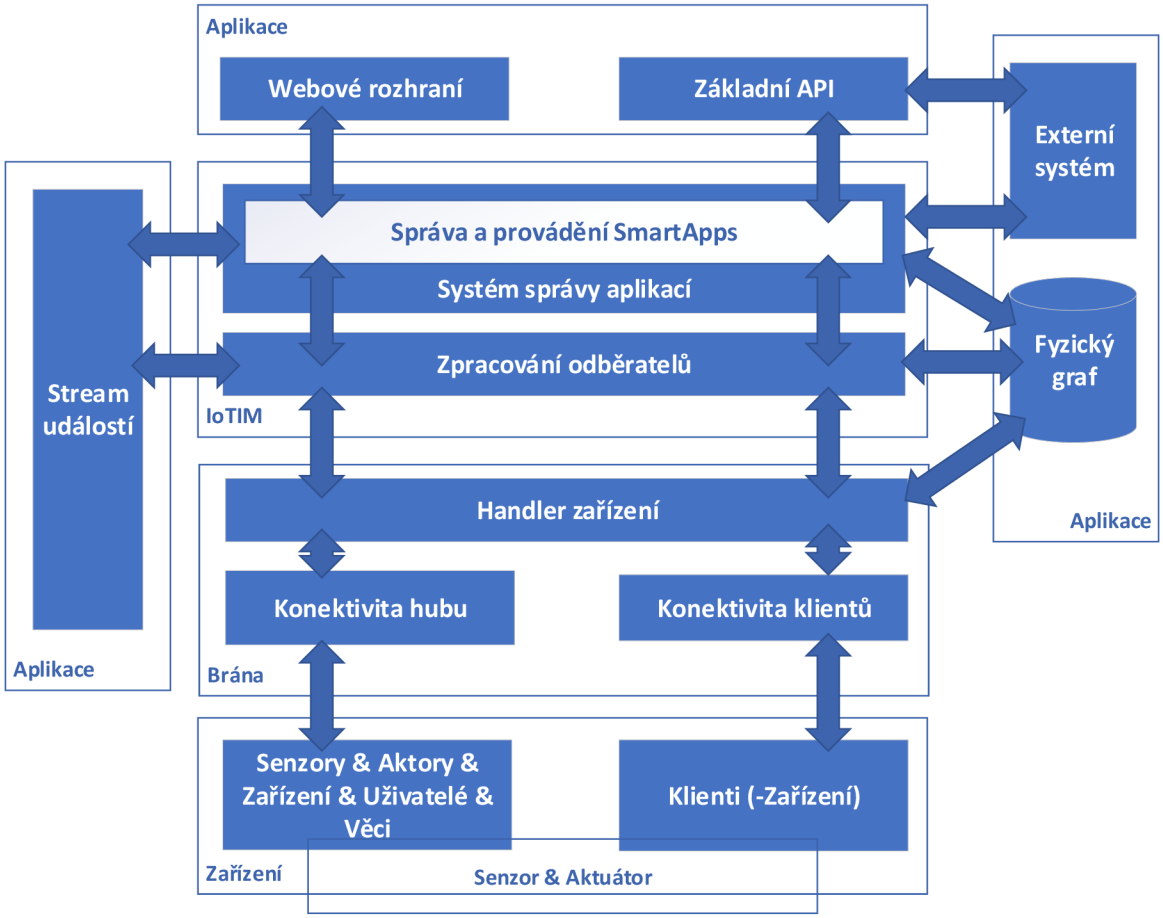
Obrázek 3.14: Architektura Microsoft Azure IoT Hub (překresleno z [138])

V současné době podporuje Azure IoT přes 1 400 různých zařízení [185], ovšem pouze 40 z nich je předem certifikováno a nevyžaduje dodatečné konfigurace vyjma nastavení uživatelského účtu v cloudu. Z těchto 1 400 zařízení je ovšem téměř 500 bran a Microsoft do seznamu zahrnuje i vývojové kity jako Arduino, základní desky, procesory a dotykové panely. Koncových zařízení připojitelných buď přímo k IoTIM přes IP protokoly nebo skrze bránu jsou pouze desítky, a například zařízení se senzorem poletavých částic PM nejsou k dispozici vůbec.

Azure IoT umožňuje logovat stavy zařízení připojených k branám v případě výpadku internetového spojení s IoT hubem. Stejně tak jsou podporovány hraniční výpočty v bráně i zařízeních. Přístup k datům v reálném čase je možný, ovšem buď přes proprietární Azure Stream Analytics nebo přímým přístupem k IoT hubu. IoT hub je v podstatě škálovatelný MQTT broker vyvinutý Microsoftem a přístup k MQTT tématu s daty zařízení znamená nutnost klientských aplikací využívat MQTT protokol. Azure IoT neumožňuje vykonávat pravidla v bráně, neposkytuje API pro perzistenci dat z externích zdrojů a synchronizace času spoléhá na služby v Internetu.

3.11.3 Samsung SmartThings

SmartThings je platforma vyvíjená společností Samsung s výhradním cílem na oblast domácí automatizace. Její architektura je znázorněna na obrázku 3.15 a opět je vidět, že vychází z obecného modelu IoT architektury. Skládá se ze tří klíčových komponent: handleru zařízení, zpracování odběratelů a správy a provádění SmartApps. Všechny další komponenty jsou propojeny právě se SmartApps. SmartThings kombinuje senzory, aktuátory, zařízení, uživatele a věci do jednoho modulu. V tomto modulu se od zbytku odlišují pouze klienti, kteří opět mohou obsahovat senzory a aktuátory. Handler zařízení pak provádí překlad nativně specifických zpráv do standardizované SmartThing události. Spolu s konektivitou hubu a klientů tvoří bránu. Základní funkcionalita platformy je poskytována komponentami zpracování odběratelů a systémem správy aplikací. Ty dohromady tvoří vrstvu IoTIM. Informace o interním uspořádání jednotlivých komponent, programovacích jazycích a službách třetích stran není k dispozici, jelikož se jedná o uzavřený proprietární systém, ovšem s otevřeným základním API ve formě REST. SmartThings obsahují zavedený pravidlový systém, ovšem velmi funkčně omezený, pouze s podporou pravidel souvisejících s ovládáním osvětlení. Ostatní funkcionality zmíněné v předchozí kapitole implementovány nejsou.

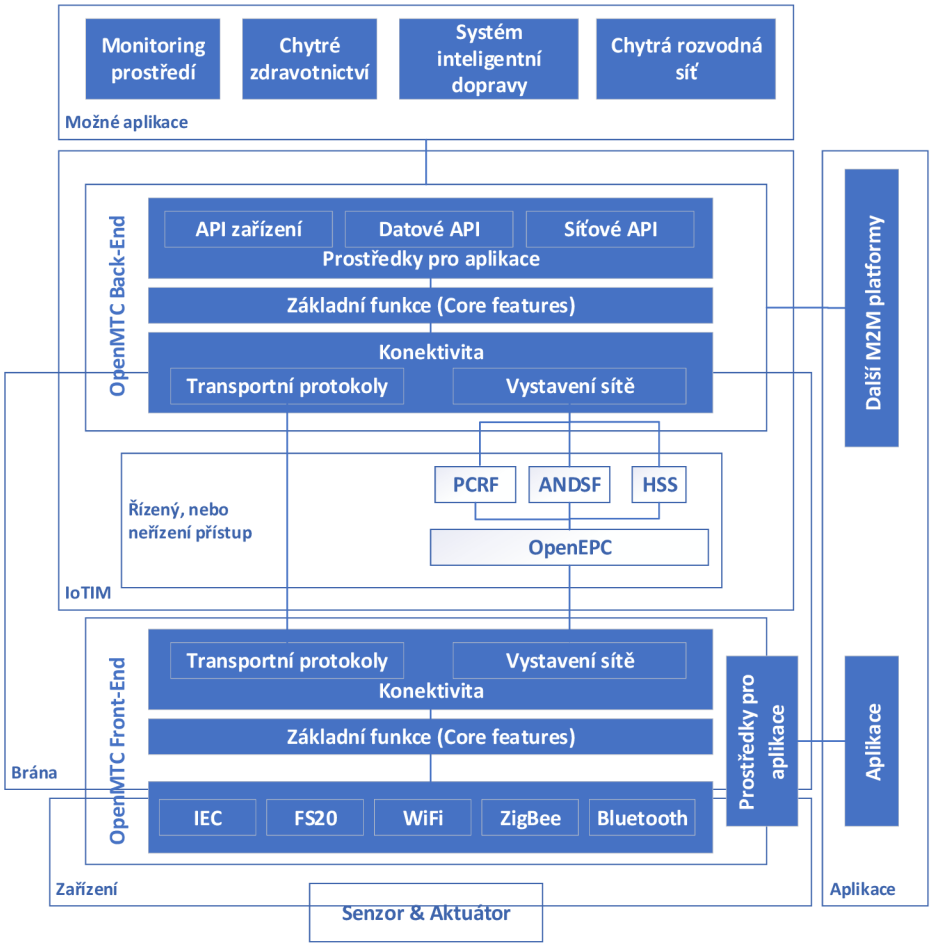


Obrázek 3.15: Architektura SmartThings (překresleno z [139])

3.11.4 OpenMTC

Open Machine Type Communications (OpenMTC) je open-source platforma pro IoT a komunikaci M2M (machine-to-machine, neboli zařízení se zařízením) [34]. Jedná se o referenční implementace oneM2M [186] standardu. Na obrázku 3.16 je znázorněna softwarová architektura platformy. Lze ji rozdělit do dvou hlavních bloků, a to na front-end a back-end. Zařízení s připojenými senzory či aktory spadají pod front-end a ani zařízení s IP technologií nekomunikují s IoTIM přímo, ale vždy přes gateway, což tuto platformu odlišuje od ostatních. Nejnižší vrstvy back-end části, jako je konektivita a základní funkcionalita, jsou sdíleny s front-end částí. Back-end pak vystavuje API pro přístup a správu zařízení a data z monitoringu sítě. Další odlišností od ostatních platforem je možnost využívat služby brány přímo v klientských aplikacích, za předpokladu že aplikace vystavuje své API na veřejné IP adrese.

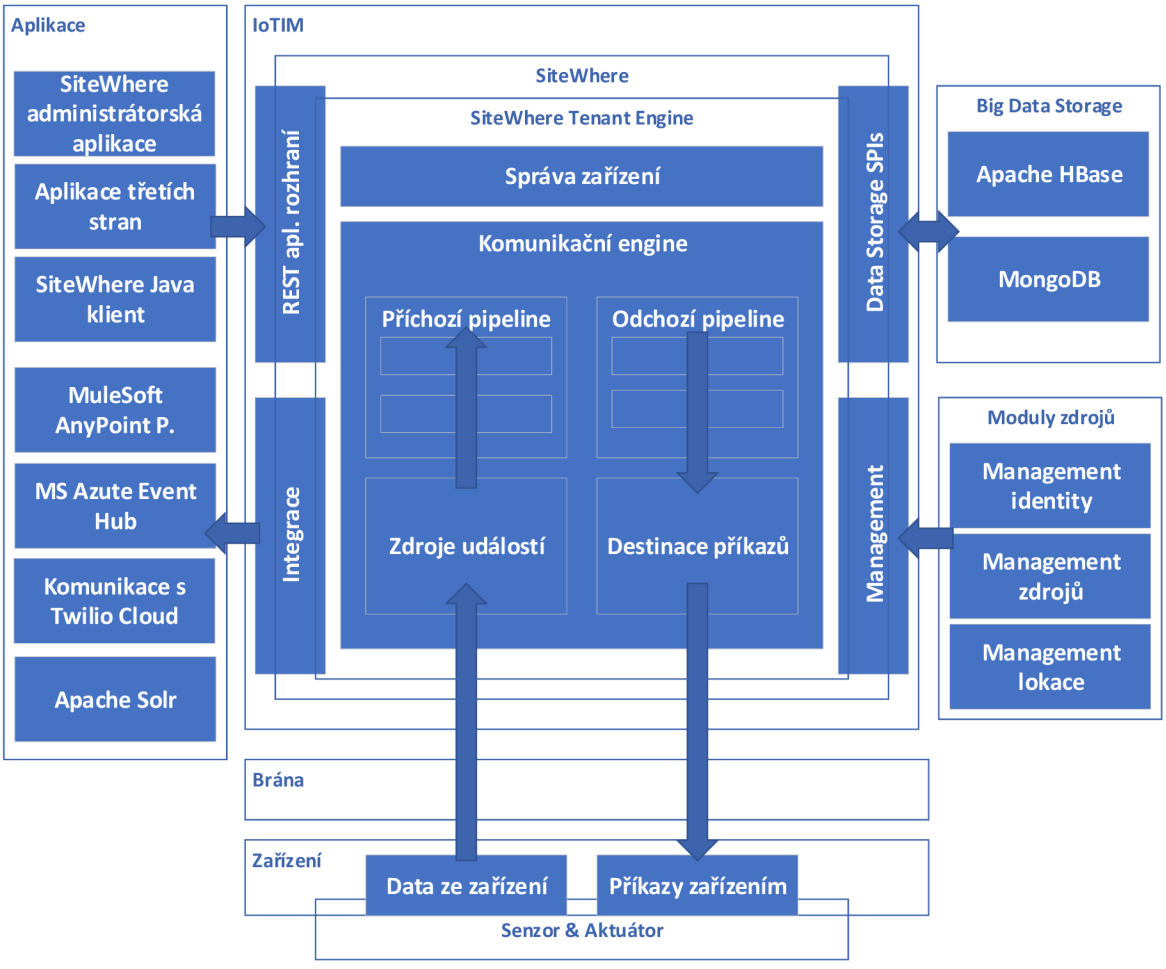
Dle dokumentace platformy podporuje brána zařízení s technologiemi WiFi, ZigBee a dále např. Bluetooth, ovšem seznam podporovaných zařízení nebo alespoň výrobců nelze na webových stránkách projektu [140] ani dokumentaci [187] nalézt. Ovšem autoři článku [2] platformu využili pro systém sledování polohy automobilu v reálném čase, článek [114] OpenMTC využívá k realizaci elektrokardiografu se vzdáleným dohledem a článek [46] popisuje využití platformy k verifikaci senzorického zařízení pro detekci lesního požáru. Platforma neimplementuje žádnou funkcionalitu popsanou v předchozí kapitole.



Obrázek 3.16: Architektura OpenMTC (překresleno z [140])

3.11.5 SiteWhere

Softwarová architektura open-source platformy SiteWhere [188] je vyobrazena v diagramu 3.17 a také odpovídá referenčnímu modelu 3.5. Svoji architekturou je řešení odlišné tím, že nepokrývá bránu. Jsou pouze popsány technické specifikace, které musí implementace brány (případně vybavená MQTT protokolem) splňovat. Celá platforma se tedy plně soustřeďuje na IoT integrační middleware. SiteWhere jako jediné z představených řešení uvádí v diagramu architektury nájemníky (anglicky tenant). Nájemník vlastní svoji unikátní instanci celého systému[5]. Při vytváření nového nájemníka v systému je nutné vytvořit jako první super-administrátorský účet, který spravuje celou instanci. Příkladem může být využití systému v panelovém domě, kde každý fyzický nájemník bytu vlastní instanci a přidává do svého odděleného systému obyvatele bytu, každého s různým oprávněním. Dokumentace [141] popisující nasazení systému na server je velice podrobná, stejně tak i popis REST API využitelného pro klientské aplikace a administrátorské rozhraní. Ovšem i když se systém nachází ve stabilní verzi 2, která je aktivně vyvíjena a zdokonalována, nebylo možné nalézt seznam podporovaných zařízení ani skutečné případy využití toho systému. SiteWhere také neimplementuje žádnou funkcionalitu popsanou v předchozí kapitole.

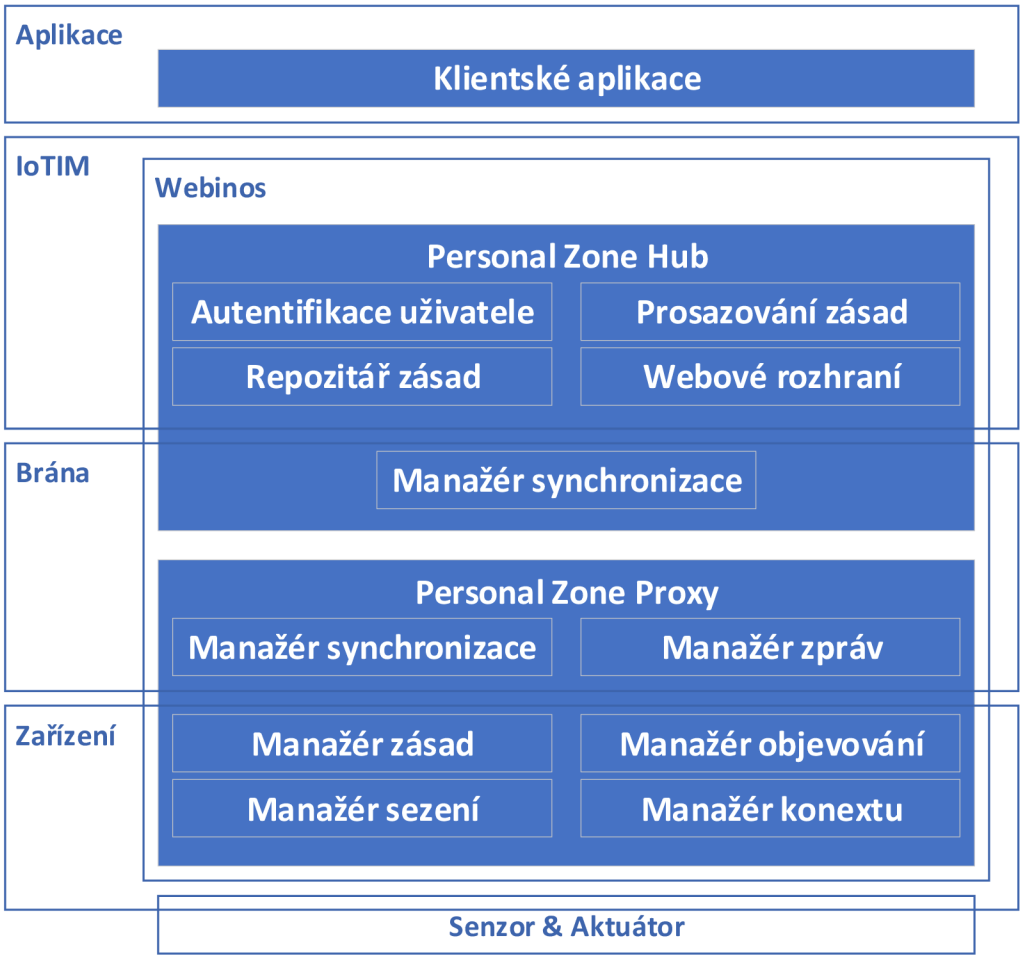


Obrázek 3.17: Architektura SiteWhere (překresleno z [141])

[5] SiteWhere využívá orchestrační platformu Kubernetes.

3.11.6 Webinos

Platforma Webinos vznikla v rámci projektu Evropské unie FP7 a jejím cílem je poskytnout zabezpečený framework pro sběr dat z osobních zařízení s možností komunikace s dalšími jednotlivci, nebo poskytování dat třetím stranám. Z těchto důvodů jsou jednotlivé aplikační bloky řešení vystaveny odlišně a soustředí se právě na osobu. Ovšem i zde lze vidět shodu s referenčním modelem IoT řešení, viz obrázek 3.18. Hlavními komponentami jsou Personal Zone Hub (PZH) a Personal Zone Proxy (PZP).



Obrázek 3.18: Architektura systému Webinos (překresleno z [39])

Na systému Webinos je vhodné ukázat nutnost kontroly nejenom vědeckých publikací, ale také webových stránek projektů, dokumentace API a repositářů zdrojových kódů. Tento systém je popsán v původním autorském článku [39]) z roku 2012, dále pak ve dvou publikacích [28, 133] z roku 2016 a další publikacích [45, 14] z roku 2018. Celkově lze nalézt v součtu přes 15 publikací v databázích Scopus a WoS od let 2012 po 2020, které se tímto systémem zabývají. Nicméně webové stránky projektu jsou v současné době zcela nedostupné, což nemusí nic implikovat a může se jednat o krátkodobý problém způsobený certifikátem. Ovšem při prozkoumání zdrojových kódů projektu veřejně dostupných na GIT platformě GitHub [189] bylo zjištěno, že většina repozitářů aplikace je bez aktivity od roku 2013, pouze u některých došlo v následujících letech k úpravám textů. Z toho převážná část stežejních repozitářů obsahuje pouze prázné soubory zdrojových kódů, nebo prvotní commity bez jakékoliv funkcionality. Pokud vývojáři

nevyužívají interní GIT repozitáře, lze projekt prohlásit za neaktivní. Ve veřejně dostupných repozitářích nebyly nalezeny žádné známky podpory funkcionalit z předchozí kapitoly.

3.11.7 Další IoT platformy

Výše popsané platformy nejsou jediné, které lze využít. V článku [78] je popsáno celkem 39 platforem, které jsou využitelné v oblasti senzorických řešení Internetu věcí, nebo alespoň oblastí domácí automatizace, tedy části IoT. Je zde zmíněno, že spousta řešení není dostatečně dospělá z hlediska softwaru pro praktické využití. I tak není uvedený výčet kompletní, protože neobsahuje výše popsané platformy Microsoft Azure, Amazon AWS, SmartThings, ani například populární řešení FIWARE nebo IBM Watson. Nyní budou krátce představeny další vybrané platformy.

FIWARE

Na open-source projekt FIWARE [190] nelze pohlížet jako na řešení, které lze nasadit na server, nainstalovat software brány, připojit zařízení a začít využívat. Jedná se o souhrn dokumentací, návodů a repozitářů mikroslužeb, které mají usnadnit vývoj a nasazení specializovaných řešení na míru, například z oblasti chytrých měst, zdravotnictví, energetiky, zemědělství či výroby a logistiky.

OpenIoT a IoT-Framework

Projekt OpenIoT [191] se soustředí na poskytnutí referenční open-source integračního middleware psaného v jazyce Java. Stejně jako projekt IoT-Framework [192] vznikl jako výsledek grantového projektu s cílem zlepšit stav poznání v oblasti Internetu věcí.

IBM Watson

I společnost IBM nabízí vlastní proprietární cloud platformu pro realizaci IoT řešení, jedná se o přímého konkurenta výše předstaných řešení od společnosti Amazon a Microsoft. Oproti nim se IBM soustředí výhradně na IoTIM a analytické nástroje. Nenabízí ihned využitelná zařízení se senzoru ani hotové brány.

ThingSpeak

ThingSpeak [193] je proprietární cloudová platforma vyvíjená a provozovaná společnosti Mathworks, autorů programovacího jazyka MATLAB. Ten je velmi často využíván výzkumníky k vývoji algoritmů, vytvářením simulací a analýz. Cílem ThingsSpeak je poskytnout jednoduché API pro MATLAB, které výzkumníkům umožní získávat senzorická data a analyzovat je.

Node-RED

Node-RED [194] je serverová platforma vyvíjená v javascriptovém frameworku NodeJS s cílem pokytnout řešení které integruje datové proudy a na jejich základě uživatelsky vytvořených pravidel provádí reakce s aktory nebo volá další externí služby.

3.12 Analýza existujících řešení z hlediska výzkumných problémů

V předchozích částech kapitoly byly představeny vybrané výzkumné oblasti a problémy řešené v oblasti Internetu věcí a popsány existující IoT platformy, a to jak komerční tak open-source.

V tabulce 3.5 jsou řešení srovnány z hlediska výzkumných problémů. Jmenovitě podporu offline pravidel a logování v bráně, synchronizace času mezi branami a IoTIM, podporou klientských aplikací s daty senzorů v reálném čase, podporou edge computingu na bráně a koncových zařízení a podporou externích zdrojů. Sloupec vlastní hostování ukazuje, zda-li je dotyčnou platformu možné provozovat na vlastním serveru v lokální síti, bez nutnosti přenášet data přes Internet. Platforma SmartThing jako jediná umožňuje provádět zjednodušená automatizační pravidla přímo v bráně, není-li dostupné spojení s cloudovým IoTIM. Platformy Azure a IBM Watson umožňují logovat stavy a hodnoty senzorů pokud není k dispozici internetové spojení s IoTIM. Žádná ze zkoumaných platforem neimplementuje přesnou synchronizaci času mezi branami a IoTIM. Pro mnoho aplikačních oblastí Internetu věcí není tato funkcionalita nutná a rozdíly v časových známkách i v jednotkách sekund jsou přípustné. Je-li v konkrétní aplikaci vyžadováno zkoumání krátkých jevů z mnoha zdrojů (bran) souběžně, nejsou současná řešení příliš vhodná a bude nezbytné s daty manipulovat. Microsoft Azure IoT a IBM Watson podporují analýzu dat v reálném čase, ovšem je možné ji provádět pouze v jejich proprietárních prostředích bez možnosti externího propojení. Amazon Web Services IoT Core podporují získávání a analýzu dat v reálném čase, ovšem pouze synchronním dotazováním, nikoliv plně asynchronním přístupem. Platformy Azure, Amazon, FIWARE a IBM Watson nabízejí programové prostředky umožňující provádět výpočty v zařízeních a branách, čímž mohou snížit objem a granularitu přenášených dat směrem k IoTIM a také její výpočetní zátěž. IoT-Framework a IBM Watson jako jediné nabízejí možnost perzistence dat, která nepocházejí přímo od zařízení, ale z externích zdrojů.

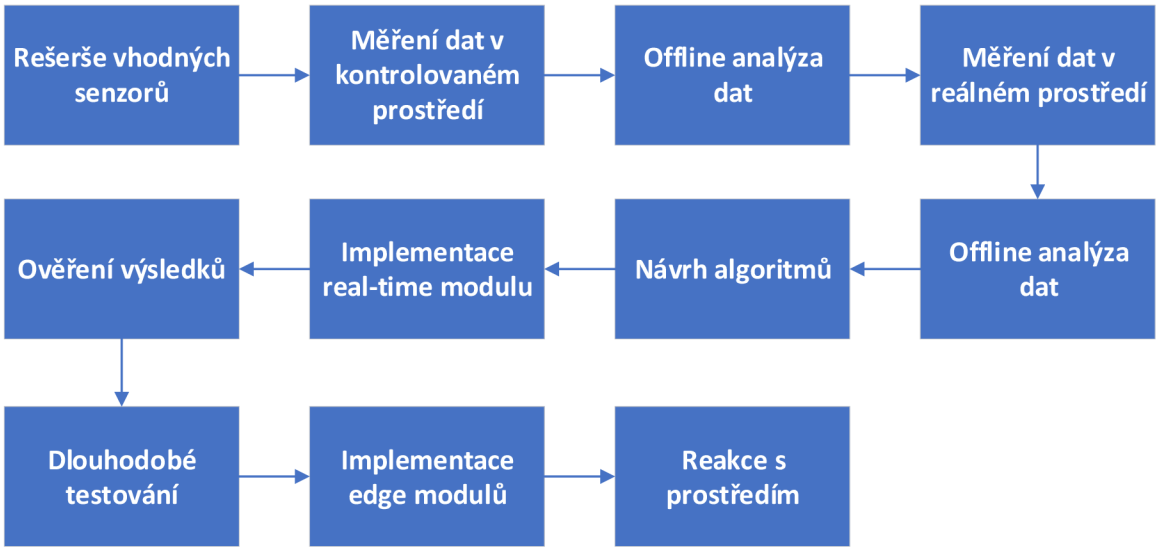
Tabulka 3.5: Srovnání IoT platforem z hlediska řešených výzkumných problémů [autor]

	Vlastní hostování	**Offline pravidla**	**Offline události**	**Synch. času**	**RT API**	**Edge computing**	**Externí zdroje**
Azure	Ne	Ne	Ano	Ne	Prop.	Zařízení, Brána	Ne
Amazon	Ne	Ne	Ne	Ne	Ano	Zařízení, Brána	Ne
SmartThings	Ne	Ano	Ne	Ne	Ne	Ne	Ne
SiteWhere	Ano	Ne	Ne	Ne	Ne	Ne	Ne
openMTC	Ano	Ne	Ne	Ne	Ne	Ne	Ne
Webinos	Nezjištěno	Ne	Ne	Ne	Ne	Nezjištěno	Ne
FIWARE	Ano	Ne	Ne	Ne	Ne	Zařízení, Brána	Ne
OpenIoT	Ano	Ne	Ne	Ne	Ne	Ne	Ne
IoT-Framework	Ano	Ne	Ne	Ne	Ne	Ne	Ano
IBM Watson	Ne	Ne	Ano	Ne	Prop.	Zařízení, Brána	Ano
ThingSpeak	Ne	Ne	Ne	Ne	Prop.	Ne	Ne
Node-RED	Ano	Ne	Ne	Ne	Ne	Ne	Ne

3.13 Potřeba nové architektury

V předchozích částech této kapitoly bylo detailně představeno šest výzkumných problémů v oblasti IoT, jmenovitě podpora off-line pravidel a logování událostí v bráně, synchronizace času mezi IoTIM a branami, získávání dat v reálném čase, edge computing a perzistence dat z externích zdrojů. Analýzou současných IoT platforem bylo zjištěno, že neexistuje taková platforma, která by nějakým způsobem implementovala řešení všech šesti těchto problematik. Některé platformy řeší pouze vybrané části a některé žádné. Návrhem nové architektury IoT platformy, která řeší všech šest problematik zároveň, vznikne řešení s přidanými hodnotami, zejména pro použití ve výzkumu.

Na obrázku 3.19 je znázorněna posloupnost činností, které je nutné provést pro realizaci senzorického projektu od rešerše až po zkušební provoz. Kroky nutné pro komercializaci zahrnuté nejsou, jelikož jsou aplikačně specifické. Diagram je zjednodušený, protože v podstatě jakémkoliv kroku může nastat nutnost vrátit se zpět a řešení přepracovat v případě nedostatečných výsledků.



Obrázek 3.19: Přechod z výzkumného experimentu na IoT projekt [autor]

Článek [129] analyzuje současný stav řešení a algoritmů pro detekci pádů osoby. Jako jednu z výzkumných výzev a příležitostí zmiňuje aplikování algoritmů ve skutečném prostředí, protože spousta z nich vyhodnocuje pád spolehlivě pouze v laboratorním prostředí. Publikace [111, 51], popisují způsob neobtěžujícího měření dechové a tepové frekvence. Zde se opět jedná o sběr surových dat a jejich analýzu. Implementované algoritmy nejsou dále rozvíjeny a zkoušeny v reálném prostředí. Článek [50] ukazuje experiment, kdy z tříosého akcelerometru lze extrapolovat informace o pohybu lidí uvnitř domu. Opět se jedná o přístup, který je možné v IoT systému řešícím problémy popsané v kapitole 3.10 realizovat až do stavu plně real-time řešení. Publikace [60] ukazuje využití metod strojového učení pro zjišťování deprese seniorů trpícími chronickými nemocemi. Přesnost algoritmu je 96% v laboratorním prostředí a s IoT systémem implementujícím výzkumné problémy je opět možné experiment převést na funkční řešení. V jakých krocích diagramu 3.19 se využijí které chybějící funkcionality současných IoT platforem bude nyní bodově popsáno.

1. Rešerše vhodných senzorů - zde musí nová architektura poskytovat širokou podporu různých environmentálních senzorů.
2. Měření dat v kontrolovaném prostředí - proces, kdy jsou měřena data. Synchronizace času bran zde garantuje koherenci dat, je-li vyžadováno měření v rozsáhlejším prostředí s více branami.
3. Offline analýza dat - aplikační rozhraní IoTIM poskytuje prostředky pro získání historických dat.
4. Měření dat v reálném prostředí - offline logování v bráně zabrání ztrátě surových dat, dojde li k problémům se spojením s IoTIM.
5. Offline analýza dat - opět je využito API IoTIM.
6. Návrh algoritmů - po ověření funkčnosti řešení v reálném prostředí je možné ve vybraném jazyce implementovat algoritmy.
7. Implementace real-time modulu - podpora real-time API na IoTIM umožňuje v reálném čase provádět výpočty a díky externím zdrojům výsledky zpětně perzistovat.
8. Ověření výsledků - podpora externích zdrojů dat umožňuje do systému integrovat různé přístroje, které se použijí k validaci výsledků a potvrzení funkčnosti.
9. Dlouhodobé testování - IoTIM slouží jako sběrný systém surových dat a dat vypočtenými moduly přes real-time API.
10. Implementace edge modulů - podpora edge computingu v bráně umožní přesun algoritmů směrem níže a přispěje v výrazné úspoře produkovaných dat. Díky offline logování jsou vypočtené hodnoty přeneseny k IoTIM při znovu navázání spojení.
11. Reakce s prostředím - dle projektu je možné vytvořit různá pravidla která budou ovlivňovat stav aktuátorů. Podpora offline pravidel v bráně zajistí reakce v i bez dostupnosti IoTIM.

Doplnění funkcionality brány o pravidlový systém a logování událostí, které nastanou během nedostupnosti IoTIM znamená náročné refaktorování softwaru brány a změnu aplikační logiky IoTIM. Dalším postupem je navrhnout architekturu nové IoT platformy, která umožní implementovat následující funkcionalitu:

- Synchronizace času mezi branou a IoT integračním middlewarem,
- logování nastalých událostí v bráně při výpadku spojení s IoTIM,
- autonomní vykonávání uživatelských pravidel v bráně během při výpadku spojení s IoTIM,
- podpora edge computingu v bráně a zařízeních,
- prostředky pro přenos dat senzorů do klientských aplikací v reálném čase,
- možnost perzistence externích dat v IoTIM.

Mezi podružné požadavky na novou architekturu je využití open-source technologií a licencování vzniklých aplikací také jako open-source. IoT middleware nového řešení musí být nasaditelný ve scénářích popsaných v kapitole 3.8.1, tedy na dedikovaném serveru, serveru virtuálním i na IaaS. Běh jako PaaS je možný pouze při využití takových technologií, které poskytoval cloud computing platformy podporuje. Vytvořením nové architektury se zahrnutím synchronizace času, branami rozšířenými o možnost pravidel, logování a edge computing, aplikačním rozhraním pro analýzu dat v reálném čase a externími daty vznikne řešení umožňující široké spektrum aplikačního využití. Díky možnosti logování událostí lze bránu využít jako sběrný systém dat ze zařízení i v prostředích bez dostupného internetového spojení. Případně jako velmi jednoduchou samostatnou automatizační jednotku.

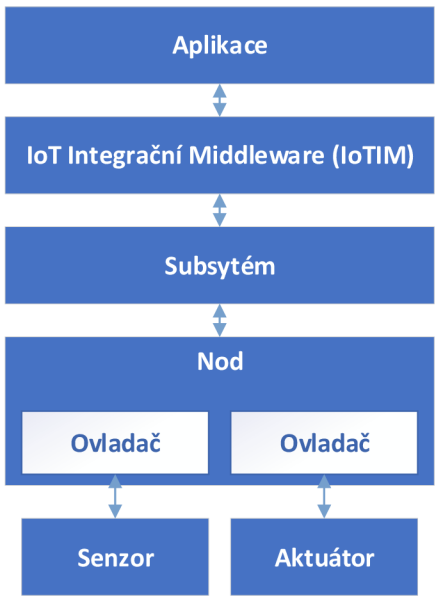
4

Návrh nového řešení

Tato kapitola představuje komplexní návrh nového řešení. Nejprve je zmíněna potřeba nového řešení, specifikovány dílčí požadavky a popsána jeho fyzická architektura. Poté je představen navržený datový model a postupně jsou navrženy aplikace IoT middleware, subsystému a nodů.

4.1 Fyzická architektura

Na obrázku 4.1 je znázorněna fyzická architektura nového řešení. Fyzickými vrstvami je identická jako řešení představená v předchozích částech, s tím rozdílem, že brána je nahrazena subsystémem a zařízení nody. Označení subsystém bylo zvoleno ze dvou důvodů: Prvním je snadnější čitelnost práce, kdy je zřejmé, zda se hovoří o existujícím řešením nebo o řešením novém. Druhým důvodem je fakt, že žádná existující implementace brány nepodporuje souběžně vykonávání pravidel, logování a edge computing. Subsystém obsahuje propracovanější aplikační logiku a proto je nazýván subsystém, jako podružný systém IoTIM.

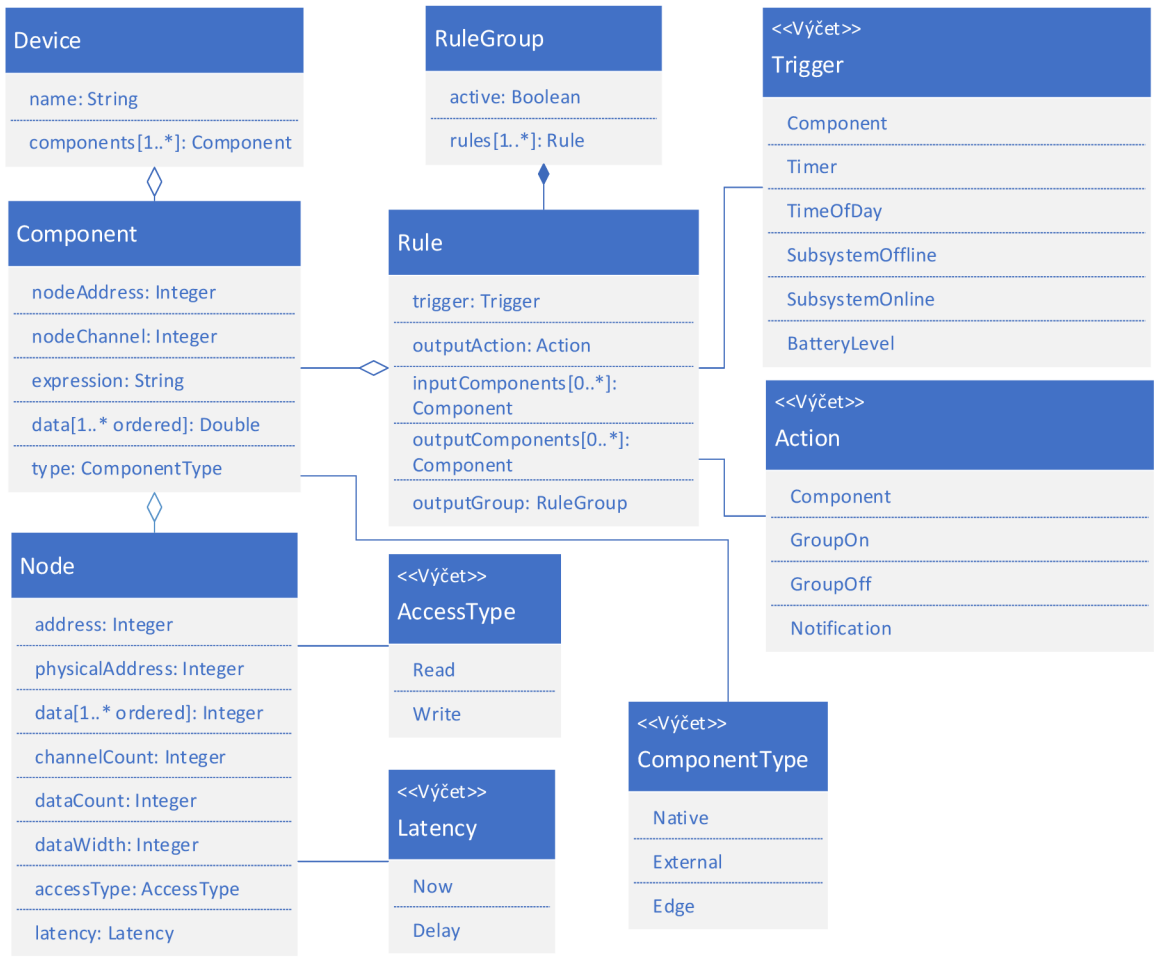


Obrázek 4.1: Fyzická architektura nového řešení [autor]

Zařízení jsou nazývána nody také z důvodů odlišení textu práce od existujících koncových zařízení se senzory či aktory, ale také z důvodu heterogenity. IoT platformy jsou v principu heterogenní, protože podporují odlišné protokoly a široké množství navzájem odlišných zařízení. Samotná zařízení jsou ovšem často homogenní v tom smyslu, že obsahují pouze jediný senzor, nebo několik identických senzorů. Nody jsou navrženy tak, aby bylo možné s pomocí vhodného protokolu a translační vrstvy podporovat více odlišných senzorů.

4.2 Datový model řešení

Datovým modelem jsou myšleny třídy, které uchovávají data v objektové struktuře. Jedná se o třídu `Node` (nod), `Component` (komponenta), `Rule` (pravidlo) a `Device` (zařízení). Jejich vztah je znázorněn v diagramu 4.2.

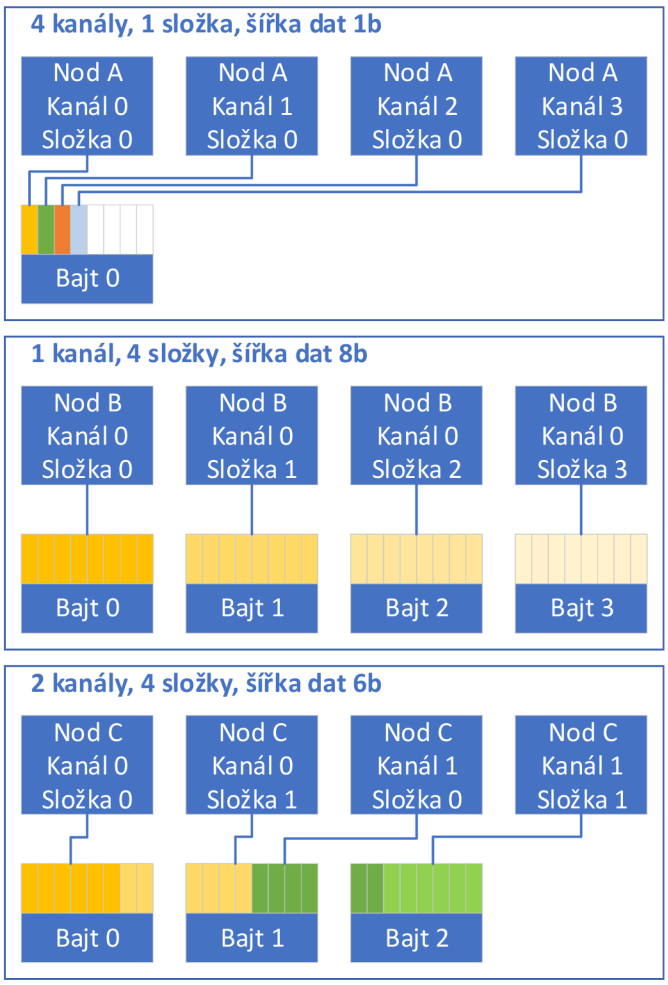


Obrázek 4.2: Datový model řešení [autor]

Nody mají těsnou návaznost na nejnižší vrstvu představené architektury. Každý nod je v rámci subsystému identifikován svou adresou a vystavuje své charakteristiky. Ty nespecifikují přesný typ zařízení (konkrétní senzor nebo aktuátor), ale pouze formát a směr toku dat a nezbytná metadata. Při adresování nového nodu pak právě díky charakteristice aplikace ví, jak má po adresaci s nodem pracovat.

Charakteristika je definována šířkou dat, počtem složek, přístupem, zpožděním, časem zpoždění, verzí firmware a popiskem. Šířka dat vyjadřuje počet datových bitů, které nod přijímá, případně odesílá. Šířka dat 1 znamená binární informaci 0 nebo 1, šířka dat 16 číslo v rozsahu 0 až 65 335. Na těchto nejnižších vrstvách řešení není brána v potaz znaménkovost dat ani desetinná čísla. Počet složek dat pak značí kolikrát se data o určité šířce opakují pro jeden kanál nodu. Počet složek 1 je tedy jednorozměrná informace, počet složek 3 pak tři různá čísla pro jeden kanál nodu. Přístup může nabývat pouze hodnoty `READ` (z nodu se data čtou) a `WRITE` (do nodu se data zapisují). Zbývající atributy pak platí pouze pro typy nodů kategorie `READ`. Typ se zpožděním `NOW` znamená snahu o vyčtení stavu nodu co nejčastěji. Typ zpoždění `DELAYED` pak využívá dodatečný atribut, udávající nejvyšší možnou rychlost čtení dat z nodu s tímto typem. Atribut zpoždění značí, jak často bude subsystém nodu posílat pokyn pro měření a získání dat. Nod se zpožděním 100 znamená periodu vyčítání dat 100 ms, která se interně dvojí. Jednou za 50 ms je nodu poslán příkaz k měření a za dalších 50 ms pak příkaz k odeslání naměřených dat subsystému.

Počet kanálů značí počet senzorů nebo aktorů, které jsou k nodu připojeny. Subsystém u objektu nodu uchovává další informace, jako enumeraci `Connection`, indikující komunikační technologii nodu. Pro práci s bezdrátovými technologiemi je dále vyžadována fyzická adresa nodu. Mezi několik dalších pomocných proměnných, které nejsou v diagramu znázorněny, patří například časové známky měření, indikace požadavku zapsání nového stavu ze subsystému do nodu atd.

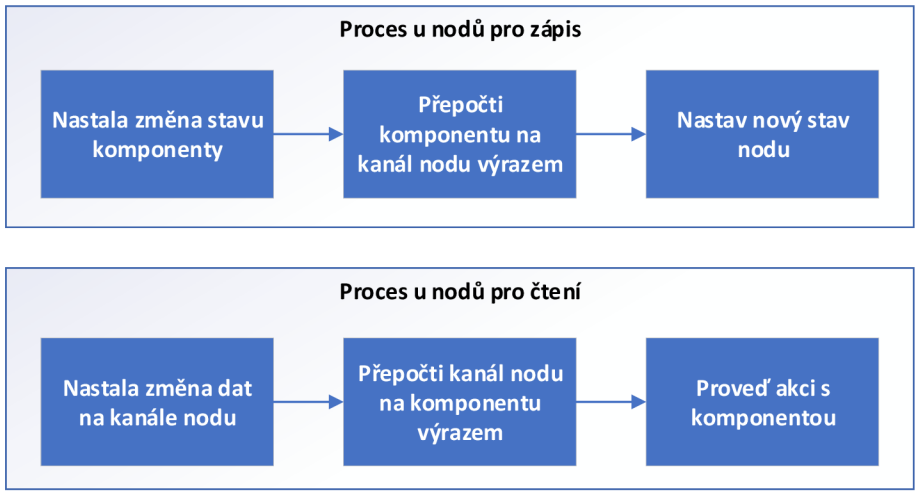


Obrázek 4.3: Vztah mezi typem nodu a jeho kanály [autor]

Význam charakteristik nodů je pro lepší pochopení znázorněn v diagramu 4.3. Jednotlivé kanály nodu jsou odlišeny barvou, protože se jedná o odlišné zařízení k nodu připojené. Stínováním barvy jsou odlišeny i jednotlivé složky typu. Nod A je charakterizován šířkou dat 1 bit a jednou složkou a čtyřmi kanály. Takový nod může mít na svých svorkách připojeny například 4 spínače. Jelikož každý spínač může být logickou součástí jiného zařízení, je každý na svém vlastním kanálu. Nod B je charakterizován šířkou dat 8 bitů a čtyřmi složkami. Příkladem takového nodu budiž LED pásek s červenou, zelenou, modrou a bílou LED. Protože LED pásek je zařízení jako celek, je celý přítomen na jednom kanálů. Posledním příkladem je pak nod C, který má šířku dat 6 bitů a dvě složky. Nod C hardwarově implementuje dva kanály, na každém z nich pracuje s dvěma složkami, každý o délce 6 bitů. Může se jednat například o nod obsahující dva kombinované senzory, měřící teplotu v vlhkost v domácím prostředí.

4.2.1 Převod komponent a nodů

Výše popsané objekty nodu slouží jako nejnižší vrstva, kterou aplikace potřebuje k vyčtení a načtení nových dat nodů. Na této vrstvě se specifikuje pouze formát dat, směr a frekvence komunikace. Mějme nod s firmwarem naprogramovaným pro jeden senzor teploty a vlhkosti DHT22. Ten lze při dosažení nejvyšší přesnosti spolehlivě vyčítat každé 4 sekundy. Dokumentace senzoru dále uvádí, že na první datové složce jsou informace o teplotě s jedním desetinným místem a na druhé složce dat relativní vlhkost také na jedno desetinné místo. Nody ale se subsystémem komunikují striktně celočíselně. Protokol senzoru samotného také pracuje celočíselně, proto vyčtený stav (225; 523) ve skutečnosti značí teplotu 22,5 stupňů Celsia a 52,3% relativní vlhkosti. V případě teploty pod bodem mrazu se nastaví nejvyšší bit jako znaménkový a teplota -1,0 °C bude přijata jako 65525.



Obrázek 4.4: Proces převodu nodu a komponenty [autor]

Takové hodnoty nejsou v žádném případě lehce interpretovatelné pro uživatele. Subsystém proto musí obsahuj proces, který dokáže surová data z nodu převést na hodnotu měřené veličiny (komponentu) a naopak. Proces transformace pro nody `READ` a `WRITE` typu je znázorněn na obrázku 4.4. V případe výše zmíněného nodu s DHT22 senzorem se samozřejmě jedná o nod pro čtení se zpožděním a při přepočtu je třeba provést úpravu teploty a vlhkosti na jedno desetinné místo a zkontrolovat záporné teploty. Aplikace

subsystému musí pro tyto účely využívat knihovnu určenou pro evaluaci výrazů, případně realizovat vlastní implementaci.

Pro senzor DHT22 lze aplikovat převod:

$$b = \begin{cases} \dfrac{65\,535 - a}{10} & \text{pro } a > 32\,768, \\ \dfrac{a}{10} & \text{jinak,} \end{cases}$$

kde proměnná a značí aktuální stav nodu a hodnota b je výstupní hodnota. Tento výpočet se aplikuje na obě složky senzoru a výsledkem budou dvě desetinná čísla. Vybraná knihovna nebo implementace musí podporovat podmínky pro větvení výpočtů. Většina běžných programovacích jazyků má knihovny pro evaluaci matematických výrazů již k dispozici a není nutné je vyvíjet [27]. Komponenta má vždycky takový počet složek, kolik má kanál nodu složek. Komponenta je pak úzce spjata s kanálem nodu, a proto je možné mít na vícekanálovém nodu různé výrazy pro různé kanály. Příkladem budiž šestikanálový nod s analogovými senzory s předpokladem AD převodníku s diskretizací na 10 bitů. Na prvních třech kanálech se nachází analogový akcelerometr, na čtvrtém analogový senzor teploty LM35, na pátém analogový senzor vlhkosti Honeywell HIH-4000 a na posledním potenciometr. Celkem lze tedy aplikovat šest výrazů:

Kanál 1: a

Kanál 2: a

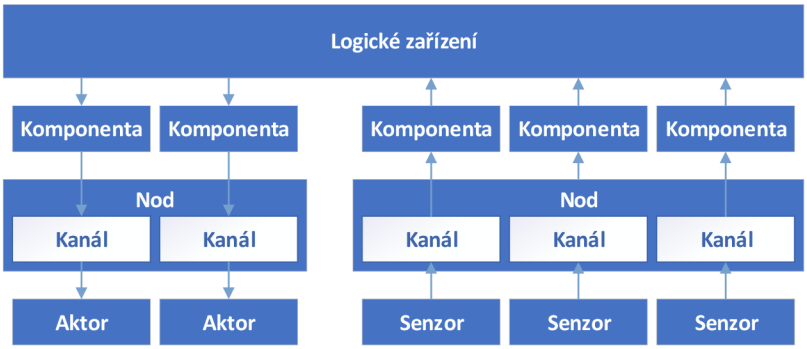
Kanál 3: a

Kanál 4: $\dfrac{a}{1\,024} \cdot 5\,000$

Kanál 5: $\dfrac{0{,}004\,887\,5 \cdot a - 0{,}86}{0{,}03}$

Kanál 6: $\operatorname{rint} \dfrac{a+1}{64}.$

Tento specifický nod se tedy mapuje na šest různých komponent. Data z akcelerometru se nepřepočítávají a uchovávají se jako surová pro další zpracování. Data z teploměru a vlhkoměru se z čísla reprezentujícího úroveň napětí přepočítávají na fyzikální veličiny a hodnoty z potenciometru jsou lineárně převedeny z rozsahu 0–1 023 na rozsah 0–15 pro hrubší krokování a eliminaci elektrického šumu.



Obrázek 4.5: Vztah mezi kanálem nodu a komponenty [autor]

Matematický přepočet lze provádět i opačným směrem – z komponenty na nod s přístupem WRITE a připojenými aktory. V tomto případě je proměnná a brána jako požadovaný stav komponenty a výsledek jako nový stav kanálu nodu. Například nod vybavený firmwarem a vhodným plošným spojem pro buzení RGBW pásku vyžaduje čtyři složky, jednu pro každou barvu. K regulaci svitu LED pásku se nejčastěji využívá pulzně šířková modulace (PWM). Dle vybraného mikrokontroléru se liší rozlišení modulace a může být realizována jako osmibitový modul, tedy s možností regulace v rozsahu 0 až 255. Intenzita barevných složek je ale pro uživatele lépe reprezentovatelná jako procentuální v rozsahu 0–100 %. Lze tedy aplikovat výraz `floor(`$a \cdot 2{,}55$`)`, který procentuální číslo vynásobí konstantou a výsledek zaokrouhlí na nejbližší celé číslo. I nody typu WRITE lze nastavovat surovými daty, stačí definovat vzorec pouze jako a. Směr přepočtu komponenty na nod nebo naopak aplikace pozná dle typu nodu. Samotné komponenty a jejich vztahy pak musí poskytnout server, neboť díky implementované architektuře nemá subsystém prostředky pro zjištění konkrétních zařízeních na svorkách nodu. Je tedy zcela abstraktní vůči datům. Vztah mezi kanálem nodu a komponenty je graficky znázorněn na obrázku 4.5.

Další možnosti komponent

V diagramu 4.2 je vidět, že každá komponenta musí mít nastaven výčet `ComponentType`, který specifikuje původ komponenty. Všechny předchozí části textu se zabíoraly výhradně komponentou s typem `Native`, které se v subsystému mapují na nody. Komponenta s typem `External` slouží pro implementaci podpory externích datových zdrojů. Komponenta s takovým typem je pro subsystém zcela neviditelná a existuje pouze v interní architektuře IoTIM, kde je perzistována. Typ `Edge` je přítomen v subsystému, ovšem není mapován na nody a slouží pro reprezentaci výsledků modulů pro hraniční výpočty (Edge Computing).

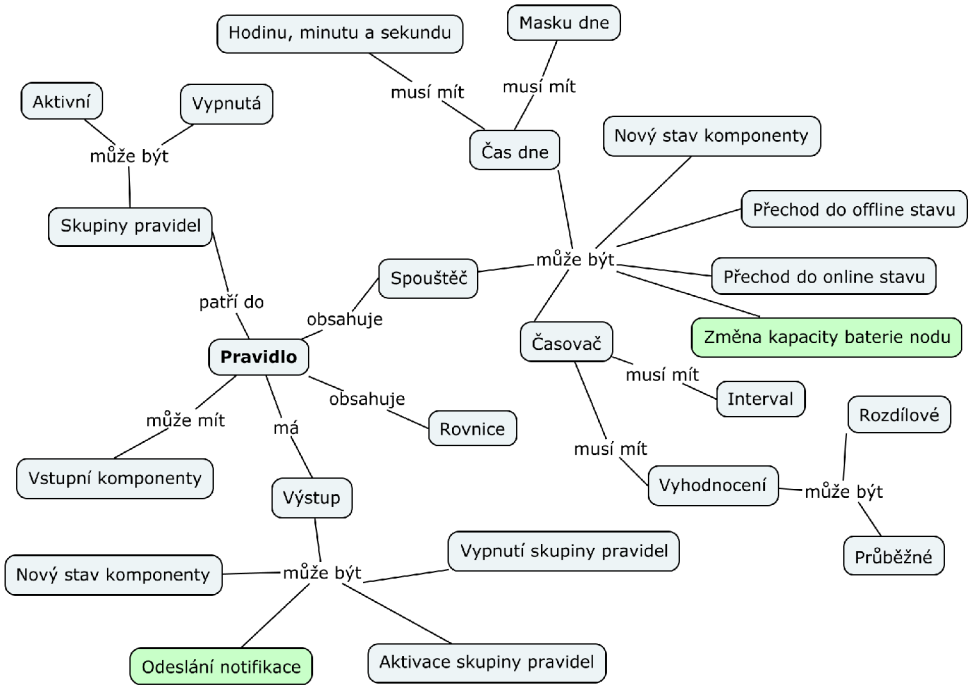
4.2.2 Pravidla

Posledním datovým objektem je `Rule`, neboli pravidlo. Pravidla se v subsystému využívají pouze v situaci, kdy nelze navázat komunikaci s IoTIM a jejich musí obsahovat informace potřebné pro vykonávání reakce na základě nějaké akce. Na IoTIM se využívají neustále a je pouze na uživateli, jaká pravidla nastaví subsystému a jaká IoTIM. Struktura pravidla je znázorněna na obrázku 4.6. Vlastnosti pravidla znázorněná zelenou barvou se týkají pouze IoTIM.

Pravidlo musí mít definovaný spouštěč a výstup. Spouštěčem může být stav vstupní komponenty, časovač, čas dne, přechod subsystému do off-line stavu, přechod subsystému do on-line stavu nebo změna kapacity baterie nodu. Výstupem pak nastavení 1 až N komponent, aktivace skupiny pravidel či vypnutí skupiny pravidel. Pokud jsou spouštěčem komponenty, je nutné uvést o které se jedná. Jednotlivé složky komponent pak figurují ve výstupních výrazech. V případě spouštění časovačem je nutné specifikovat jeho interval. Interval vyhodnocování časových pravidel bude závislý na konkrétní implementaci platformy. U časovače lze zvolit i způsob vyhodnocení. Pokud je nastaven na rozdílové, je výstupní akce provedena pouze pokud je výsledek odlišný od předchozího výpočtu. Průběžné vyhodnocení pak provádí výslednou akci pokaždé, nezávisle na výsledku operace. Pokud je spouštěčem čas dne, je nutné specifikovat hodinu, minutu a sekundu dne, kdy se má pravidlo provést. Lze také aplikovat masku, kterou lze vybrat pouze některé dny v týdnu.

Výstupem pravidla může být buď změna stavu komponenty pro zápis, aktivace či vypnutí skupiny pravidel nebo notifikace. Pokud má být výsledkem zápis komponent, je opět nutné uvést o jaké komponenty

se jedná. U pravidla v subsystému pak nezáleží, jestli se jedná o nativní komponentu pro zápis nebo vstupní komponentu edge modulu. Ke každé složce komponenty je nutné uvést výraz. Pokud je pravidlo relevantní pouze po některé složky, je možné nechat výraz prázdný. Když je výstupem aktivace nebo vypnutí skupiny pravidel, je nutné uvést pouze jeden výraz. Aby se provedla požadovaná akce, musí být výraz roven jedné.



Obrázek 4.6: Struktura pravidla [autor]

Ukázky pravidel

Oblast domácí automatizace je součástí problematiky IoT a může sloužit jako ukázka jednoduchého pravidla. Pravidlem, které se v instalaci inteligentního domu často uplatňuje, je prosté rozsvícení světla vypínačem. Mějme první nod s logickou adresou 1 a jedním kanálem, kde je připojen vypínač a druhý nod s adresou 2, kde je připojeno výkonové relé spínající světlo, také s jedním kanálem. Na nody se budou přímo mapovat komponenty s identifikátory například 100 a 200. Spouštěčem pravidla bude změna komponenty s identifikátorem 100, výstupem bude zápis do komponenty s identifikátorem 200. Protože komponenta s ID 200 má pouze jednu složku, je nutné uvést pouze jeden výraz, který bude jednoduchý, protože vypínač je binární výstup 0 nebo 1 a relé je binární vstup 0 nebo 1. Výraz bude mít následující podobu:

`c200_0 = c100_0`,

kde písmeno `c` značí prefix komponenty, 100 je její identifikátor a druhé číslo oddělené podtržítkem je nultá složka komponenty.

Druhým příkladem může být řízení zalévání zahrady. Mějme dvě pravidla patřící do stejné skupiny. První pravidlo každý den v 19 hodin sepne čerpadlo ženoucí vodu do zahradního zavlažovače. Druhé pravidlo pak čerpadlo každý den v 19 hodin a 15 minut vypne. Tento scénář lze zefektivnit přidáním tří dalších pravidel. Všechny budou využívat komponentu mapovanou na nod s dešťovým senzorem. Třetí pravidlo vypne skupinu pravidel pro zalévání, pokud je na senzoru zjištěna voda, tedy prší a není třeba

zahradu zalévat. Čtvrté pravidlo vypne při dešti samotné čerpadlo a páté pravidlo se vyhodnotí, pokud je senzor již suchý a skupinu pravidel pro zalévání aktivuje zpět.

Poslední příklad spadá do kategorie zabezpečení. Každý pátek v 18 hodin se spustí pravidlo aktivující skupinu pravidel s ID 1 a každé pondělí v 6 hodin ráno pravidlo, které skupinu deaktivuje. Ve skupině je přítomné pouze jedno pravidlo, které permanentně aktivuje sirénu při sepnutí pohybového čidla. Sirénu lze vypnout pouze druhým pravidlem, které je aktivováno skrytým tlačítkem.

4.3 Návrh IoTIM

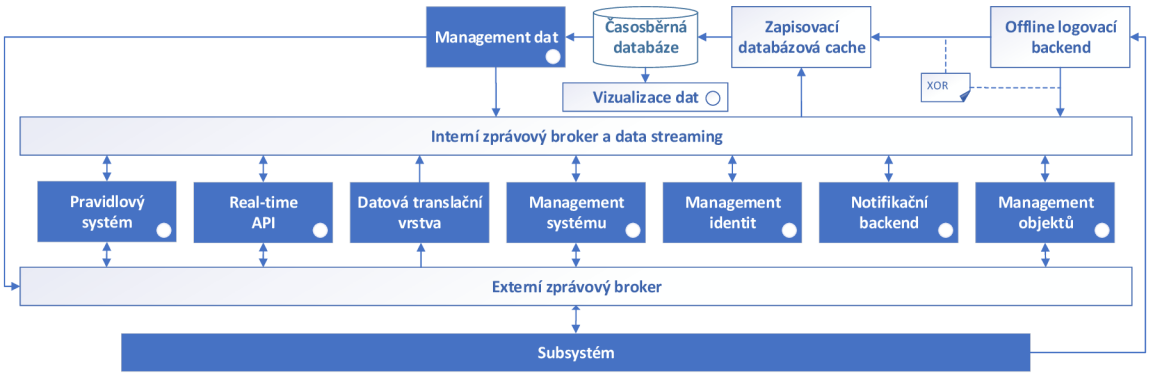
Obrázek 4.7: Digram případů užití IoTIM [autor]

Cílem aplikačního middlewaru je přijímat data čtecích komponent subsystémů a perzistovat je v databázi a poskytovat přes aplikační rozhraní jejich historický vývoj, případně nové stavy v reálném čase. Opačným směrem mohou uživatelé v podobě klientských aplikací nastavovat zapisovací komponenty, které se skrze subsystém mapují na nody a ty pak nastavují aktory. Základní úkony musí být možné automatizovat pomocí pravidlového systému. Dále musí IoTIM vystavovat přes API veškeré prostředky pro management uživatelů, komponent, nodů, pravidel, subsystémů a výpočetních modulů [83]. Aktéři systému a úkony jsou znázorněny v diagramu 4.7, kde lze identifikovat tři role aktérů: klient, systém a

subsystém. Klientem jsou myšleny uživatelské aplikace reagující se systémem autonomně nebo na základě pokynů uživatelů. Mezi případy užití klienta pro provedení autorizace v systému patří možnost zobrazit si dashboard, tedy přehledovou stránku s aktuálním stavem systému, grafy či tabulkami, explicitním nastavením stavu komponenty pro zápis, žádost o otevření proudu dat v reálném čase, přehled nad stavem subsystémů a CRUD operace nad objekty v IoTIM a aktualizací samotného IoTIM. Rozšířením CRUD operací nad nody je pak možnost detekovat nové nody připojené k systému a jejich identifikace, například skrze stavovou LED, pokud to hardwarové vybavení nodu umožňuje. Druhým aktérem je subsystém, který do systému poskytuje nové stavy komponent pro čtení, změny úrovně nabití baterie bezdrátových nodů, informační a chybové stavy subsystému a jeho nodů a události, které nastaly během off-line stavu subsystému. Posledním aktérem je systém samotný. Ten musí vykonávat pravidla na základě stavů vstupních komponent nebo času, detekovat nově připojené subsystémy, dostupnost těch existujících a přiřazovaní adres novým nodům připojených k subsystému.

4.3.1 Kompozice mikroslužeb

Zátěž IoTIM se může odlišovat dle nasazení. V prostředí domácí automatizace lze očekávat nízký datový tok od subsystému, jelikož není nutné získávat senzorická data frekventovaně. Stejně tak se počet zapisovacích komponent s aktory bude pohybovat maximálně v řádu desítek. Požadavky na prostředky tedy nebudou vysoké a výkon běžného stolního počítače bude dostatečný. Naopak při realizaci senzorického experimentu s MEMS senzory lze očekávat od každého přítomného subsystému i několik stovek měření za sekundu. V tomto případě bude nejvíce zatěžována perzistetní vrstva. Aby bylo možné realizovat tyto odlišné scénáře, je vhodné IoTIM realizovat jako sadu vzájemně komunikujících mikroslužeb. Výhodou tohoto přístupu je možnost škálování prostředků dle vytížení jednotlivých části systému. V diagramu 4.8 jsou zobrazeny jednotlivé služby a jejich propojení. V případě bloků znázorněných plnou barvou se jedná o mikroslužby, které je nezbytné vyvinout. U stínovaných bloků lze využít již existující technologie a pouze je vhodně nakonfigurovat. Aplikace se zakresleným kolečkem vystavují dále aplikační rozhraní pro klientské aplikace. Na tyto technologie nebude nahlíženo jako na mikroslužby. Nyní budou popsány zodpovědnosti jednotlivých technologií a mikroslužeb.



Obrázek 4.8: Architektura mikroslužeb IoT middleware [autor]

Externí zprávový broker

Externí zprávový broker poskytuje subsystémům prostředky pro spojení se službami a deleguje zprávy od subsystémů k IoTIM a obráceně.

Interní zprávový broker a data streaming
Pokud služby vyžadují interakci mezi sebou, využívají interní broker, který zajistí doručení zprávy. Dle vybrané implementace může také sloužit jako perzistentní úložiště, které veškerou komunikaci loguje, pokud není přítomen odběratel zprávy.

Offline logovací back-end
Subsystémy se spojují s logovacím back-endem při znovunavázání spojení s IoTIM. Ten od nich postupně kopíruje log všech událostí, které nastaly během nedostupnosti IoTIM.

Zapisovací databázová cache
Tato služba nemusí být dle zvolených technologií pro implementaci přítomna. Některé časosběrné databáze takové služby poskytují k efektivnějšímu zápisu dat do databáze a k balancování zátěže. Pokud je databáze zaneprázdněna sestavováním odpovědi na historická data, slouží cache jako dočasná paměť a nová data jsou do databáze vložena až při poklesu zatížení.

Časosběrná databáze
Časosběrná databáze perzistuje pouze data komponent, a to jak mapovaných na nody (nativní), tak i externích a hraničních. Pokud vyžaduje nějaká služba perzistentní úložiště, je vždy realizováno v rámci služby. Dle požadavků jednotlivých aplikací se může jednat o prostý konfigurační soubor, souborovou databázi nebo plnohodnotnou relační či dokumentovou databázi.

Vizualizace dat
Služba poskytuje grafické rozhraní pro vizualizaci dat komponent v grafech a tabulkách. Tento modul může být dle konkrétní implementace realizován i jako klientská aplikace, komunikující s mikroslužbou managementu dat.

Datová translační vrstva
Tato služba provádí překlad z datového formátu využívaného mezi IoTIM a subsystémy do formátu vyžadovaného zapisovací databázovou cache.

Management dat
Cílem této služby je vystavovat API klientským aplikacím. Ty jej využívají k dotazovaní na aktuální či historická data komponent a nastavování stavů komponent mapovaných jak na fyzické nody, tak i na komponenty virtuální pocházející z externích zdrojů.

Management systému
Tato služba detekuje stav jednotlivých subsystémů, umožňuje vzdálenou aktualizaci softwaru subsystémů a detekuje nově přidané subsystémy. Pokud je do systému přidán již nakonfigurovaný subsystém, služba zjišťuje jeho nody, komponenty a pravidla.

Management objektů
Cílem této služby je v subsystémech detekovat nové nody, dle požadavků uživatele mapovat komponenty a vytvářet edge computing moduly v subsystému.

Management identit
Služba má na starosti správu uživatelů a uživatelských oprávnění v rámci jednotlivých dalších služeb. Pokud uživatel vyžaduje provedení akce nějaké další služby, je nejprve přes tuto službu zjištěno, zdali má potřebná oprávnění.

Pravidlový systém

Pravidlový systém (Rule engine z anglického jazyka) vystavuje API klientským aplikacím umožňující CRUD operace nad pravidly a skupinami pravidel. Přes externí broker přijímá v reálném čase vyčtené komponenty subsystémů a od služby managementu dat komponenty k zápisu a komponenty virtuální. Na základě pravidel může nastavit komponentu, změnit stav skupiny pravidel nebo vyslat notifikačnímu back-endu žádost o provedení definované akce.

Real-time API

Pokud klientská aplikace vyžaduje data komponent v reálném čase, využije této služby k žádosti o otevření datového toku.

Notifikační back-end

Služba vystavuje API pro tvorbu notifikačních událostí, které se mohou provést při splnění určitého pravidla. Implementované technologie se mohou lišit dle implementace a může se jednat například o zasílání e-mailu, SMS, podporu chatovací služby Telegram, Twitter atd.

4.3.2 Možná oprávnění klientských aplikací

Tabulka 4.1: Oprávnění klientských aplikací rozděleno dle zodpovědných služeb [autor]

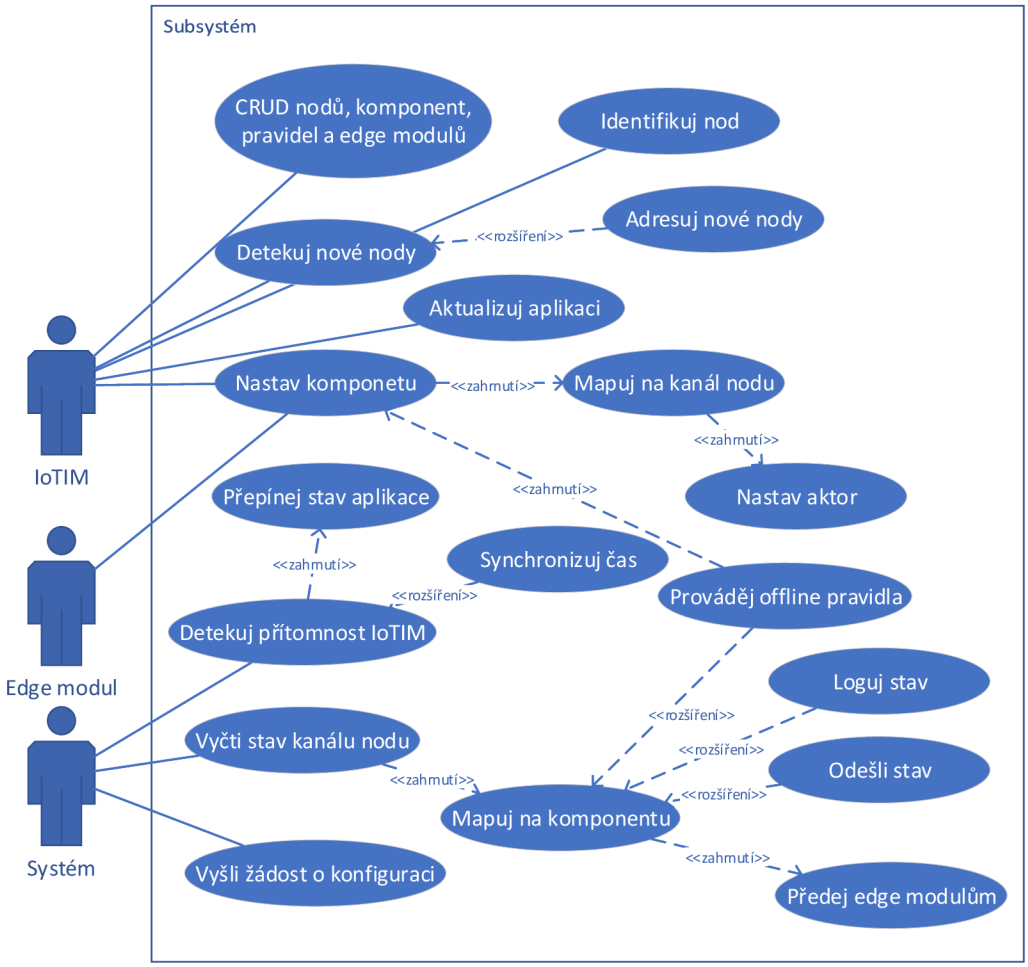
Mikroslužba	**Oprávnění uživatele**
Management identit	Vytváření nových uživatelů CRUD existujících uživatelů CRUD oprávnění existujících uživatelů
Management systému	Začlenění subsystému Importování objektů existujícího subsystému
Management objektů	Detekce nebo začlenění nodů Identifikace nodů Mapování komponent CRUD výpočetních modulů
Management dat	Obdržení dat specifické komponenty Obdržení dat jakékoliv komponenty Nastavení stavu specifické komponenty Nastavení stavu jakékoliv komponenty
Pravidlový systém	CRUD pravidel
Notifikační backend	CRUD notifikační reakce vlastních pravidel CRUD notifikační reakce jakéhokoliv pravidla
Real-time API	Otevření real-time spojení Obdržení dat specifické komponenty Obdržení dat jakékoliv komponenty Nastavení stavu specifické komponenty Nastavení stavu jakékoliv komponenty
Vizualizace dat	Přístup k vizualizaci Vytváření nových vizualizačních pohledů

Bezpečnost je při realizaci IoT middlewaru jedna ze stěžejních vlastností [23]. Na základě digramu užití aplikace 4.7 byly stanoveny požadavky na oprávnění klientských aplikací. V tabulce 4.1 jsou oprávnění vypsána ve vztahu k relevantní službě. Oprávnění, která povolují kompletní manipulaci s

daným objektem, jsou nadřazena oprávněním jednotlivých instancí. Realizace autentifikace a autorizace klientských aplikací je pak závislá na konkrétní implementaci architektury IoTIM.

4.4 Návrh subsystému

Subsystém nahrazuje bránu rozšířením o možnost logování, off-line pravidel a synchronizací času s IoTIM a výpočetních modulů. Mezi činnosti, které musí aplikace subsystému provádět, patří mapování komponent na nody a zpět, detekce výpadku spojení, komunikaci s nody na vybraných sběrnicích a management objektů dle povelů IoTIM. Diagram případů užití je znázorněn na obrázku 4.9 a opět lze identifikovat tři typy aktérů. IoTIM, se kterým je subsystém spojen vybranou komunikační technologií, nastavuje nové stavy komponent pro zápis, spouští proces detekce nových nodů na všech nebo vybraných sběrnicích a technologiích, provádí CRUD operace nad nody, komponentami, pravidly a edge moduly a v poslední řadě provádí pokyn k aktualizaci aplikace. Běžící edge moduly přijímají od systémového aktéra nové stavy komponent, provádějí výpočty a na základě jejich výsledků nastavují komponenty stejným způsobem jako IoTIM. Systém dále neustále detekuje přítomnost IoTIM aplikace a v případě nedostupnosti přepíná celou aplikace do off-line stavu. Periodicky kontroluje nebo asynchronně získává nové stavy nodů, které mapuje na komponenty a stavy odesílá IoTIM nebo interně loguje. Pokud je subsystém nový bez přidělené konfigurace, tak se snaží získat od IoTIM nezbytné konfigurační parametry.

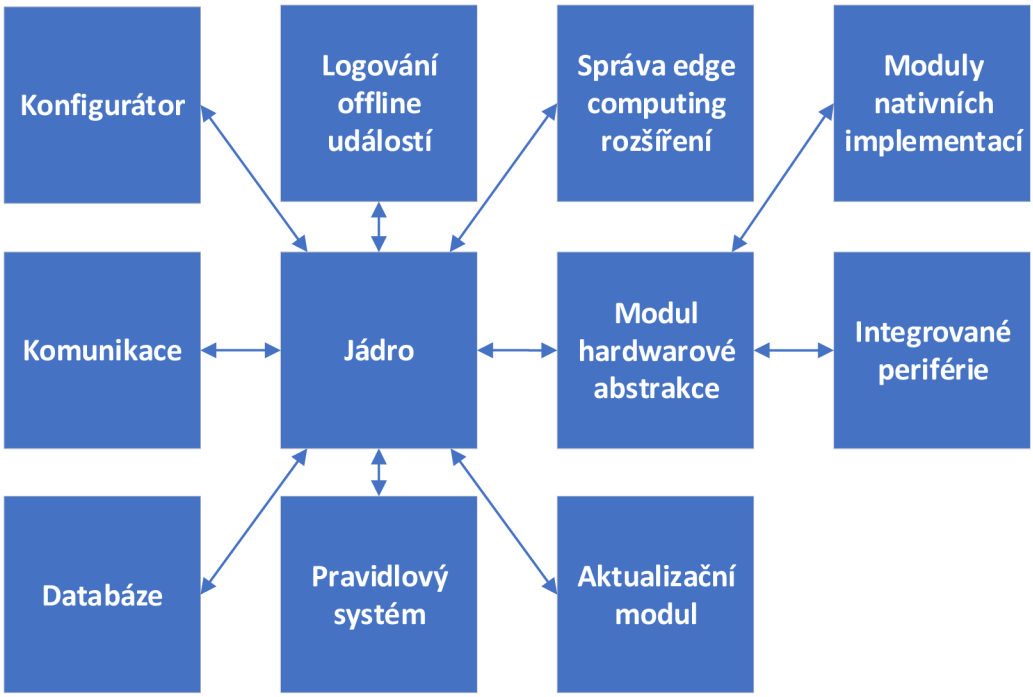


Obrázek 4.9: Digram případů užití subsystému [autor]

4.4.1 Bloková architektura aplikace

Existující brány využívají téměř výhradně jednodeskové embedded počítače s operačním systémem Linux a procesory architektury ARM, které jsou energeticky úsporné. I když výkon těchto počítačů vzrůstá, stále nedosahují výkonu architektury AMD64. Jelikož subsystém vykonává omezenější škálu činností než IoTIM, není nutné řešení implementovat jako sadu mikroslužeb. Monolitická aplikace poskytne dostatečné možnosti škálování, při využití paralelismu a asynchronních přístupů.

V diagramu 4.10 jsou znázorněny jednotlivé bloky aplikace nutné ke splnění případů užití.



Obrázek 4.10: Architektura bloků aplikace subsystému [autor]

Jádro

Jádro je hlavním prvkem aplikace. Orchestruje komunikaci všech ostatních bloků a obsahuje hlavní logiku programu. Kontroluje dostupnost IoTIM a základě toho přepíná subsystém do stavu on-line nebo off-line.

Komunikace

Komunikační blok navazuje spojení v IoTIM a provádí překlad ze zvoleného datového formátu na třídy či struktury (dle vybraného implementačního jazyka) a zpět.

Databáze

Subsystém musí mít perzistentě uloženy nody, komponenty a pravidla. Nejjednodušším způsobem je prostý konfigurační soubor, který ovšem s narůstajícím počtem entit bude hůře spravovatelný. Jako vhodné úložiště se pro subsystém jeví souborová databáze. Ta není tolik náročná na prostředky jako databáze s architekturou klient-server, ovšem oproti souboru poskytuje výhody v podobě transakcí.

Konfigurátor

Zatímco objekty je vhodné mít uloženy v databázi, tak běžné konfigurační parametry, jako IP adresa externího zprávového brokera, identifikátor subsystému a konfigurace nativního hardware, je pro účely

ladění a migrace vhodnější načítat z konfiguračního souboru. Další parametry se mohou lišit dle specifické implementace aplikace.

Modul hardwarové abstrakce
Protože různé komunikační technologie využívají odlišné přístupy a knihovny, slouží tento modul k abstrakci jednotlivých implementací. Jádro vidí vše jako nody, na které se mapují komponenty. Pokud pravidlový systém, edge modul nebo IoTIM vyžaduje nastavení nodu skrze komponentu, předává jádro nový stav přes tento modul, který jej deleguje konkrétní implementaci.

Moduly nativních implementací
Těchto modulů může v aplikaci existovat 1 až N a jedná se o implementace jednotlivých komunikačních technologií pro nody, jako jsou například CAN, KNX, ZigBee a Sigfox.

Integrované periférie
Hardware existujících bran je často realizován s využitím jednodeskových embedded počítačů jako Raspberry Pi či BeagleBone nebo TinkerBoard. Ty obsahují množství GPIO pinů a sběrnic jako I2C, SPI či 1-Wire. Lze tedy již přímo na subsystém integrovat senzory a aktory. Aby se integrovaný hardware choval identicky jako nody připojené externě, jsou periferie obsluhovány v modulu implementujícím stejné rozhraní jako nativní implementace. Pro jádro jsou pak tyto periferie stejné jako ostatní nody.

Pravidlový systém
Pravidlový systém subsystému musí implementovat aplikační logiku dle diagramu 4.6, tedy reagovat nové stavy komponent okamžitě, nebo spouštět pravidla s určitým intervalem, reagovat na změnu stavu aplikace, měnit stavy skupin pravidel atd.

Logování off-line událostí
Pokud se subsystém nachází v off-line režimu, veškeré události (nové stavy komponent, vyhodnocení pravidel, změny stavu baterií nodů, informační a chybové zprávy) jsou směrovány na tento modul, který zajití jejich perzistentní uložení. Po přechodu subsystému do on-line stavu se data přesouvají do časosběrné databáze přes off-line logovací backend na IoTIM.

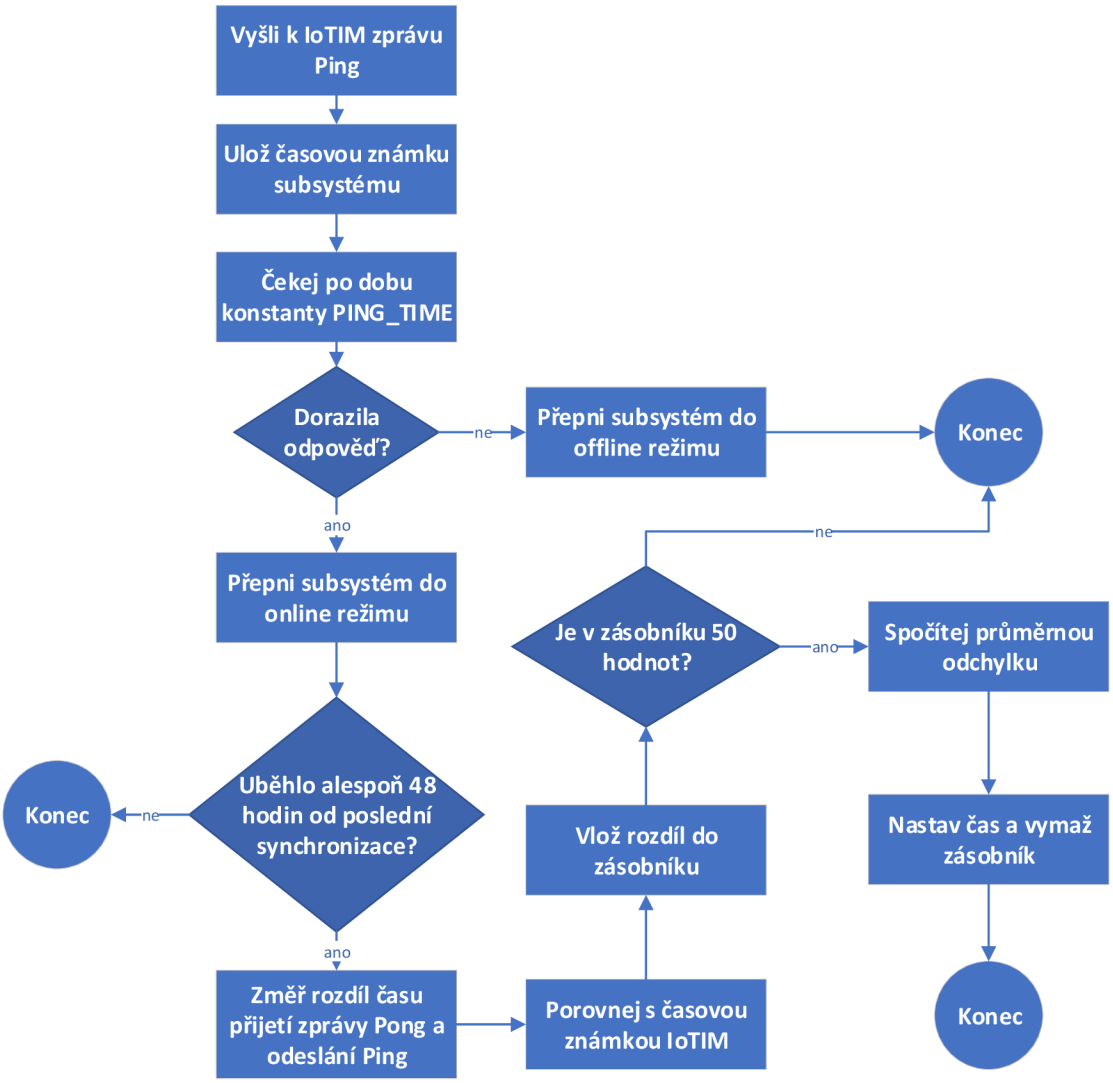
Správa edge computing rozšíření
Tento modul přijímá stavy nativních komponent a poskytuje je výpočetním modulům skrze lokální komunikační technologii. Moduly aplikují různé výpočty a odesílají modulu zpět výsledky, které modul předává jádru jako `Edge` komponenty. Ty jsou v on-line stavu odeslány na IoTIM stejným způsobem jako nativní komponenty, případně v off-line stavu logovány a předány pravidlovému systému.

Aktualizační modul
Aktualizační modul provádí na pokyn IoTIM stažení nové verze subsystému a potřebných knihoven či konfiguračních souborů. Kontroluje integritu a architekturu souborů a pokud je vše v pořádku, oznamuje IoTIM možnost aktualizace. Modul musí korektně ukončit běžící proces a vytvořit proces nový s novou verzí aplikace.

4.4.2 Mechanismus detekce IoTIM a synchronizace času

Jedním z požadavků na nové řešení je zabudovaná podpora synchronizace času mezi branami (zde označovány jako subsystémy) a integračním middlewarem. Na vývojovém diagramu 4.11 je znázorněn princip, kterým lze docílit synchronizaci času a zároveň detekovat přítomnost IoTIM a přepínat stav

subsystému mezi on-line a off-line. Jedná se o zjednodušenou variantu přístupu navrženého v článku [72]. V subsystému je přítomen proces, který periodicky odesílá k IoTIM zprávu a ukládá si časovou známku odeslání. Poté po dobu definovanou konstantou čeká na odpověď. Pokud z jakéhokoliv důvodu nedorazí, subsystém nastavuje aplikační logiku do off-line stavu a pracuje s lokálními nody a výpočetními moduly autonomně. V případě že odpověď dorazí, subsystém přepíná logiku do on-line stavu a kontroluje, kdy došlo dle jeho interního času naposledy k synchronizaci hodin. Pokud ne, je měřen rozdíl času subsystému mezi odesláním a přijetím zpráv, tedy latence sítě. Následně je porovnána časová známka IoTIM a subsystému a k této diferenci je buď přičtena nebo odečtena polovina rozdílu času mezi odesláním a přijetím, podle toho jestli je čas subsystému pozadu nebo napřed. Toto číslo je vloženo do zásobníku a po získání 50 měření je spočítána průměrná odchylka a nastaven systémový čas.



Obrázek 4.11: Vývojový diagram detekce IoTIM a synchronizace času [autor]

4.4.3 Zprávy pro komunikaci s IoTIM

Subsystémy a IoTIM musejí mezi sebou komunikovat. Komunikaci lze rozdělit na běžný stav, kdy subsystém sděluje nové stavy komponent, přijímá nové stavy komponent a informuje IoTIM. Do druhé kategorie patří zprávy pro management objektů subsystému a jeho samotného. Ty probíhají nejfrekventovaněji během prvotní konfigurace subsystému. Poslední kategorií jsou zprávy pro aktualizaci aplikace subsystému na novější verzi. Samotné technologie pak závisí na konkrétní implementaci architektury. Lze využít několik formátů zpráv, jak textových tak binárních, i komunikačních technologií s různými vzory komunikace (request-reply nebo publish-subscribe), viz vybrané technologie představené v kapitole 3.5.3. Konkretní implementace může obsahovat i dodatečné parametry zpráv vyžadované samotnými technologiemi.

Zprávy v běžném stavu

Data
Nejčastejší zpráva je taková, která obsahuje informace o nový datech komponenty. Musí být přítomna alespoň adresa nodu, kanál, list s daty a časová známka s co nejvyšší granularitou. Zprávu odesílají obě vrstvy.

Error
Tato zpráva slouží k upozornění IoTIM, že došlo v aplikaci subsystému k neočekávanému problému. Obsahuje enumeraci s identifikátorem chyby a volitelné informace, které se mohou vztahovat k identifikátoru objektů či textovému popisu chyby. Mezi chybové stavu může patřit například špatná inicializace hardwaru, chyba v matematických přepočtech pravidel, rozpadnutí signálového spojení bezdrátových nodů atd.

Info
Podobně jako `Error`, `Info` slouží k informování IoTIM o průběhu nějaké události. Mezi informační události může patřit například znovu připojení bezdrátového nodu, synchronizace systémových hodin či provedení požadované CRUD operace nad objekty. Dle implementace lze zpráv sloučit se zprávou `Error` a utvořit tak univerzálnější událost `Event`. Naopak lze také frekventovanější zprávy implementovat odloučeně, například změnu kvality signálového spojení mezi bezdrátovým nodem a subsystémem či změnu nabití baterie nodu.

PingPong
Tato zpráva hraje velmi důležitou roli pro detekci přítomnosti IoTIM a přechodů stavů aplikace. Mikroslužba managementu systémů po jejím přijetí doplní svou časovou známku a odesílá jí zpět. Subsystém pak zprávu využívá ke zjišťování odchylky systémového času a času IoTIM.

Zprávy pro CRUD operace a management aplikace

Component
Zpráva pro manipulaci s komponentou. Obsahuje unikátní identifikátor komponenty, deskriptor operace, převodní výrazy a volitelná metadata.

EdgeComputer
Zpráva obsahuje nezbytné informace pro přidání výpočetního modulu do subsystému. Obsahuje identifikátor a popis modulu, název spustitelného souboru, binární balík výpočetního modulu, seznam nativních

komponent které jsou vyžadovány pro výpočty a seznam edge komponent, které jsou vystavovány jako výpočetní výsledky modulu.

Identify
Tato zpráva umožňuje sepnout nebo vypnout blikání statusové LED na nodech. Je nutné uvést adresu nodu a vyžadovaný stav. Tato funkcionalita je volitelná a nemusí být možné ji implementovat při zahrnutí existujících zařízeních.

Node
Zpráva pro manipulaci s nodem. Obsahuje jeho adresu, počet kanálů, typ připojení a volitelné argumenty dle konkrétní implementace. Zprávu lze také řetězit pro hromadnou operaci s několika nody najednou.

Rule
Veškeré CRUD operace s pravidly se provádějí touto zprávou. Musí být uvedeny všechny identifikátory, spouštěče, vstupy a výstupy.

RuleGroup
Zpráva určená pro vynucení aktivace nebo deaktivace skupiny pravidel nebo permanentnímu odstranění skupiny pravidel. Musí obsahovat identifikátor skupiny a stav aktivity.

ScanRequest
IoTIM touto zprávou iniciuje skenování komunikačních technologií subsystému k detekci nových nodů. Musí být specifikovány konkrétními sběrnice či technologie.

SitrepRequest
IoTIM vyžaduje od subsystému kompletní informace o jeho hardware, konfiguraci a datových objektech. Přítomnost této zprávy zjednoduší migraci subsystému pod jinou instanci IoTIM nebo pro bezpečnostní audit.

SitrepResponse
Subsystém odpovídá na zprávu `SitrepRequest` reportem.

Zprávy pro aktualizaci aplikace

Aplikace obsahuje zabudované prostředky pro její aktualizaci. Následující zprávy jsou využity k procesu aktualizace.

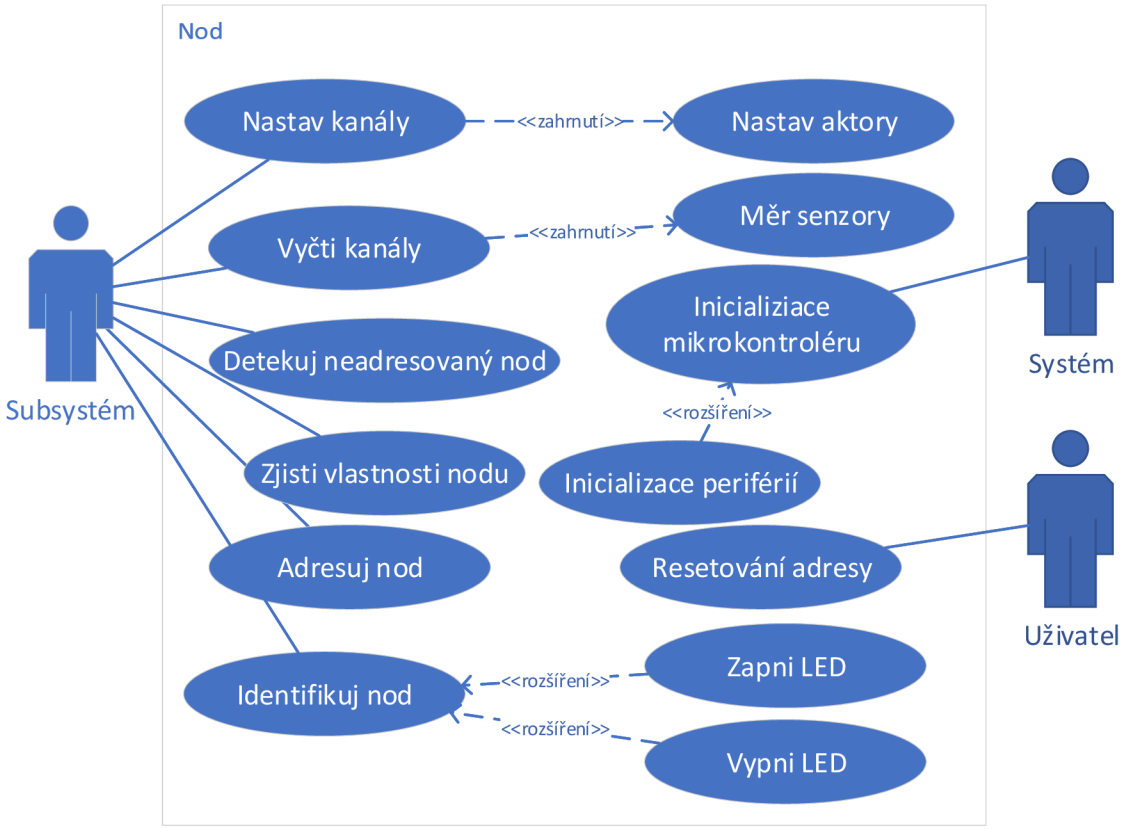
UpdateAnnounce
IoTIM oznamuje subsystému dostupnost nové verze. Ve zprávě je obsažena adresa URL adresa nové verze, kontrolní součet a identifikátor.

UpdateConfirmation
Subsystém odesílá IoTIM svůj stav a připravenost k aktualizaci. Stav je buď úspěšný nebo chybový s možností specifikace konkrétního chybového stavu.

UpdateStart
IoTIM po úspěšné verifikace nové verze aplikace spouští proces aktualizace.

4.5 Návrh nodů

Nody jsou nejnižší vrstva platformy a lze na ně nahlížet z hlediska softwaru a hardwaru. Softwarově se jedná o firmware běžící v mikrokontroléru nebo mikroprocesoru, který slouží jako translační vrstva mezi rozhraním senzoru a subsystémem. Hardwarově se jedná o plošný spoj obsahující součástky pro sběrnice a technologie kompatibilní s konkrétní implementací subsystému a vhodné vybavení pro komunikaci s požadovanými senzory nebo aktuátory. V diagramu užití 4.12 jsou uvedeny aktéři a činnosti, které musí nod umožňovat. Tento případ užití je pro případ vývoje nového hardwaru a firmwaru, existující zařízení s různými technologiemi lze programově transformovat na nody v aplikaci subsystému a vystavovat je vyšším vrstvám jako nativní komponenty.



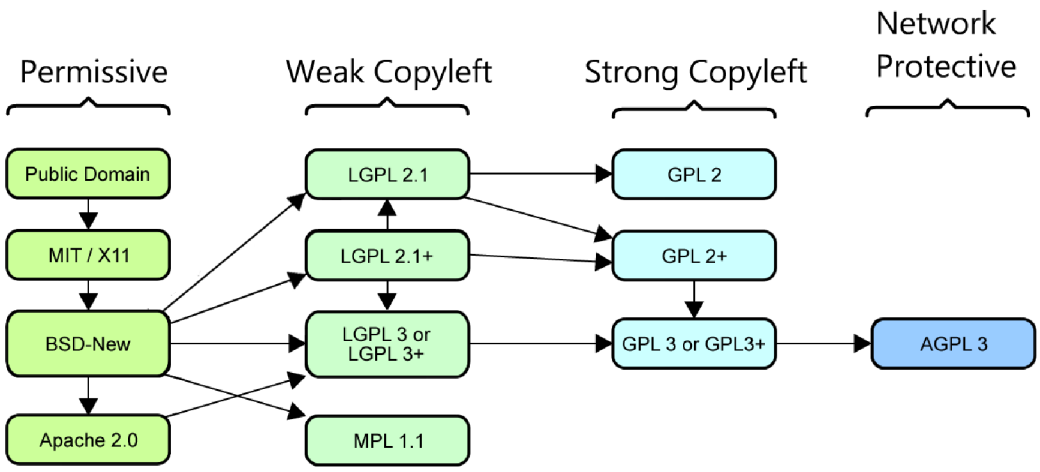
Obrázek 4.12: Diagram případů užití nodu [autor]

Aktor subsystém může v případě nodu s aktorem vyžadovat nastavení kanálů nodu, na což firmware nodu musí reagovat nastavením příslušných aktorů, jejichž stav je měněn. Naopak subsystém vyžaduje sdělení stavu senzorů, jedná-li se o nod se senzory. Proces komunikace je závislý na vybraných technologiích. Může být řízen subsystémem, jedná-li se o master-slave sběrnici, kdy subsystém řídí veškerou komunikaci. V případě úsporných bezdrátových technologií mohou nody odesílat změny stavu senzorů asynchronně, a to buď při jakékoliv změně, nebo při překročení předem dohodnutého prahového limitu. Subsystém také detekuje přítomnost nově připojených nodů, zjišťuje jejich vlastnosti definované v kapitole 4.2 a dle pokynů IoTIM jim přiděluje v rámci subsystému unikátní adresu. Pro základní diagnostiku obsahuje prostředky pro svojí indikaci, jako LED nebo akustický signál. Aktor systém provádí inicializaci mikro-

kontroléru (oscilátor, přerušení, watchdog atd.) a následně dle typu senzoru nebo aktuátoru inicializaci a konfiguraci ovladače specifické periférie, kterými vybraná komponenta komunikuje, například UART nebo SPI. Uživatel pak má možnost nod resetovat pro přesun k jinému subsystému, například fyzickým tlačítkem. I nody vybavené technologiemi pro komunikaci s IoTIM přímo by měly využívat subsystém pro zachování off-line funkcionality. Výjimkou budiž LoWPAN technologie SigFox, LoraWAN a NB-IOT, které komunikují s radiovými stanicemi.

4.6 Licence aplikací

Obrázek 4.13 ukazuje vztahy mezi jednotlivými populárními open-source licencemi. Pokud aplikace využívá pouze knihovny s liberální licencí MIT, je možné ji vydat pod stejnou licencí, případně si zvolit jakoukoliv návaznou permisivnější. Licenční ujednání GPL a LGPL pochází od Free Software Foundation, tvůrců kolekce softwarů GNU, nepostradatelnou částí operačního systému GNU/Linux. Licence udává, že každý kdo ve zdrojových kódech provede nějaké úpravy, je povinen všechny úpravy zveřejnit. Velmi častým omylem [19] je tvrzení, že komerční projekt nemůže využívat software s (L)GPL licencí. V licenčním ujednání není žádná klauzule, která by toto implikovala. Není povoleno pouze linkovat zdrojový kód softwaru s (L)GPL licencí. Tedy komerční software nemůže obsahovat kód šířený pod (L)GPL, ovšem využívat externí služby přes API dovoleno je.



Obrázek 4.13: Vztah populárních opensource licencí [142]

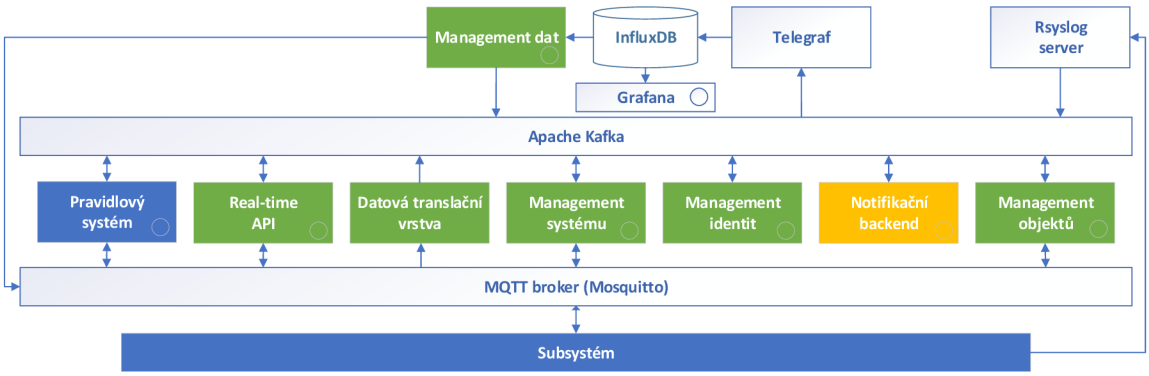
5

Ukázková implementace IoT middleware

V této kapitole jsou popsány vybrané aspekty implementace IoTIM vycházející z navržené architektury mikroslužeb. Nejprve jsou představeny konkrétní technologie vybrané k realizaci a následně popsány vybrané funkcionality vytvořené implementace.

5.1 Technologie využité pro implementaci

Diagram 5.1 je strukturou obdobný jako diagram návrhu architektury IoTIM 4.8, ovšem obsahuje technologie vybrané pro ukázkovou implementaci IoTIM. Bloky označené modře jsou realizovány v programovacím jazyce C++, zelené využívají Rust a notifikační back-end je vytvořen v jazyce Python.



Obrázek 5.1: Vybrané implementační technologie IoTIM [autor]

5.1.1 Zprávové brokery

Implementace využívá jako externí zprávový broker protokol MQTT, konkrétně minimalistickou implementaci Mosquitto, psanou v čistém ANSI C. Nepodporuje škálování ani clusterování, ovšem má minimální velikost, jednoduchou konfiguraci a dle testů provedených v [33, 103] výbornou latenci a propustnost. Obsahem samotných témat jsou serializované FlatBuffers zprávy, označeny postfixem FB, aby bylo možné využívat v mikroslužbách objekty se stejným logickým jménem. V C++ lze tento problém efektivně řešit využitím jmenných prostorů, kdy FlatBuffer zprávy mohou ležet ve jmenném prostoru `msg::` a objekty v prostoru např. `app::`. Řešení s postfixem bylo zvoleno jako multiplatformní

kompromis. Protože velké množství zpráv se týká pouze jednoho vybraného subsystému, je ve většině dále popsaných zpráv přítomna proměnná `uint subsystemId`, která obsahuje identifikátor daného subsystému.

Jako interní zprávový broker byla zvolena technologie Kafka [125] od organizace Apache. Kafka byla původně vyvinuta vývojáři profesní sociální sítě LinkedIn a později předána Apache a uvolněna jako open-source. V současnosti je Kafka využívána společnostmi jako Netflix, Uber, AirBnB, Spotify, Seznam, Pinterest či Slack. Technologii lze definovat jako distribuovanou streamovací platformu, kde se obdobně jako u MQTT vyskytují *témata*. Producenti zapisují data do témat logy a spotřebitelé je odebírají. Odlišnost obou technologií tkví v možnosti mít u Apache Kafka celý log zpráv perzistentní a to do doby, než je překročena retenční politika daného tématu. Doba se může odlišovat téma od tématu, a může být teoreticky nekonečná, resp. omezená pouze velikostí perzistentního úložiště. Dále je zahrnuta podpora replikace témat, podpora škálování dle aktuálního vytížení, podrobná konfigurace a externích rozšířeních Kafka Connect, které umožňuji integraci alternativních zdrojů dat do témat.

5.1.2 Databáze a logování offline událostí

Jako databáze pro perzistenci dat komponent byla zvolena časosběrná databáze InfluxDB [175]. Jedná se o NoSQL databázi s enginem implementovaným na míru časovým řadám. Data jsou organizována do jednotlivých měření (measurements) a dále identifikována pomocí tagů. Protože kombinace měření, tagu a časové známky vytváří unikátní klíč, vyžaduje Influx databáze nanosekundovou granularitu časových známek pro zamezení vzniku duplicit. V případě nativních komponent je měřením identifikátor subsystému a jako tag je využita adresa nodu a jeho kanál. Je-li na jednom kanálu více složek, jsou ukládány všechny do jednoho záznamu měření. V případě externích zdrojů jsou měření ukládána pod identifikátorem `external` a tagy jsou virtuální adresa a kanál nodu, který si definovala klientská aplikace skrze mikroslužbu managementu objektů. Systém Telegraf [195] je pak prostředníkem odebírající Kafka kanál a data vkládá do databáze. V případě, že je databáze zaneprázdněna odpovědí na komplexní dotaz mikroslužby Managementu dat, jsou data ukládána v Kafka kanálu.

Moderní distribuce operačního systému Linux využívají rsyslog [196] jako výchozí démon pro logování ostatních démonů. Ve výchozím stavu jsou logy ukládány do souboru `/var/log/syslog`, ovšem technologii lze flexibilně konfigurovat a různé aplikace logovat různým způsobem. Jednou z možných konfigurací je odesílat logy na vzdáleného rsyslog démona a pokud není možné navázat spojení, ukládat zprávy na lokální úložiště a odeslat je opožděně. Pokud nastane pouze výpadek MQTT brokera, jsou logy odesílány na IoTIM v reálném čase, pokud se jedná o výpadek celého spojení, jsou logy doručeny zpětně.

Na IoTIM je rsyslog nastaven tak, aby logy předával do tématu Kafka brokeru. Odběratelem tématu je Telegraf, tedy agregační systém pro databázi InfluxDB. Data jsou již ze subsystému formátována v nízkoúrovňovém textovém formátu, který Telegraf preferuje pro dosažení nejvyššího výkonu. Díky přítomnosti Kafka tématu jsou off-line zprávy logovány na disk pokud je Influx databáze zaneprázdněna náročným dotazem. Data jsou do databáze vkládána s původní časovou známkou subsystému.

5.1.3 Kontejnerizace aplikací a API pro klientské aplikace

Pro snadné nasazení na serveru byla zvolena kontejnerizační technologie Docker [17]. Každá mikroslužba a serverová aplikace běží ve vlastním kontejneru. IoTIM lze nasadit pomocí skriptu Docker Compose,

který vytvoří potřebné Docker kontejnery. Pro oddělení síťového provozu jsou vytvořeny tři ethernetové sítě. První slouží k přístupu klientských aplikací k jednotlivým mikroslužbám a vizualizačnímu prostředí Grafana, druhá k interní komunikaci skrze Apache Kafka a poslední slouží pro aplikace vyžadující MQTT broker a subsystémy. Mezi budoucí práce pak patří vytvoření Kubernetes clusteru s dostatečným počtem nodů.

Mikroslužby vystavují aplikační rozhraní pro klientské aplikace jako standardní REST a zprávami v JSON formátu. Protože každý kontejner má v klientské síti vlastní IP adresu a klientské aplikace by tak musely navazovat více TCP spojení na různé servery, je využit nginx jako reverzní proxy. Aplikace se tedy dotazují jediné IP adresy a dle URL je volána konkrétní mikroslužba. Jednotlivé služby jsou v nginx rozlišovány pomocí prefixů, kdy například aplikační dotaz na mikroslužbu pravidlového systému má URL prefix `rules/` a Real-time api `rt/`.

5.1.4 Implementace real-time dat

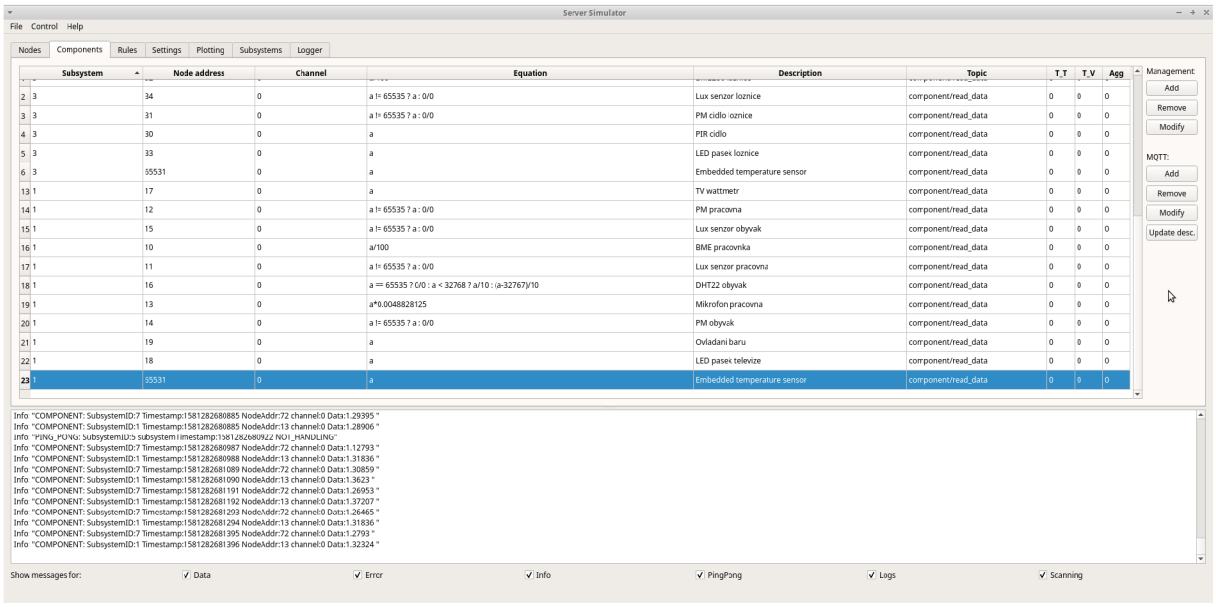
Jedním z klíčových požadavků na vlastnosti IoTIM je možnost poskytnout klientským aplikacím data co nejrychleji od vyčtení subsystémem bez nutnosti periodického dotazování. Za tuto funkcionalitu je zodpovědná samostatná mikroslužba, která si ve svém interním úložišti uchovává tabulku s identifikátorem klienta a seznamem dat komponent, které je klient autorizován odebírat. Mikroslužba vystavuje REST API a pro real-time data využívá WebSockets. Klientská aplikace vyžadující data si nejprve přes REST API vyžádá číslo TCP portu a seznam komponent, které nyní vyžaduje přijímat. Aplikace provede verifikaci oprávnění a pokud je v pořádku, odesílá vybraný port na kterém jsou aplikaci odesílána data ihned po přijetí přes MQTT broker. Seznam aktivně odesílaných komponent může klientská aplikace kdykoliv změnit opětovným voláním REST API endpointu. Stejně tak lze přes WebSocket nastavovat nativní komponenty.

5.2 Podpůrná aplikace Server Simulator

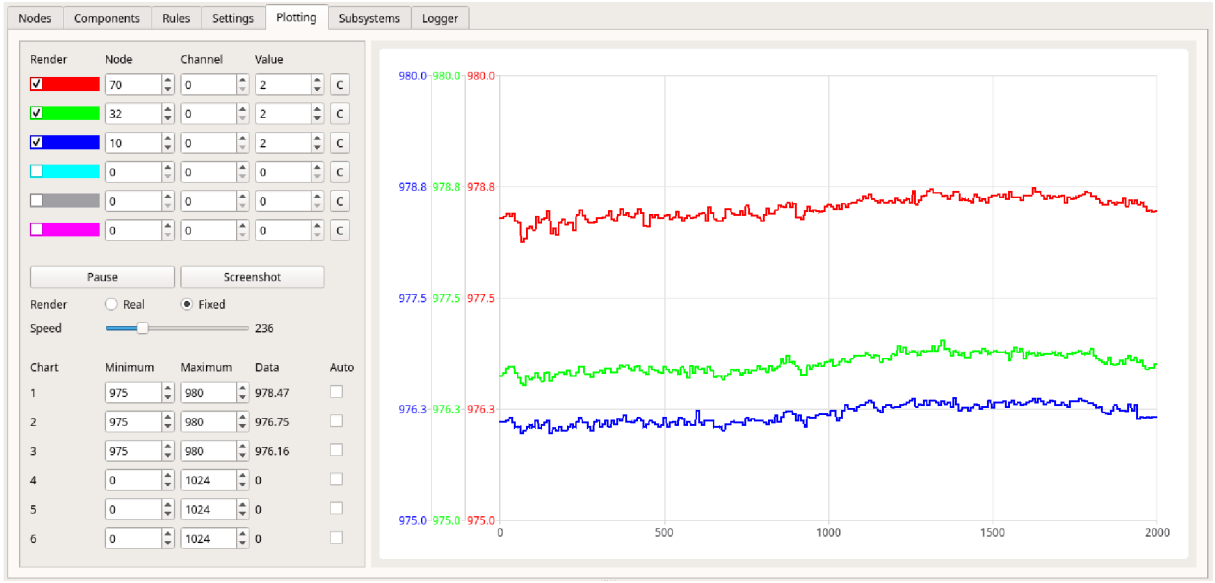
Protože v současném stavu jsou implementace IoTIM mikroslužeb ve vývoji, byla navržena aplikace s názvem Server Simulator za účelem chování IoTIM co nejvíce napodobit. Tento krok byl nezbytný, jelikož všechny CRUD operace, konfigurace a management subsystému jsou prováděny middlewarem. Dále také vznikla při vývoji nodů potřeba logování dat z důvodu verifikace správné funkčnosti jejich hardwaru a firmwaru. Aplikace je vyvinuta v programovacím jazyce C++ v kombinaci s frameworkem Qt5. Aplikace je dispozici pro operační systémy Windows 10 a Linux pod open-source licencí GPLv3.

Aplikace může být využita k managementu subsystému. Lze vyvolat hledání nových nodů na různých sběrnicích a ručně přidělit jejich adresu, provádět CRUD nodů, komponent a pravidel a také nastavit nový subsystém v lokální síti. Veškerá data, chybové stavy a informace lze zobrazovat v konzoli. Data komponent je možné ukládat do CSV souborů s nastavitelnou cestou a nutným počtem hodnot. Ukládání lze kdykoliv pozastavit pro jednotlivé komponenty nebo pro všechny najednou. Aplikace je tak využitelná i jako datalogger. Odpovídání na zprávu o detekci přítomnosti IoTIM lze kdykoliv vypnout a uvést tak subsystém do off-line stavu. Dále je možné data (až 6 složek různých komponent) v reálném čase zobrazovat v grafu. Pro každou komponentu si uživatel může nastavit individuální měřítko nebo osy škálovat dynamicky. Vykreslování je buď dle rychlosti toku dat nebo s nastavitelnou periodou obnovování grafu. Kreslení grafu lze kdykoliv pozastavit nebo získat screenshot ve formátu PNG. Na obrázku 5.2

je zobrazen pohled na komponenty několika subsystémů, obrázek 5.3 ukazuje možnosti zobrazení dat v grafu.



Obrázek 5.2: Aplikace Server Simulator [autor]



Obrázek 5.3: Grafy v aplikaci Server Simulator [autor]

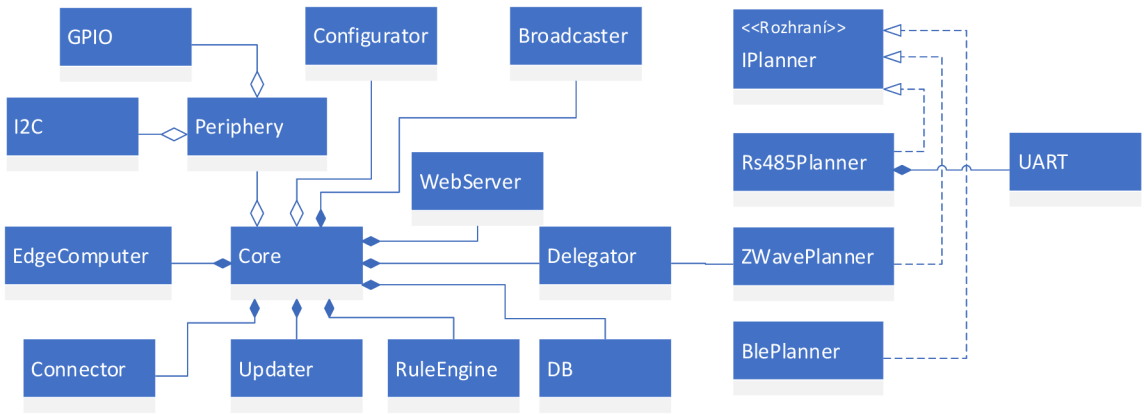
6

Ukázková implementace subsystému

Tato kapitola se zabývá prostřední vrstvou nového řešení – subsystémem. Jsou popsány vybrané části implementace a také vytvořený hardware sloužící pro komunikaci s nejnižší vrstvou řešení – nody. Protože aplikace využívá výkonově omezený ARM hardware, je realizována jako monolitická aplikace s podporou vláken a možností aplikaci rychle přizpůsobit novým generacím ARM mikroprocesorů díky interní architektuře založené na frontách událostí. Výhodou monolitické aplikace je možnost distribuovat celou aplikaci subsystému jako systémový balíček. Aplikace využívá nativních hardwarových přístupů operačního systému Linux a jiné OS nejsou podporovány

6.1 Vztah jádra k ostatním modulům aplikace

Každá C++ aplikace se spouští voláním metody `main`. V případě subsystému je pouze vytvořena instance třídy `Core`, tedy jádro celé aplikace. Jádro obsahuje několik členských instancí objektů, například `Delegator delegator`, reflektujících návrhový diagram 4.4.1. Jádro implementuje rozhraní listenerů a ve svém konstruktoru nastavuje svou instanci třídám, které listenery využívají ke komunikaci s jádrem. Listenery slouží k asynchronní komunikaci s jádrem.



Obrázek 6.1: Vztah jádra a ostatních tříd [autor]

Pokud jádro vyžaduje například změnu stavu nodu pro zápis, volá na instanci delegátoru metodu `updateNodeData`. Opačně pokud dojde k vyčtení nového stavu nodu pro čtení, volá delegátor na jeho

instanci listeneru metodu `readNodeChanged`, která o tomto stavu upozorní jádro v implementované metodě. Všechny třídy využívající listenery pro komunikaci s jádrem jsou znázorněny na obrázku 6.1. Mezi virtuální třídy, které jádro implementuje jako listenery, patří `WebServer`, `RuleEngine`, `Updater`, `Connector`, `Periphery`, `EdgeComputer` a `Delegator`. V metodě `main` je následně volána metoda `start` jádra, která zjišťuje stav subsystému (konfigurovaný nebo čekající na konfiguraci) a inicializuje moduly databáze, delegátoru, ladícího webserveru atd.

6.1.1 Tikání jádra

Z důvodu, že mnoho činností subsystému musí být periodicky spouštěno a nejedná se o výpočetně dlouhé úkony vyžadující vytvoření nového vlákna, využívá jádro principu tikání. V nekonečné smyčce je odměřován systémový čas a v každém průchodu smyčkou je volána metoda komunikačního bloku `connector.loop(50);`.
Ta po dobu 50 milisekund čeká na příchozí MQTT zprávu, a pokud je nějaká přijata, je metoda ukončena předčasně. V dalším kroku je srovnán aktuální čas s několika pomocnými proměnnými a pokud je překročen, je provedena akce tiknutí. Například je každých 100 milisekund kontrolován stav periferního modulu a každých deset sekund kontrolován stav výpočetních modulů. Plánovače pak tikání využívají k různým činnostem, jako je časování znovupřipojení k neaktivním bezdrátovým zařízením. Každých 500 milisekund je také událost tiknutí propagována do pravidlového modulu k vyhodnocení pravidel se spouštěčem časovače. Pokud je přijata MQTT zpráva mimo okamžik volání smyčky komunikačního bloku, není zpráva ztracena, ale umístěna do fronty.

6.2 Komunikace s IoTIM

FlatBuffers zprávy tvoří formát pro standardizovanou výměnu informací mezi subsystémy a IoTIM, samotná komunikace probíhá skrze MQTT, kde jsou témata použity k organizaci vyžadovaných činností. Například zprávu `ComponentFB` lze využít k přidání nové komponenty, úpravy komponenty stávající nebo k jejímu odstranění. Pro rozlišení se využívají právě MQTT témata, která jsou shrnuta v tabulce 6.1. Komunikace může vždy probíhat pouze mezi serverem a subsystémem, jiné kombinace neexistují. Oproti návrhu zpráv v kapitole 4.4.3 došlo k přidání informační zprávy `BatteryStateFB`, informující IoTIM o změně nabití baterie bezdrátového nodu a `DescriptionFB` sloužící ke změně metadat objektů.

Tabulka 6.1: Vztah MQTT témat a FlatBuffers zpráv [autor]

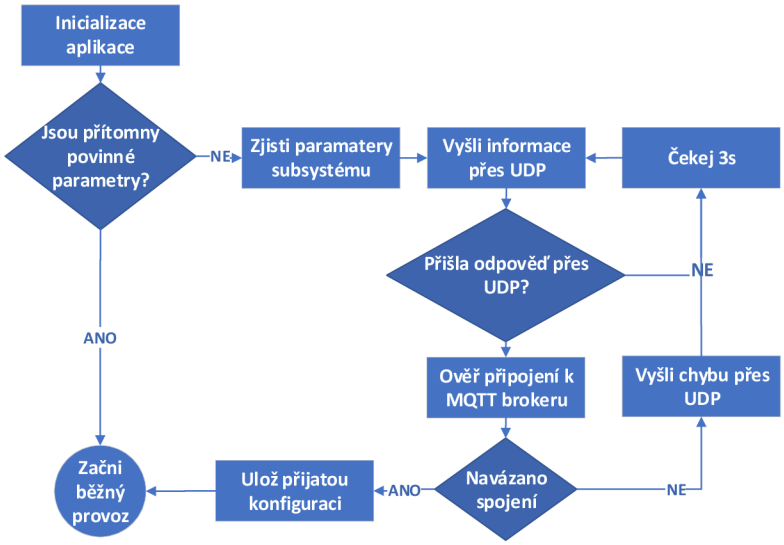
MQTT téma	Flatbuffer zpráva	Odesílatel
component/add	ComponentFB	IoTIM
component/change	ComponentFB	IoTIM
component/read_data	DataFB	Subsystém
component/remove	ComponentFB	IoTIM
component/write	DataFB	IoTIM
edge/add	EdgeComputerFB	IoTIM
edge/remove	EdgeComputerFB	IoTIM
error	ErrorFB	Subsystém
group/change	RuleGroupFB	IoTIM
group/remove	RuleGroupFB	IoTIM
info	InfoFB	Subsystém
node/add	NodeFB	IoTIM
node/address_assign	NodesFB	IoTIM
node/battery_state	BatteryStateFB	Subsystém
node/identify	IdentifyFB	IoTIM
node/remove	NodeFB	IoTIM
node/scan_request	ScanRequestFB	IoTIM
node/scan_response	NodesFB	Subsystém
rule/add	RuleFB	IoTIM
rule/change	RuleFB	IoTIM
rule/remove	RuleFB	IoTIM
subsystem/fw_update_announce	FWUpdateAnnounceFB	IoTIM
subsystem/fw_update_confirmation	FWUpdateConfirmationFB	Subsystém
subsystem/fw_update_start	FWUpdateStartFB	IoTIM
subsystem/ping	PingPongFB	Subsystém
subsystem/pong	PingPongFB	IoTIM
subsystem/sitrep_request	SitrepRequestFB	IoTIM
subsystem/sitrep_response	SitrepResponseFB	Subsystém
update_description	DescriptionFB	IoTIM

6.2.1 Prvotní autokonfigurace subsystému v lokální síti

V lokálním prostředí je celé řešení nasazeno jako kompletně oddělená síť, kdy server obsahuje dvě síťové karty a provádí routing mezi vnějším světem a subsystémy. Poté by bylo možné stanovit, že adresa serveru a konfigurace sítě bude vždy stejná. Tím pádem by mohly všechny subsystémy obsahovat pevnou adresu serveru, např.: 10.0.0.1 s maskou sítě 255.255.0.0. Tento ideální případ je ovšem v naprosté většině situací nerealizovatelný, ať již z důvodu nutnosti fyzického serveru se dvěma síťovými adaptéry, dedikovaného síťového přepínače nebo provozu IoTIM na VPS či aplikačním cloudu. Z toho důvodu bylo vyvinuto řešení, jak automaticky konfigurovat subsystém, pokud obsahuje čistý obraz operačního systému s aplikací a nezná IP adresu a port MQTT brokera, se kterým může komunikovat se serverem.

Flow diagram postupu je znázorněn na 6.2. Při spuštění aplikace je provedena inicializace tříd a hardwarových komponent. Poté se zkontroluje obsah konfiguračního souboru `subsystem.conf` v JSON formátu. Neobsahuje-li záznam o identifikátoru subsystému a údajích o MQTT brokeru (IP adresa nebo jméno hosta, port, uživatelské jméno a heslo), sestaví se hardwarový report s informacemi jako je IP

adresa subsystému, MAC adresa ethernetového portu, celková a dostupná paměť RAM, dostupné a volné místo na SD kartě. Také se sestaví balíček integrovaných komponent (notifikační LED, tlačítka, teploměr) a schopností subsystému. Těmi je myšleno jaké sběrnice umí subsystém obsloužit (RS485, Bluetooth Low Energy nebo Z-Wave). Tyto informace jsou vyslány jako FlatBuffer zpráva skrze UDP broadcast na portu 10001. Pokud je server přítomen, přijme zprávu a přiřadí subsystému identifikátor. Na portu 10003 vyšle UDP broadcast s konfigurační odpovědí. Ve zprávě je také uvedena MAC adresa subsystému pro případ, že je konfigurováno více subsystémů zároveň. Ve zprávě lze také uvést seznam nodů, komponent, pravidel a výpočetních modulů. Tím je možné okamžitě převést subsystém do funkčního stavu, pokud se jedná o migraci staršího subsystému na nový, např. výměna po hardwarové poruše starého. Aplikace se poté pokusí připojit na MQTT broker. Pokud to z nějakého důvodu není možné, odesílá chybovou zprávu opět přes UDP broadcast a čeká na opravenou konfiguraci. Pokud je vše v pořádku, přechází subsystém do korektního nastaveného stavu a informace si ukládá do konfiguračního souboru. Pokud nastane situace, kdy je subsystém připojován k IoTIM běžícím v aplikačním cloudu nebo lokální síť blokuje UDP broadcast, lze ke konfiguraci využít aplikaci Server Simulator.



Obrázek 6.2: Postup konfigurace subsystému přes Ethernet [autor]

6.3 Podpora edge computingu

Výpočty v subsystému jsou realizovány jako externí aplikace, které komunikují se subsystémem pomocí TCP spojení. Subsystém je poprvé přijme skrze FlatBuffer zprávu `EdgeComputerFB`. Na každý výpočetní modul je nahlíženo jako na aplikaci, kterou je třeba rozbalit a spustit. Ve zprávě je přítomno ID sloužící k identifikaci aplikace a jako klíč pro databázi. Dále zpráva obsahuje popisek modulu, název spustitelného souboru a list nativních komponent vyžadovaných pro výpočty a list edge komponent, které musí IoTIM odeslat subsystému před samotným modulem. Subsystém při přijetí modulu provede rozbalení TAR balíčku, přítomného ve zprávě jako binární pole. Výchozí lokací je složka `/usr/bin/subsystem/ID/`, kde ID je identifikátor modulu. Subsystém vytvoří složku, provede rozbalení a ve třídě `EdgeComputer` vytvoří nový proces, kterým nahradí spustitelný soubor a ukládá si

jeho PID. Při inicializaci aplikace je ve třídě `EdgeComputer` vytvořen naslouchací TCP socket na portu 8888. Výpočetní aplikace po svém vlastním spuštění vytváří klientské spojení a odesílá subsystému zprávu obsahující PID aplikace pro správné párování socketu a modulu. Subsystém při aktualizaci komponenty předává její data třídě `EdgeComputer` a ta pomocí mapy zjišťuje, kterému modulu patří. Zároveň třída periodicky kontroluje existenci PID modulů. Pokud PID již neexistuje, modul přestal pracovat a třída jej znovu spouští, aktualizuje si jeho PID a odesílá jádru chybovou zprávu. Pokud modul vypočte potřebné hodnoty, odesílá je přes socket subsystému jako JSON zprávu. Tu třída `EdgeComputer` zpracuje a předává data jádru.

6.4 Komunikace s Nody

Každá technologie je obsluhována ve své vlastní třídě. Pro komunikaci s kabelovými nody skrze sběrnici RS-485 se jedná o třídu `Rs485Planner`, Bluetooth Low Energy nody jsou obsluhovány třídou `BlePlanner` a Z-Wave zařízení využívají třídu `ZWavePlanner`. Všechny plánovače implementují virtuální třídu `IPlanner`, díky které může třída `Delegator` volat vždy stejné metody pro nastavování stavu nodů a jejich management.

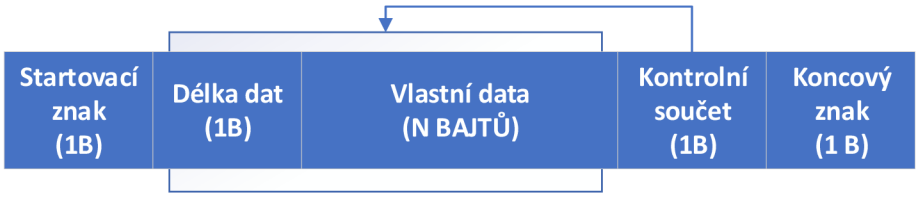
6.4.1 RS-485

Technologie RS-485 byla zvolena z důvodu cenové dostupnosti, dostatečné přenosové rychlosti a topologii. Oproti sběrnici CAN bus, která je omezena na pouhých 8 bajtů vlastních dat, není RS-485 jakkoliv omezena počtem přenášených bajtů. Díky sběrnicové topologii lze na rozdíl od Ethernetu nody řetězit.

Namísto využití již existujících protokolů Modbus nebo ProfiBus byla zvolena cesta vyvinutí vlastního komunikačního protokolu. Oba protokoly jsou velmi robustní, ale kvůli podpoře různých přenosových technologií obsahují v datových rámcích redundantní data, nepotřebná pro účely aplikace.

Zde jsou pro běžnou komunikaci vyžadovány pouze příkazy pro nastavení nodu, získání dat z nodu a jeho prvotní adresaci. Modbus i ProfiBus předpokládají, že každé zařízení na sběrnici může obsahovat až tisíce konfiguračních registrů, které mohou být nastavovány. V každém datovém rámci se pak odesílá zdrojová adresa, cílová adresa, typ požadované akce a adresa registru. Oba protokoly umožňují přes sběrnici RS-485 odeslat i přijmout několik stovek zpráv za sekundu. Požadavky některých senzorických projektů jsou ovšem vyšší, přes tisíc zpráv za sekundu.

Implementovaný protokol využívá datového paketu znázorněného na obrázku 6.3. Každý paket začíná bajtem se startovacím znakem (decimální hodnota 35, odpovídající ASCII znaku #), následuje nenulový bajt s délkou vlastních dat. Minimální délka vlastních dat je jeden bajt a teoretické maximum až 255 bajtů. Vlastní data vždy obsahují alespoň identifikátor zprávy. Předposledním bajtem v rámci je kontrolní součet CRC8, vypočítaný z bajtů vlastních dat a bajtu s jejich délkou. Poslední bajt je koncový znak s binární hodnotou 36, což v ASCII odpovídá znaku $. Subsystém i nody paket akceptují jako validní pouze v případě, že jsou splněny všechny požadavky paketu. Pokud nody přijmou nevalidní paket, je ignorován. Subsystém při přijetí nevalidního paketu tuto událost loguje. Komunikace probíhá přenosovou rychlostí 384 000 baudů za sekundu. Mezi nody a subsystémem bylo navrženo celkem jedenáct zpráv obsažených v paketu, které lze rozdělit do čtyř kategorií. První slouží pro objevení nodů, druhá pro přidělování adres, třetí pro běžnou komunikaci a poslední pro identifikaci a migraci.



Obrázek 6.3: Formát paketu [autor]

Zprávy pro objevení nodů

Subsystém využívá tuto skupinu zpráv, aby objevil nové nody bez přidělené adresy a zjistil jejich vlastnosti. Všechny zprávy jsou v programu uvozeny prefixem `MESSAGE_`, který je v textu pro přehlednost odstraněn.

DISCOVERY_REQUEST
Tuto zprávu posílá subsystém pro zjištění přítomnosti nových nodů bez přidělené adresy a neobsahuje žádná dodatečná data. Nody s adresou zprávu ignorují a ty bez ní využití plovoucí pin k vyčtení elektrického šumu prostředí a vygenerování pseudonáhodného čísla. Před odesláním odpovědi čekají tolik milisekund, kolik je velikost vygenerovaného čísla.

DISCOVERY_RESPONSE
Nody bez adresy posílají tuto zprávu jako odpověď na `DISCOVERY_REQUEST` zprávu. Obsahuje pseudonáhodné číslo, které je přítomno, aby jej subsystém mohl později odeslat nodu, který původní zprávu vyslal a došlo k přidělení správných adres.

REVEAL_REQUEST
Pokud subsystém během časového okna pro objevování nodů obdržel alespoň jednu zprávu typu `DISCOVERY_RESPONSE`, odesílá na sběrnici tuto zprávu, která obsahuje pouze pseudonáhodné číslo generované novým nodem během přijetí `DISCOVERY_REQUEST`.

REVEAL_RESPONSE
Nod, který není adresovaný, ale obdržel `DISCOVERY_REQUEST` a zároveň `REVEAL_REQUEST` odesílá tuto zprávu s následujícím obsahem:

- Pseudonáhodné číslo (pro potvrzení subsystému, že jde o identický nod),
- šířku dat jedné složky,
- počet složek,
- zda je nod pro čtení nebo zápis, pokud pro čten, tak jaká je jeho latence[1],
- počet kanálů nodu,
- pokud je nod čten a vyžaduje prodlevu, zde je přítomna informace o minimální latenci nodu,
- verze firmwaru nodu,
- krátký popisek firmwaru nodu.

[1] Ta může být buď pro okamžité čtení nebo s minimálním časovým kvantem.

Zprávy pro adresaci nodů

Pokud jsou objeveny nové nody, subsystém využívá tuto skupinu zpráv ke spolehlivému přidělení adres nodů.

ADDRESS_SET
Subsystém posílá tuto zprávu nodům bez adresy. Obsahuje pseudonáhodné číslo, které generoval nod, a jeho novou adresu. Nod si adresu zatím ukládá pouze do paměti RAM. Nod na tuto zprávu reaguje pouze po úspěšném objevení dle předchozích zpráv pro objevení.

ADDRESS_CHECK
Po přijetí zprávy `ADDRESS_SET` odesílá nod zpět subsystému tuto zprávu pro zpětnou verifikaci. Součástí zprávy je nová adresa nodu a opět pseudonáhodné číslo.

ADDRESS_SAVE
Když subsystém obdrží zprávu `ADDRESS_CHECK` a pseudonáhodné číslo a adresa souhlasí, odešle nodu tuto zprávu. Ten po jejím zpracování ukládá svou adresu do paměti EEPROM nebo Flash dle typu mikrokontroléru a přechází do plně funkčního adresovaného stavu. Při svém dalším spuštění čte firmware nodu adresu a pokud je přítomna, je smyčka pro přidělení adresy okamžitě přeskočena.

Zprávy pro běžnou komunikaci

Tato skupina zpráv se využívá pro vyžádání nového stavu nodu, jeho načtení či nastavení nového stavu nodu a zpětnou verifikaci.

REQUEST
Subsystém touto zprávou vyžaduje od nodu odeslání aktuálního stavu připojených senzorů nebo ovládacích prvků. Zpráva obsahuje kromě identifikátoru pouze adresu cílového nodu.

GET
Touto zprávou odpovídá nod na zprávu `REQUEST` nebo `SET`. Zpráva obsahuje adresu nodu následovanou polem naměřených dat. Počet bajtů dat lze vypočítat vynásobením počtu dat typu nodu počtem složek typu a počtem kanálů nodu. Výsledek je pro přepočet na bajty nutno vydělit 8 a případně zaokrouhlit nahoru[2].

MEASURE
Některé nody obsahují digitální senzory a vyčtení jejich stavu trvá určitou dobu. Během tohoto času nod nemusí reagovat na dění na sběrnici. Po přijetí zprávy `MEASURE` nod zjistí stavy senzorů, ale na jejich odeslání subsystému poté vyčkává do přijetí zprávy `REQUEST`, na kterou reaguje zprávou `GET`. Subsystém ví, kterému nodu je nutné odeslat před `REQUEST` a `GET` cyklem zprávu `MEASURE` podle jeho identifikátoru typu, respektive proměnné `accessType` v objektu typu.

SET
Pokud nod obsahuje aktory (respektive libovolné výstupy), subsystém mění jejich stav právě touto zprávou. Pro ověření, že nod zprávu přijal a zpracoval, musí odpovědět subsystému zprávou `GET`, obsahující nový stav.

[2] Přenosová jednotka je jeden bajt, proto i nod který vrací pouze jediný bit, odešle stav jako celý bajt.

Zprávy pro identifikaci a začlenění

Tyto zprávy může využít administrátor systému buď pro identifikaci jednotlivých nodů, pokud adresoval několik nodů stejných charakteristik, nebo k začlenění již existujících nodů do migrovaného subsystému.

IDENTIFICATION_ON
Touto zprávou může subsystém zapnout na nodu blikání indikační LED. Zpráva obsahuje adresu nodu a je užitečná zejména v případě, kdy administrátor adresoval více nodů stejných charakteristik a chce je od sebe rozlišit.

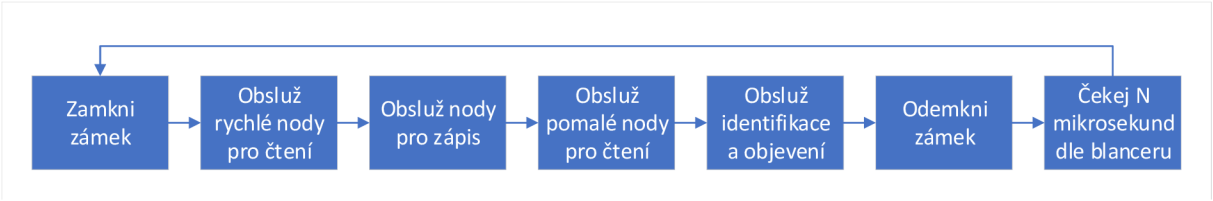
IDENTIFICATION_OFF
Touto zprávou subsystém naopak vypíná blikání indikační LED na nodu.

PROPERTIES_REQUEST
Pokud došlo k fatální poruše subsystému, nebo pokud byl nod přesunut a odstraněn z jiného subsystému bez vymazání paměti, lze tuto zprávu využít k odeslání charakteristik nodu zpět na subsystém. Součástí zprávy je vyjma identifikátoru pouze adresa nodu. Pokud takový nod existuje, odesílá odpověď.

PROPERTIES_RESPONSE
Po přijetí zprávy `PROPERTIES_REQUEST` odesílá adresovaný nod odpověď `PROPERTIES_RESPONSE`, která se od zprávy `REVEAL_RESPONSE` liší pouze tím, že na místo pseudonáhodného čísla obsahuje adresu nodu. Zbylá struktura zprávy je identická.

Plánovač RS-485 nodů

Třída `Rs485Planner` má za cíl provádět obsluhu kabelových nodů dle aktuálních požadavků jádra. Její instance je přítomna ve třídě `Delegator` a samotná třída má naopak v konstruktoru předaný ukazatel na instanci delegátoru. Plánovač se startuje metodou start(), ta nejprve zjistí z konfiguračního souboru název UART portu a požadovanou přenosovou rychlost (baudrate) a pokusí se otevřít UART. Pokud se tento proces nezdaří, je do vektoru `permanentErrors<ErrorStruct>` vložena chybová hláška, která bude periodicky odesílaná IoTIM, aby mohlo dojít k její opravě.



Obrázek 6.4: Sekvence obsluhy kabelových nodů [autor]

Pokud je inicializace UART úspěšná, plánovač vytváří nové vlákno, ve kterém bude běžet nekonečný cyklus pro obsluhu nodů. Jejich obsluha probíhá ve smyčce znázorněné v diagramu 6.4. Nejprve plánovač uzamkne mutex, indikující že probíhá cyklus manipulace s nody. Poté plánovač iteruje skrze mapu nodů pro čtení bez latence a posílá jim zprávu `REQUEST` a čeká 10 milisekund na odpověď. Pokud během tohoto intervalu neobdrží zprávu `GET`, odesílá jádru chybový stav. Následně plánovač iteruje mapu nodů pro zápis a pokud je třeba nodu poslat nový stav, je stav odeslán zprávou `SET` a čeká se na potvrzení zprávou `GET`. Poté je iterováno skrze mapu s nody pro čtení s vyžadovanou prodlevou. Pokud uběhla od poslední komunikace s nodem doba rovná polovině intervalu minimální latence, plánovač dle pomocné

proměnné cykluje mezi zprávou `MEASSURE`, na kterou nevyžaduje odpověď, a zprávou `REQUEST`, na kterou očekává odpověď. V předposledním kroku plánovač kontroluje, zdali není požadavek na změnu stavu identifikace nodu. Posledním krokem je odemknutí zámku a uspání vlákna po dobu N mikrosekund, dle aktuálního vytížení dalších části subsystému. Plánovač odesílá jádru pouze změny stavu nodů.

6.4.2 Bluetooth Low Energy

Technologie Bluetooth Low Energy (BLE) specifikuje podrobný komunikační protokol pomocí hierarchického systému generických atributů (GATT). Nejvyšší vrstvou jsou profily, které udávají, jakou technologií (BLE nebo Bluetooth Classic) zařízení komunikuje, a verzi komunikačního standardu. Každé zařízení pak vystavuje služby a v nich charakteristiky. Všechny složky GATT musí mít uvedeny unikátní identifikátory. Bluetooth SIG specifikuje standardní služby a charakteristiky. Příkladem budiž služba `Heart Rate` s identifikátorem 0x180D. Služba slouží pro čtení srdečního tepu. Služba specifikuje povinnou charakteristiku `Heart Rate Measurement`, která je pro čtení, a obsahuje aktuální počet tepů za minutu a volitelnou charakteristiku[3] `Body Location`, také pro čtení s informací o fyzickém umístění senzoru na lidském těle.

Vývojáři pak mohou vytvářet zařízení skládáním již definovaných služeb. Například fitness náramek může implementovat službu srdečního tepu, stav baterie, krevního tlaku a počtu ušlých kroků za den. V praxi se ovšem často stává, že výrobci zařízení chtějí, aby uživatelé využívali pouze jimi vyvinuté doprovodné aplikace pro chytré telefony nebo počítače a implementují si služby vlastní, bez veřejně dostupné dokumentace.

Komunikaci s nody zajišťuje třída `BlePlanner`. V operačním systému Linux se ke komunikaci s periférie mi Bluetooth Classic nebo BLE využívá démon BlueZ. Doporučený způsob je komunikovat s démonem přes D-BUS, což je technologie pro komunikace mezi procesy pro operační systém Linux. Pokud programátor vyžaduje nějakou činnost po BlueZ, například iniciaci skenování, musí sestavit D-BUS zprávu, označit BlueZ jako adresáta a čekat na odpověď. Pro odstínění od skládání zpráv a čekání na odpověď je využita knihovna tinyB. Obsahuje metody pro skenování zařízení, listovaní profilů, služeb a charakteristik BLE zařízení, zapisování charakteristik do zařízení. Čtení změn charakteristik je prováděno v asynchronním listeneru. Protože nody neobsahují metadata, byly navrženy univerzální služby a charakteristiky reflektující tento přístup. Služba `WRITE_SERVICE` je služba pro zapisovatelné nody a specifikuje povinnou charakteristiku `WRITE_CHARACTERISTICS`. Pro nody s typem čtení je specifikována služba `READ_SERVICE` se charakteristikou `READ_CHARACTERISTICS`. Vlastní přenášená data jsou definována jako pole bajtů s plovoucí délkou a jejich formát je identický jako datové části zpráv `MESSAGE_SET` a `MESSAGE_GET` protokolu kabelových nodů. Poslední službou je `IDENTIFY_SERVICE` sloužící k sepnutí identifikační LED nodu a stanovuje charakteristiku `IDENTIFY_CHARACTERISTICS` s jedním bajtem, kde hodnota 1 LED zapíná a hodnota 0 LED vypíná. Pokud se jedná o bezdrátový nod napájený z baterie a podporou měření jejího stavu, může nod vystavovat standardní službu `Battery Service`.

Adresace nových nodů je komplikovanější, jelikož na rozdíl od RS-485 se mohou v blízkosti subsystému vyskytovat i jiná BLE zařízení. Pokud není nod spojen se subsystémem, v periodickém intervalu 5 sekund vysílá advertisement paket. Součástí paketu je název zařízení, identifikátor výrobce a volitelná data. Volitelná data obsahují informace o vlastnostech nodu. Když jádro dostane od IoTIM pokyn k adresaci nových nodů, plánovač zapne přes tinyB funkci skenování na Bluetooth adaptéru. Po uplynutí

[3] Zařízení implementující Heart Rate službu nemusí tuto charakteristiku implementovat.

doby 10 sekund se začne seznam detekovaných zařízení procházet a zařízení bez správného identifikátoru jsou zahozeny. K novým nodům je v aplikaci pro úplnost vygenerováno náhodné číslo a objektu nodu je nastaveny jeho fyzická MAC adresa. Až IoTIM přidělí nové adresy, je navázáno spojení. Na rozdíl od kabelových nodů si BLE nody neukládají svoji adresu a vždy akceptují spojení s libovolným subsystémem, pokud došlo k rozpadu spojení předcházejícího. Lze tak vytvořit mobilní zařízení, které se může přesunovat v rámci místností a navazovat spojení se subsystémem s nejlepší sílou signálu.

6.4.3 Z-Wave

Zařízení komunikující technologií Z-Wave nejsou v této práci nazývány nody cíleně. Zatímco RS-485 a Bluetooth Low Energy nody mohou obsluhovat několik odlišných zařízení, výrobky pro inteligentní dům postavené na Z-Wave mají vždy jasně definovaný účel. Může se jednat o žárovku, stmívač, termostat, pohybový senzor atd., ovšem vždy je podstata zařízení jasná a Z-Wave protokol tento stav reflektuje. Specifikace protokolu Z-Wave definuje tzv. třídy příkazů (COMMAND_CLASS), které mohou mít několik zaměření. Buď se týkají popisu vlastností samotného zařízení (Aplikační třídy), správy komunikačního protokolu (Network-Protocol třídy), managementu komunikace a konfiguraci komunikační vrstvy (Transport-Encapsulation třídy). Příkladem může být nástěnný spínač, který bude definován aplikační třídou `COMMAND_CLASS_SWITCH_BINARY` značící, že se jedná o vypínač. Dále může vystavovat třídu komunikačního protokolu `COMMAND_CLASS_POWERLEVEL`, oznamující stav baterie zařízení, a také komunikační třídu `COMMAND_CLASS_SECURITY` pro oznámení řídící jednotce, že se jedná o zařízení podporující šifrovanou AES komunikaci Z-Wave+.

Veškerou komunikaci se Z-Wave zařízeními má na starost třída `ZWavePlanner`, implementující stejné rozhraní `IPlanner` jako pro ostatní vybrané technologie. Je využívána knihovna `openzwave` [197], vyvinutá za účelem komunikace se Z-Wave zařízeními prostřednictvím USB rádií. Knihovna využívá kolekci XML souborů s definicemi příkazových tříd. Každé podporované zařízení musí mít svůj XML soubor. Knihovna pak podle definice ví, jaká data které třídy má očekávat a provádí konverzi se surových binárních dat na celá čísla, desetinná čísla, textové řetězce a enumerace. Protože refaktoring tohoto vysokoúrovňového přístupu by vyžadoval desítky až stovky člověkohodin a musel by být trvale udržován vůči hlavní vývojové větvi knihovny, obsahuje třída `ZWavePlanner` prostředky pro reverzní mapování zpět na surová data.

Třída se ve své inicializační metodě prostřednictvím konfigurační třídy `Configurator` načítá mapovací soubor `zwavemappings.json`, obsahující informace kterou třídu příkazu Z-Wave zařízení mapovat na jaký typ a složku nodu, viz ukázka 6.1.

Výpis 6.1: Ukázka mapování kombinovaného senzoru

```
{ "name" : "Heiman HS1HT Smart Temperature and Humidity Sensor",
  "product_id" : 4096,
  "manufacturer_id" : 608,
  "nodes" : [{
       "id" : 1,
       "channels": 1,
       "data_range": 32,
       "data_count": 2,
       "direction": "R",
       "latency": "N",
       "min_delay": 0 ,
       "fw_description": "H1S1HT Temperature and Humidity reading"
  }], "mappings" : [
     {  "node_id" : 1,
        "node_channel" : 0,
        "zwave_channel": 49,
        "zwave_channel_index": 1,
        "zwave_data_type": "FLOAT",
        "zwave_direction": "R"
     },{"node_id" : 1,
        "node_channel" : 1,
        "zwave_channel": 49,
        "zwave_channel_index": 5,
        "zwave_data_type": "FLOAT",
        "zwave_direction": "R"
     }],
  "initial_sequence":[ ]
}
```

Proces přidělení adresy nových zařízení se v terminologii Z-Wave technologie nazývá inkluze. V tomto režimu poslouchá USB adaptér multicast zprávy od zařízení bez logické adresy. Na nových zařízeních je vždy nutné například stiskem servisního tlačítka explicitně spustit proces inkluze, aby omylem nedošlo k přiřazení zařízení do nevhodné sítě. Spuštění a zastavení inkluze je nutné provést programově voláním knihovny. Třída `ZWavePlanner` inkluzi spouští na pokyn serveru po dobu 30 sekund. Poté jsou procházeny výsledky a je-li dle mapování nalezena shoda s identifikátorem výrobce a zařízení, je vytvořen nový neadresovaný nod spolu s pomocnou strukturou obsahující nezbytné informace pro konverzi dat. Třída `ZWavePlanner` během inkluze také kontroluje přítomnost příkazové třídy `COMMAND_CLASS_BATTERY` indikující zařízení napájené z baterie. Pomocí tikání jádra je pak každých 15 minut kontrolován stav baterie a předáván zpět jádru.

6.4.4 Proces mapování nodů pro čtení na komponenty

Když instance tříd `Rs485Planner`, `BlePlanner` nebo `ZwavePlanner` vyčtou nový stav nodu, resp. Z-Wave zařízení, volají metodu delegátoru `updateNodeData` s vektorem nových dat, adresou nodu a časovou známkou jako argument. Delegátor pouze volá metodu rozhraní `delegatorReadNodeChanged` se stejnými argumenty implementovanou jádrem. Místo přímého mapování dat nodu na komponenty je tento proces proveden asynchronně, aby nebyla vlákna plánovačů blokována během mapovací operace. Místo toho je v jádře, které si udržuje kopie objektů nodů v mapě, nalezen správný objekt, aktualizována jeho data a je vložen do fronty `dataQueue`, viz ukázka 6.2.

Výpis 6.2: Ukázka metody vkládající nody do fronty

```
1 void Core::delegatorReadNodeChanged(u16 addr, vector<u32> data, u64 time)
2 {
3     Node *n = &readNodes[addr];
4     n->setData(data);
5     n->setSampleTimestamp(time);
6     dataQueue.push(*n);
7 }
```

Jádro obsahuje metodu `dataProcessLoop` s nekonečným cyklem, spouštěnou ve vlastním vlákně, která je blokována voláním
`Node *n = &dataQueue.pop();`.
Vlákno čeká a při vložení objektu nodu na druhém konci fronty je nad všemi kanály v cyklu sestaven kompozitní klíč komponenty. Dle klíče je hledán objekt komponenty v mapě `readComponents`. Pokud je nalezena komponenta[4], je přenastavena časová známka a současná data nodu. Voláním `nodeToComponent` na objektu komponenty je vrácena enumerace `ConversionState` nabývající po přepočtu knihovnou `muParser` těchto možných stavů:

- `SAME` - nový stav komponenty je identický s předchozím,
- `CHANGE` - nový stav komponenty je odlišný oproti předchozímu,
- `IS_NAN` - přepočet knihovnou muParser nevrátil platné číslo,
- `PARSE_ERROR` - přepočtový výraz obsahuje syntaktickou chybu.

V případě stavu `SAME` není již provedena žádná akce. U všech ostatních je dle stavu subsystému výsledek buď odeslán serveru, nebo logován do úložiště. V případě dat se v off-line stavu data komponenty přidávají do fronty pro pravidlový systém.

6.5 Pravidlový systém

Veškerá potřebná funkcionalita pro vykování pravidel je implementovaná ve třídě `Rules`. Obdobně jako převod nodů pro čtení na komponenty je veškerá komunikace s pravidlovým systémem realizována přes frontu. Pravidlový systém pak ve svém vlákně čeká na pokyny a provádí reakce. Instrukce se

[4] Nod nemusí nezbytně využívat všechny své kanály.

pravidlovému systému odesílá sestavením struktury `EngineCommand` a jejím vložením do fronty `rulesQueue`. Struktura obsahuje enumeraci `EngineAction` určující akci kterou má pravidlový systém provést. Lze provést 10 odlišných akcí, každá pak vyžaduje nastavení odlišných proměnných ve struktuře `EngineCommand`.

COMPONENT_EVALUATE
Jedná se o nejběžnější pokyn pravidlovému systému, který slouží k aktualizaci stavu komponenty. Pravidlový systém pak okamžitě vyhodnocuje pravidla se spouštěčem `COMPONENT` a ve své interní kolekci komponent si data komponenty aktualizuje pro výpočet pravidel s jinými spouštěči.

TICK_EVENT
Jádro odesílá události tiknutí po přetečení času `RULE_ENGINE_TICK_TIME`. Slouží pro vyhodnocení pravidel se spouštěčem časovače nebo času dne.

SUBSYSTEM_OFFLINE
`SUBSYSTEM_OFFLINE` je odeslán pravidlovému systému při přechodu subsystému do nouzového off-line stavu. Ihned jsou vyhodnocena pravidla s tímto spouštěčem.

SUBSYSTEM_ONLINE
`SUBSYSTEM_ONLINE` je odeslán pravidlovému systému při přechodu subsystému zpět do on-line stavu.

RULE_ADD
Jádro vyžaduje přidání nového pravidla do pravidlového systému. CRUD operace jsou přijímány pouze v on-line stavu od serveru.

RULE_CHANGE
Jádro vyžaduje modifikace existujícího pravidla v pravidlovém systému.

RULE_REMOVE
Jádro vyžaduje odstranění existujícího pravidla z pravidlového systému.

GROUP_CHANGE
Tímto pokynem lze upravit stav skupiny pravidel na aktivní nebo vypnutou.

GROUP_REMOVE
Pravidlový systém dle identifikátoru skupiny maže všechna pravidla s tímto skupinovým ID.

COMPONENT_DELETE
Před odstraněním komponenty mapované na nod jádro ověřuje, zdali není obsaženo v nějakém pravidlu. Pokud ano, musí uživatel nejprve upravit nebo odstranit pravidla tuto komponentu využívající. Pravidlový systém pak komunikuje zpět s jádrem stejným způsobem jako ostatní bloky aplikace, tedy pomocí volání listeneru třídy (zde se jedná o `RulesListener`).

6.6 Detekce off-line stavu

Detekce off-line stavu probíhá na dvou úrovních, MQTT spojení a odpovědí serveru na `PingPongFB` zprávy. Klientská knihovna `mosquitto` obsahuje listener, který je volán pokud nebylo možné navázat spojení v MQTT brokerem, případně došlo k rozpadu již navázaného spojení. Třída `Connector` dědí metody třídy `MQTT` a přes listener oznamuje jádru navázání spojení metodou `connectorConnected`

a ztráty spojení `connectorDisconneted`. Implementace metod nastavuje hodnotu členské proměnné `mqttOnline`. V hlavní smyčce třídy zodpovědné za tikání je odměřován uplynulý systémový čas a překročí-li hodnotu definice `RECONNECT_TICK_TIME` (výchozí hodnota 2 sekundy), je proveden pokus o znovupřipojení. Komunikace se serverem není bez MQTT spojení prováděna a subsystém se nachází v off-line stavu.

Druhým stupněm je kontrola dostupnosti serverové aplikace. Po uplynutí času `PING_TICK_TIME` (ve výchozím sestavení stanoveným na jednu sekundu) subsystém cykluje dvě činnosti. Nejdříve do lokální proměnné `pingTimestamp` ukládá časovou známku a pak odesílá serveru přes třídu `Connector` zprávu `PingPongFB` s identifikátorem subsystému, touto časovou známkou a nulovou časovou známkou serveru. Server má sekundu na přijetí zprávy, doplnění své časové známky a odeslání zprávy zpět subsystému. Přijetí zprávy subsystémem má za následek přepsání členské proměnné `subsystemTimestamp` v jádře.

V druhém kroku porovnává jádro ve smyčce lokální proměnnou `pingTimestamp` a členskou proměnnou `subsystemTimestamp`. Když jsou rovny, znamená to přítomnost serveru a subsystém pokračuje v on-line režimu. Pokud byl v předchozím cyklu off-line režim, subsystém odesílá všechny současné stavy komponent pro synchronizaci serveru. V případě dostupných off-line logů je navázáno TCP spojení s rsyslog serverem a okamžitě jsou off-line logy přenášeny souběžně s on-line funkčností aplikace. Na IoTIM je rsyslog konfigurován tak, aby došlo k vložení všech logů v chronologicky správném pořadí. Posledním krokem přechodu do on-line stavu je notifikování pravidlového systému k jednorázovému provedení pravidel se spouštěčem `SUBSYSTEM_ONLINE`. V opačném stavu, kdy v předchozím cyklu server na zprávu odpověděl a nyní ne, jsou pravidlovému systému poslány pokyny k provedení pravidel se spouštěčem `SUBSYSTEM_OFFLINE`.

6.7 Databáze

Subsystém musí mít perzistentně uloženy objekty nodů, komponent, pravidel a konfigurací výpočetních modulů. Všechny operace s databází a (de)serializacemi jsou implementovány ve třídě `DB`. Pro datové objekty byla vybrána transakční souborová databáze LMDB (Lightning Memory-Mapped Database) [198] psaná v jazyce C, ukládající data jako pár klíč-hodnota. Nejedná se tedy o běžnou relační databázi, protože jsou podporovány pouze základní operace. Klíč-hodnotu lze uložit, přepsat, lze najít hodnotu podle klíče a odstranit pár dle klíče. Všechna data jsou uložena v jednom souboru s příponou mdb, který je platformově závislý. Protože databáze není realizována jako klient-server a nepotřebuje žádné aplikační vlákno, lze říci, že se v podstatě jedná o knihovnu. Interně jsou data organizována do B+ stromu s využitím přímého mapování na soubor. LMDB podporuje vnořené databáze a aplikace subsystému je využívá pro oddělení nodů, komponent, pravidel a výpočetních modulů. Pro manipulaci s databází musí být inicializována struktura prostředí `MDB_env`.

Protože aplikace využívá standardní C++ objekty, je nutné pro jejich perzistenci provést serializaci na proud bajtů. Tuto operaci lze pro jednoduché třídy provést manuálně, ovšem datové objekty subsystému obsahují mnoho textových řetězců, dynamicky alokovaných vektorů a primitivních proměnných různých rozsahů. Programování serializačních a deserializačních funkcí by tak zabralo desítky až stovky hodin. Místo toho je použita knihovna Cereal [199], sloužící právě k (de)serializaci C++ objektů. Knihovna využívá šablony funkcí a proto lze (de)serializovat pouze primitivní proměnné a kontejnery ze standardní

`std` knihovny, jako vector, map, string, array, pair atd. Mějme třídu `Foo`, která obsahuje číselnou proměnnou A a textový řetězec B:

Výpis 6.3: Ukázka hlavičkového souboru třídy Foo

```
#include <string>
class Foo {
public:
    void setA(int A);
    void setB(std::string B);

    int getA();
    std::string getB()

private:
    int A;
    std::string B;
}
```

Pro serializaci a deserializaci této třídy je nutné přidat hlavičkové metody `save` a `load`, v jejichž implementaci je volána přetížená funkce `ar()` s argumenty členských proměnných:

Výpis 6.4: Ukázka metod serializace a deserializace

```
class Foo {
public:
    template <class Archive>
    void save(Archive &ar) const
    {
        ar(A, B);
    }

    template <class Archive>
    void load(Archive &ar)
    {
        ar(A, B);
    }
...
```

Serializace třídy, tedy konverze instance třídy na pole bajtů, se provede následujícím kódem:

Výpis 6.5: Ukázka metody serializace

```
ostringstream ss(std::stringstream::binary);
{ //Scope serialization
    cereal::BinaryOutputArchive archive(ss);
    archive(type);
}
```

```
6
7 string content = ss.str();
8 const char *poleBajtu = content.c_str();
```

Výsledné pole bajtů je možné pak uložit do databáze. Nejprve je vytvořena transakce s ukazatelem na prostředí, poté je otevřena vhodná sub-databáze a předán ukazatel na transakce. Následně je nutné vytvořit dvě struktury `MDB_val`, první s informací o počtu bajtů a hodnoty klíče, a druhou s informací o počtu bajtů a ukazatelem na pole serializovaných dat. Pak lze provést transakci perzistence. U datových objektů je jako klíč využit buď identifikátor (komponenta, pravidlo, výpočetní modul), nebo adresa (nod).

Načtení dat z databáze a deserializace pak probíhá opačným způsobem. Je nutné znát klíč objektu, vytvořit transakci, otevřít databázi, otevřít kurzor, nastavit kurzor na adresu klíče, získat data a zrušit transakci. Deserializace se provede následujícím kódem, kde struktura `MDB_val data` obsahuje pole bajtů s původní třídou Foo.

Výpis 6.6: Ukázka metody deserializace

```
1 Foo foo; //Vytvoreni prazdne instance
2
3 stringstream ss;
4 //Konverze pole bajtu na stringstream
5 ss.write(static_cast<const char*>(data.mv_data), data.mv_size);
6 cereal::BinaryInputArchive archive(ss);
7 archive(foo); //Naplneni foo puvodnimi hodnotami
```

Třída DB obsahuje metody pro perzistenci objektu, smazání objektu dle klíče a načtení mapy všech objektů daného typu. Sub-databáze jsou vytvořeny během prvotní konfigurace subsystému. Při znovuspouštění aplikace si jádro od databáze vyžádá načtení kompletního modelu, nody předává delegátoru, pravidla pravidlovému modulu a konfiguraci výpočetních modulů třídě `EdgeComputer`.

6.8 Logování událostí

Při procesu implementace aplikace došlo k rozhodnutí, že agregace dat komponent není možné provádět jednoduchým a univerzálním způsobem a z tohoto důvodu nejsou implementovány. Bez metadat o nodech a komponentách se ze softwarového pohledu jedná pouze o objekty, ze kterých je třeba periodicky vyčítat data, případně je na pokyn serveru nebo pravidel nastavovat. Přepočet z nodu na komponenty a naopak je také prováděn pouze matematickým vzorcem. Proto by pro agregaci bylo nutné stanovit podmínky a agregační funkci, například aritmetický průměr ze všech hodnot za celý den. Existují i zařízení, jako tlačítka a spínače, kde aritmetický průměr bude z dlouhodobého hlediska vždy 0,5 a pro historická data je zajímavý více počet stisknutí, případně v jakou dobu stisky nastaly. Takové výpočty by znamenaly i vytvoření pokročilého časovače, který by data komponent bral na pozadí z databáze a prováděl agregace. Tyto operace by dále způsobily snížení životnosti SD karty. Také by bylo obtížné interpretovat agregovaná data komponent s výsledky pravidel a informačními či chybovými zprávami. Události jsou proto jednoduše ukládány na SD kartu a pokud není dostatek místa, jsou ty nejstarší odstraněny.

Třída `Core` implementuje následujících 5 metod bez návratové hodnoty:

Výpis 6.7: Metody pro serializaci událostí

```
1 void logError(Error err, u64 id, std::string &description, u64 tstamp);
2 void logInfo(Info info, s64 opt, std::string &description, u64 tstamp);
3 void logComponentData(const Component &component);
4 void logRuleEvaluated(const Component &component, u32 ruleId, u64 tstamp);
5 void logBatteryState(u16 nodeAddress, u8 batteryLevel, u64 tstamp);
```

Pokud se subsystém nachází v on-line stavu, je volána patřičná metoda ve třídě `Core`, sestaven FlatBuffer a zpráva je odeslána k IoTIM přes MQTT broker. V off-line stavu je využívána technologie rsyslog, tedy logovací démon systému Linux. Ten je přes prvotní konfigurační proces nastaven jako klient a připojuje se k rsyslog serveru na IoTIM. Pokud dojde k selhání MQTT brokera, jsou data okamžitě přes TCP spojení rsyslogu odesílána na IoTIM. V případě, že se jedná o fyzický výpadek spojení, ukládá rsyslog na subsystému události do souborů. Dochází-li místo v úložišti, jsou odstraňovány chronologicky nejstarší záznamy. Po znovunavázání spojení dojde k dávkovému přenosu logů na IoTIM.

6.9 Sestavení aplikace

Aplikace využívá ke svému sestavení systém CMake. Tradičním postupem sestavení programů psaných v jazycích C a C++ na operačním systému Linux je pomocí souboru `Makefile`. Vytvoření takového souboru je pro programy s více než jednotkami C souborů komplikované, protože je nutné ošetřit specifika různých distribucí Linuxu, jako je umístění knihoven, verze a typ použitého kompilátoru a typ balíčkovacího systému distribuce.

CMake odstiňuje vývojáře od nízkoúrovňových odlišnosti jednotlivých distribucí i operačních systémů. Projekt využívající CMake musí mít soubor `CMakeLists.txt`, kde jsou specifikovány potřebné knihovny, kompilační cíle, názvy projektů a podpůrné operace. Lze prohledávat zdrojové kódy a hledat určené definice, dokonce lze sestavení projektu větvit na základě podmínek. CMake pak zavoláním příkazu `cmake ..` zjistí přítomnost kompatibilního kompilátoru a pokud je vše v pořádku, vytvoří `Makefile` pro cílový systém. Pokud není přítomna požadovaná knihovna, je vypsána chybová hláška spolu s doporučením, jak problém opravit. Dále kompilace probíhá standardním voláním `make`. Pro instalaci sestavené aplikace lze zavolat `sudo make install`, nebo využít plugin CMake pro vytvoření DEB balíčku a ten poté nainstalovat. V distribuci DEB balíčku je obsažena služba pro systémový démon `systemd`. Subsystém je po nainstalování možné spustit jako službu na pozadí stejnými standardizovanými příkazy, jako další serverové služby běžně využívané v operačním systému GNU/Linux.

V případě subsystému CMake při vytváření `Makefile` souboru analyzuje hlavičkový soubor `src/config/config.h` a dle výskytu definic upravuje výsledné sestavení. Lze tak vytvořit odlišná sestavení aplikace s podporou různých technologií a různým chováním. Vybrané definice budou nyní popsány.

ENABLE_RS485

Zakomentováním této definice v hlavičkovém souboru dojde k ignorování metod třídy `Rs485Planner`. Pokud server bude vyžadovat CRUD operaci s RS-485 nody, bude serveru vrácena chybová hláška `RS485_NOT_SUPPORTED`.

ENABLE_BLE

Ignorace této definice vynechá kompilaci třídy `BlePlanner`. Pokud server bude vyžadovat CRUD

operaci s Bluetooth Low Energy nody, serveru obdrží chybovou hlášku `BLE_NOT_SUPPORTED`. Bez definice není pro sestavení vyžadována knihovna `tinyb`.

ENABLE_ZWAVE
Zakomentováním této definice dojde k ignorování metod třídy `ZwavePlanner`. Pokud server bude vyžadovat CRUD operaci se Z-Wave zařízeními, serveru je vrácena chybová hláška `ZWAVE_NOT_SUPPORTED`. Knihovna `openzwave` pak není vyžadována pro korektní sestavení.

ENABLE_WEBSERVER
Tato definice umožní vypnutí ladícího webserveru. Ve výchozím stavu, který stále předpokládá experimentální nasazení řešení, je tato volba povolena. Bez této definice není vyžadována přítomnost knihovny `Pistache` v systému.

ENABLE_LEDS, ENABLE_BUTTONS a ENABLE_TEMPERATURE
Tyto definice umožňují využít hardware specifický pro Raspberry Pi. Lze vypnout nebo zapnout podporu LED, tlačítek a I^2C teploměru. Integrované prvky se pak hlásí serveru jako nody s připojením `EMBEDDED`, a není možné je ze subsystému odstranit. Odstraněním definice `ENABLE_TEMPERATURE` pak přestane aplikace vyžadovat vývojový balíček `libi2c-dev`.

ENABLE_OFFLINE_DB_LOG
Pokud byl subsystém zamýšlen jako samostatná automatizační jednotka pro malý objekt, lze zakomentováním definice `ENABLE_OFFLINE_DB_LOG` vypnout logování událostí. Pravidlový systém zůstane funkční, ale žádné události se nebudou ukládat do systémového logu. Díky tomu není úložiště zatěžováno I/O operacemi.

ENABLE_CLOCK_ADJUSTMENTS
Definice `ENABLE_CLOCK_ADJUSTMENTS` sestaví aplikaci s mechanismem pro nastavení času dle výsledků `PingPongFB` zpráv mezi subsystémem a serverem.

6.10 Aktualizační modul

Nová verze aplikace subsystému může být vydána z důvodu přidané funkcionality, optimalizace nebo opravy chyby. Aktualizaci programu lze provést stažením nového balíčku, reinstalací a restartováním služby subsystému pomocí příkazu `sudo systemctl restart subsystem.service`.

Aplikace obsahuje mechanismus pro vlastní aktualizaci na novou verzi, bez nutnosti přímé interakce se subsystémem a aktualizace balíčků. Využívají se zprávy popsané v kapitole 4.4.3. Server odesílá subsystému zprávu `FWUpdateAnnounceFB` s URL adresou a kontrolním součtem nové verze aplikace. Třída `Connector` tuto informací předává jádru, které ji předává třídě `Updater` v aktualizačním modulu. Ta vytváří nové vlákno a pomocí knihovny `curl` stahuje soubor pod názvem `subsystem-new`. V dalším kroku se pomocí GNU programu `md5sum` počítá kontrolní součet staženého programu. Následně je soubor kontrolován vůči hlavičce ELF, zdali se skutečně jedná o spustitelný binární soubor pro operační systém GNU/Linux. Poté se porovnává architektura s již existujícím souborem aplikace `subsystem`. V případě úspěšného i neúspěšného stavu je serveru zpětně odeslána zpráva `FWUpdateConfirmationFB`. Ve zprávě je přítomna enumerace `UpdateErrorFB` s následujícími stavy:

- `NONE` – žádná chyba nenastala a subsystém je připraven k aktualizaci,

- `CURL_PROBLEM` – nezdařilo se inicializovat knihovnu `curl`,
- `INVALID_URL` – poskytnutá adresa není platná nebo dosažitelná,
- `MD5_CHECKSUM` – nesouhlasí kontrolní součty,
- `NOT_ELF_BINARY` - stažený soubor není binární spustitelný,
- `WRONG_ARCHITECTURE` – neodpovídající architektura,
- `DISK_SPACE` – nedostatek místa na úložišti.

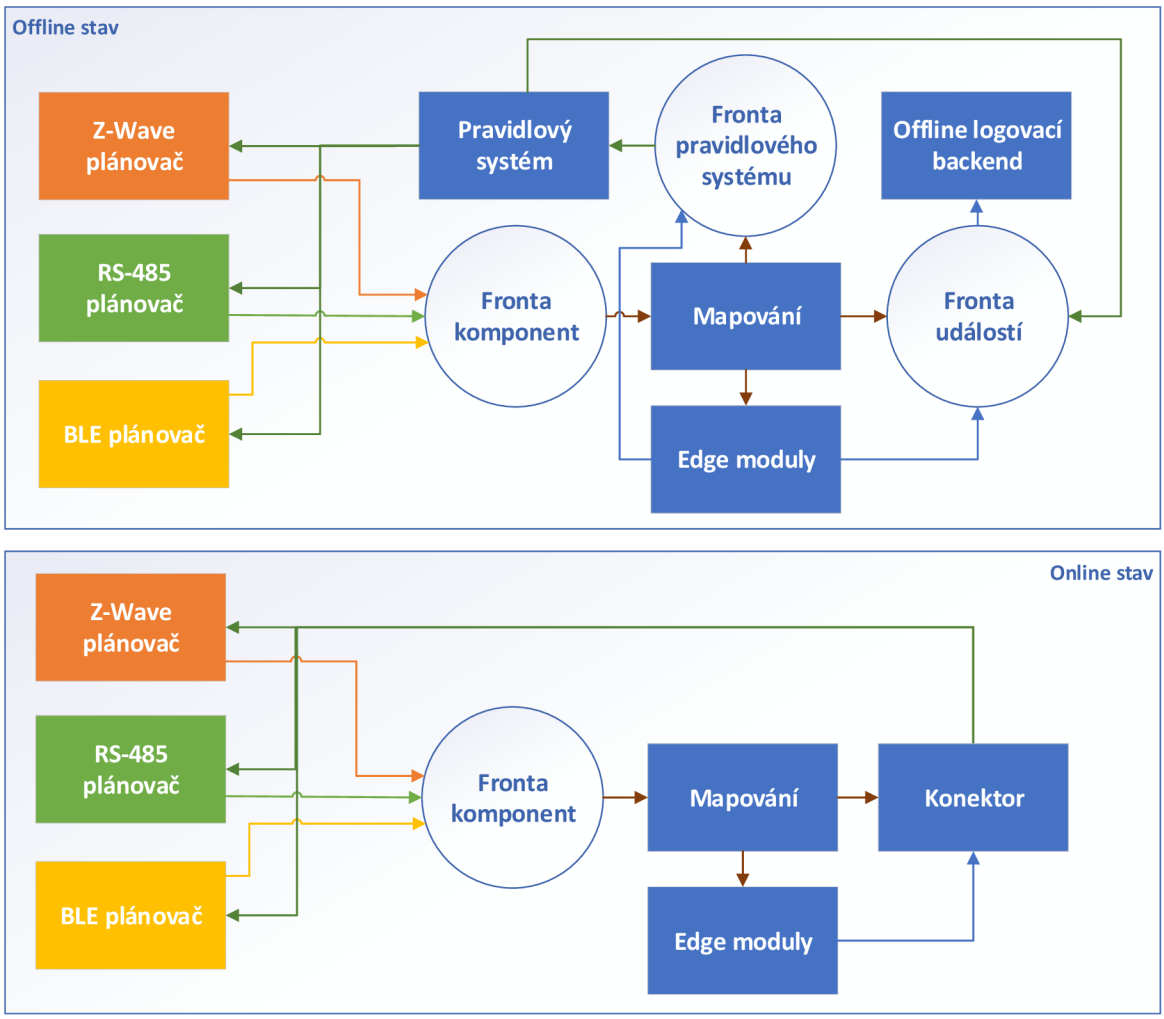
IoTIM může po přijetí zprávy odeslat zprávu `FWUpdateStartFB` pro zahájení aktualizace. Pokud nebylo v předchozím kroku dosaženo úspěchu, je zpráva jádrem ignorována. V opačné situaci volá jádro metodu `performUpdate` třídy `Updater`. Ta přejmenuje původní soubor `subsystem` na `subsystem-old` a nový soubor `subsystem-new` na `subsystem`. Následně je současný proces aplikace zdvojen voláním `fork()`. Jádru rodičovského procesu je dokončení zdvojení oznámeno voláním listeneru `updaterFinished`, s booleovským argumentem. Pokud je pravdivý, jádro ví, že je rodičem, uzavírá souborové deskriptory a ukončuje svůj proces s návratovou hodnotou `EXIT_SUCCESS`. Dceřinný proces pouze uzavírá své deskriptory a vrací se zpět do původní metody. Ta čeká jednu sekundu, aby mohl být korektně ukončen rodičovský proces. Poté je již voláním `execlp` nahrazen původním proces novým procesem, který má identický název, ovšem do operační paměti se kopíruje z nové verze binárního souboru aplikace.

6.11 Paralelismus a balancování zátěže

Z předchozích podkapitol je zřejmé, že aplikace využívá možnosti paralelního běhu a vláken. Tento přístup je v jádře aplikace dále využit k zajištění co nejnižší latence při běhu aplikace jak v on-line, tak v off-line režimu. Využití fronty pro mapování nodu pro čtení na komponenty je již popsáno v podkapitole 7.1.2. Pokud je subsystém v běžném on-line režimu, je ve smyčce metody `dataProcessLoop` získán objekt nodu, proveden matematický přepočet na komponentu a je-li výsledek odlišný od výsledku předchozího, jsou data komponenty skrze třídu `Connector` serializována do FlatBuffer zprávy a odeslána serveru a souběžně přes třídu `EdgeComputer` všem Edge modulům vyžadujícím data této komponenty.

V off-line stavu je situace komplikovanější, protože nová data komponenty je nutné serializovat do vektoru událostí a zároveň její nový stav aktualizovat v pravidlovém systému a provést výpočet všech pravidel se spouštěčem komponenty, kde je tato komponenta zahrnutá. Pokud je výsledek pravidla odlišný a výsledkem pravidla jsou opět komponenty, je nutné je mapovat zpět na nod a změnit jejich stav. Tyto operace trvají více než milisekundu a mohlo by dojít k situaci, kdy se fronta s nody plní rychleji, než je možné data synchronně zpracovávat. V aplikaci jsou proto využita další dvě vlákna, v on-line stavu neaktivní a blokována. Po matematickém přepočtu je objekt komponenty vložen do dvou struktur. Struktura `LogEvent` slouží k perzistenci všech pěti typů událostí (data komponenty, výsledek pravidla, chyba, informace, stav baterie) a nastavením enumerace `LogType` je specifikováno, o jakou událost se jedná. V případě nového stavu komponenty je enumerace nastavena na hodnotu `COMPONENT_DATA` a ve struktuře musí být nastaveny proměnné s časovou známkou, adresou nodu, číslem kanálu a vektorem dat. Struktura je následně vložena na konec fronty `logQueue`. Struktury jsou pak sekvenčně získávány z fronty v nekonečné smyčce metody `logLoop`, konvertovány na JSON a posílány do syslogu.

Ihned po vložení struktury `LogEvent` do fronty je dále sestavena struktura `EngineCommand` pro pravidlový systém. Vnořená enumerace `EngineAction` specifikuje požadavek pro pravidlový systém. Může se jednat o nový stav komponenty, tiknutí nebo CRUD operace na komponentami a pravidly. Opět je využita fronta a vlastní vlákno pro obsluhu pravidlového systému. Pokud je výsledek pravidla odlišný, je přes frontu `logQueue` událost logována. Přepočet komponent zpět na nod a upozornění plánovače probíhá synchronní sekvencí volání. Vztah front a bloků aplikace je znázorněn na obrázku 6.5, kde jsou odlišná vlákna zvýrazněna barvami šipek.

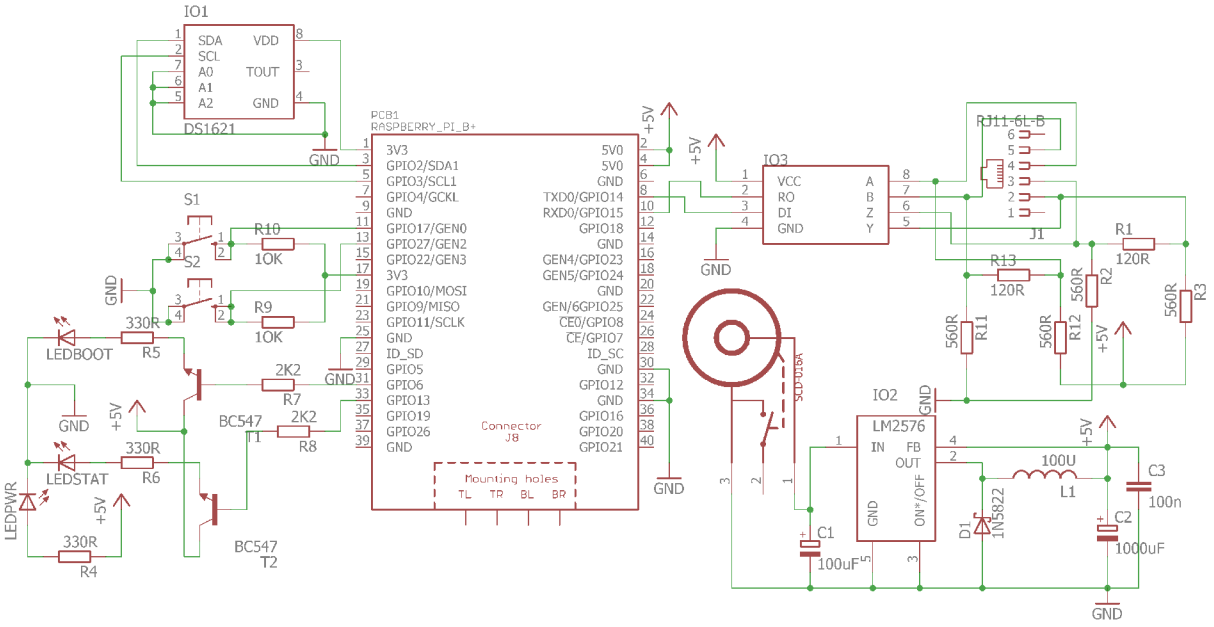


Obrázek 6.5: Vztah vláken a bloků aplikace [autor]

Procesy výpočtu pravidla jsou výpočetně náročnější než mapování nodů na komponenty. V případě dlouhodobé maximální zátěže subsystému tak dojde k situaci, kdy se fronty pro pravidla plní rychleji, něž je možné požadavky zpracovávat. Proto je při každém mapování nodu pro čtení kontrolován počet položek ve frontě. Pokud je alespoň v jedné frontě více než 500 čekajících objektů, je přes delegátor zpomalen plánovač kabelových nodů. Při každém volání je zvýšen čas uspání plánovače o 5 mikrosekund. Zpomalování je prováděno v každém průchodu smyčky `dataProcessLoop`, dokud nedojde ke snížení zaplnění všech front pod 500 objektů. Pokud všechny fronty obsahují méně jak 10 objektů, je naopak rychlost plánovače zrychlena. Tento mechanismus umožňuje efektivně balancovat zátěž aplikace. Při

nenáročné automatizaci pracuje plánovač RS-485 nodů s minimální latencí. Limit 500 zpráv umožňuje výskyt krátkodobých extrému. Při dlouhodobém extrému dojde ke snížení frekvence plánovače a zvýšení latence. Po skončení tohoto extrému dojde postupně k návratu do výchozího stavu.

6.12 Hardware subsystému



Obrázek 6.6: Schéma zapojení subsystému [autor]

Pro Raspberry Pi (druhé generace a vyšší) byl navržen speciální hardware, jehož výsledný plošný spoj se montuje na čtyři 22 mm vysoké distanční můstky M2,5 a je spojen s Raspberry přes GPIO pinovou lištu. Schéma zapojení je znázorněno na obrázku 6.6, výkres plošného spoje je pak v příloze A na obrázku 5 a fotografie výsledné desky na obrázku 6. Vyšší mezera mezi plošným spojem a Raspberry byla zvolena, aby nedošlo ke zhoršení odvodu tepla z komponent Raspberry Pi. Hardware je realizován jako shield, který se spojí s Raspberry Pi GPIO paticí a je upevněn distančními můstky. Spoj obsahuje řadič sběrnice RS-485 (Maxim Integrated MAX488), teplotní I^2C čidlo DS1621 pro monitoring teploty subsystému, dvě uživatelsky nastavitelná tlačítka a LED spínané NPN tranzistory (z důvodu proudového odběru GPIO pinu). Obvod je napájen přes standardní 2,5 mm Jack konektor s napájecím napětím 6–45 V. Pro stabilizaci napětí na 5 V je využit spínaný regulátor LM2576. Tento regulátor je k dispozici ve dvou variantách, LM2576T-5 a LM2576HVT-05, kdy druhý model nabízí maximální vstupní napětí až 63 V. Vzhledem k účinnosti 77 % resp 75 % a snaze udržet teploty subsystému v přijatelných mezích není doporučeno používat zdroj s výstupní napětím vyšším než 12 V.

7

Ukázková implementace nodů

Nody jsou nejnižší vrstvou nového řešení a jedná se koncové hardwarové prvky vybavené mikrokontrolérem pro obsluhu různých senzorů či aktuátorů. Využití kabelových zařízení je zamýšleno tam, kde není přijatelný výpadek komunikace, jelikož jakákoliv bezdrátová technologie může být rušena, případně podle využitých konstrukčních materiálů v prostředí způsobovat poklesy kvality spojení.

Nody s technologií Bluetooth Low Energy pak poskytují alternativu ke kabelovým nodům bez nutnosti instalace kabeláže, vyjma napájení.

Z-Wave zařízení jsou komerční produkty různých výrobců s jasně definovanou sadou funkcionalit a vlastností.

7.1 RS-485

V současné době se těší 32-bitové mikrokontroléry, zejména architektury ARM, stále vzrůstající popularitě, ať již z důvodů hardwarově akcelerovaných výpočtů v plovoucí desetinné čárce, jednodušší organizace paměti, rozsáhlé instrukční sadě nebo množství zabudovaných periferních modulů. I přes tento trend byly jako mikrokontroléry vybrány modely z 8-bitové řady 16F společnosti Microchip. Prvním důvodem je náklonnost společnosti k prototypovému vývoji, protože všechny mikrokontroléry do 40 pinů včetně jsou k dispozici v pouzdře DIP, což umožňuje vývoj s využitím prototypovacích desek, nezávislost na SMD součástkách a tedy možnosti osazovat plošné spoje standardní hrotovou nebo transformátorovou pájkou. Druhým důvodem je, že nod vyžaduje pouze minimální prostředky na mikrokontrolér, konkrétně ADC převodník pro generování náhodného čísla, jeden UART pro komunikaci přes RS-485, dva bajty non-volatilní paměti pro uložení adresy, jednotky kilobajtů programové paměti a několik desítek bajtů datové paměti. Třetím důvodem je, že Microchip neustále představuje nové modely a nedochází ke stagnaci. Naopak nejnovější modely jako PIC16F15345 obsahují technologie dříve dostupné pouze u nejvybavenějších 32-bitových mikrokontrolérů, jako podpora mapování periférií, kdy si vývojář může téměř libovolně zvolit, která periférie bude přítomna na kterém pinu. Je tak možné změnit například výchozí umístění pinů SDA a SCL pro I^2C sběrnici. Tento přístup výrazně zjednodušuje následný návrh plošného spoje, kdy se periférie mapují podle rozvržení součástek.

7.1.1 Standardní nod

Prvním vytvořeným hardwarem je standardní nod, jehož cílem je obsloužit co největší množství různých senzorů a aktorů. To jakým způsobem se nod chová, lze ovlivnit typem firmwaru, který je do

mikrokontroléru nahrán programátorem PicKit.

Hardware nodu

Schéma zapojení standardního nodu je znázorněno na obrázku 7.1, v příloze A je pak výkres plošného spoje na obrázku 7 a fotografie osazeného hardwaru na obrázku 8. Výsledná deska má rozměry 62×62 mm. Centrálním prvkem je patice DIL20, do které se vkládají 20-pinové mikrokontroléry. Obvod je napájen souosým konektorem Jack 2,5 mm a vyžaduje stabilizované vstupní napětí 5 V. Vstupní napětí je přivedeno k mikrokontroléru a dvěma řadičům MAX485 pro převod diferenčních signálů RS-485 na TTL úrovně. Řadič RXMAX slouží k příjmu paketů od subsystému, jeho piny RE a DE jsou spojeny se zemí a řadič tak přijímá data trvale. Příjem lze vypnout programově, ve firmwaru nodu. Řadič TXMAX má trvale vypnutí režim přijímání a odeslání odpovědí subsystému zprostředkovává pin RB6 mikrokontroléru. K oběma řadičům je možné připojit terminační rezistory 120 Ω, pokud se jedná o poslední nod na sběrnici. Připojení se provede aplikací jumperu na piny TERM1 a TERM2.

Paralelně k pinům RA4 a RA5 je připojen krystal Q1 o frekvenci 18,432 MHz. V závislosti na použitém mikrokontroléru může udávat hodinový kmitočet, případně lze využít zabudovaný RC oscilátor. Na pin RA3 je připojeno resetovací tlačítko RST, sloužící ke smazání adresy nodu a skoku na první programovou instrukci. Piny RC4 a RC5 mají připojeny diody CLED a SLED. Dioda CLED krátkým bliknutím indikuje poruchu v komunikaci. Dioda SLED pomalu bliká, pokud nemá nod ještě přidělenou adresu. Po přidělení adresy neustále svítí, nebo rychle bliká pokud byla nodu poslána zpráva `IDENTIFY_ON`.

Pro programování mikrokontroléru slouží pinová lišta ICSP, připojená na programovací piny RA0, RA1. Ty se také používají při procesu prvotní adresace ke generování náhodného čísla pro časování odpovědi. Pin pro resetování nodu je paralelně přiveden také na ICSP pin, aby mohl programátor přepnout mikrokontrolér do režimu flashování.

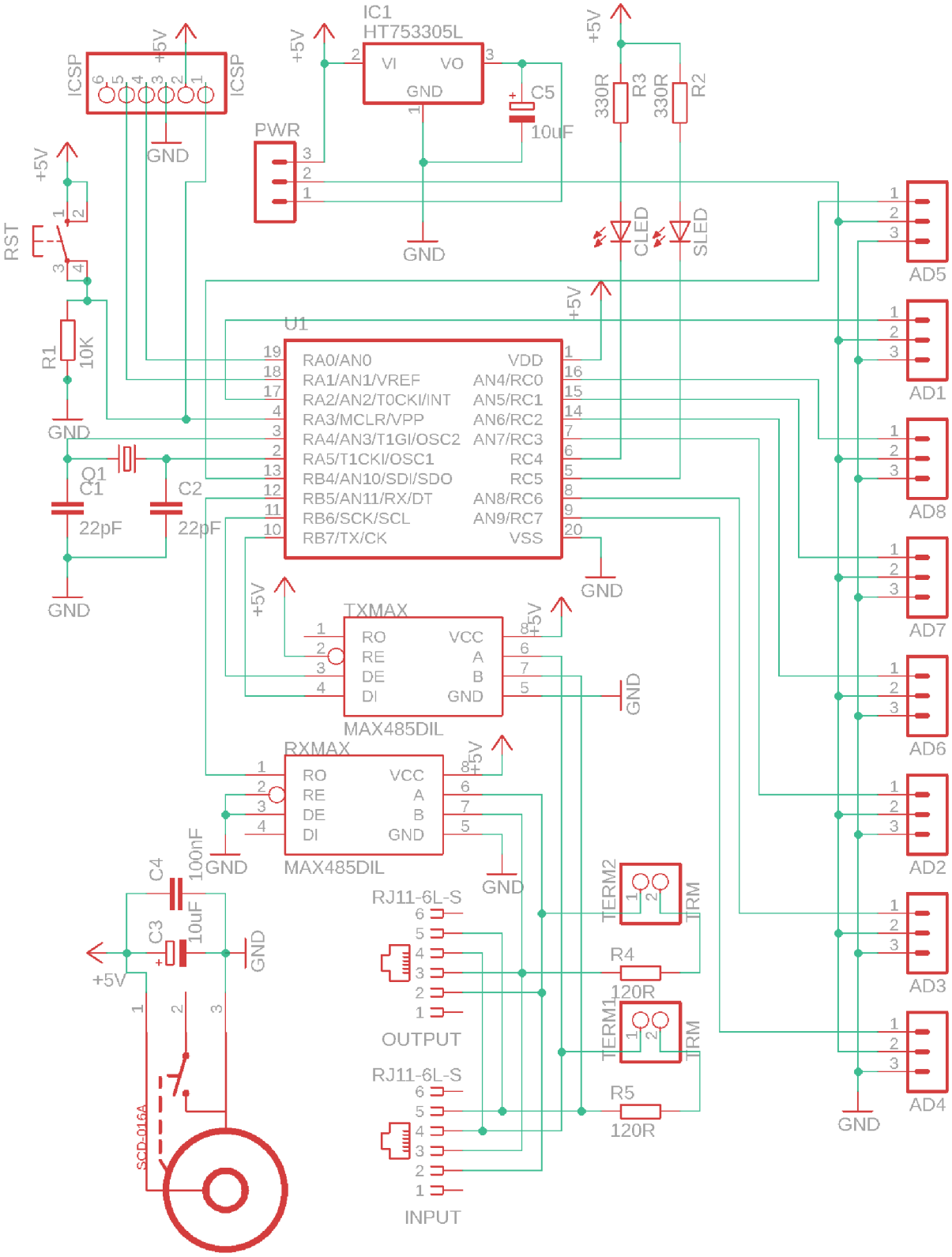
Zbylé piny mikrokontroléru jsou přivedeny na svorky AD1 až AD8. Každá svorka má dále přivedenou zem a buď 5 V přímo z napájecího konektoru, nebo výstup 3,3 V ze stabilizátoru HT753305L. K přepínání napájení slouží pin lišta PWR, cílové napětí se volí jumperem[1].

Standardní nod tedy využívá 20-pinové 8-bitové mikropočítače Microchip, střední řady 16F. Všechny modely v této řadě mají stejně pojmenované IO piny i jejich základní mapování. Pokud se vybraný mikrokontrolér ukáže během vývoje jako nedostatečný, lze jej nahradit vyšší řadou. Přechod samozřejmě musí zahrnovat nutné úpravy jako odlišné konfigurace registrů oscilátoru. Ty se mohou jmenovat jinak, případně mít více konfiguračních bitů v závislosti na vybavenosti mikrokontroléru. V současné době je k dispozici 19 různých mikrokontroléru kompatibilních s požadavky na kabelový nod. Standardní nod využívá tři modely, PIC16F690, PIC16F1579 a PIC16F1543. Další specifické nody pak navíc 14 pinový model PIC16F15324.

Firmware nodu

Chování standardního nodu lze ovlivnit tím, jaký firmware je nahrán v jeho mikrokontroléru. Všechny firmwary využívají hlavičkový soubor `common.h`, jehož metody jsou implementovány v C souboru `common.c`. Hlavičkový soubor `common.h` pak vyžaduje, aby každý firmware měl ve svém adresáři přítomný hlavičkový soubor `project_specific.h` s následujícím obsahem:

[1] Jumper ovlivňuje pouze napětí na svorkách. Mikrokontrolér pracuje vždy v napěťové logice 5 V. Budoucí revize standardního nodu bude vybavena přepínatelnými level shiftery pro individuální piny.



Obrázek 7.1: Výkres standardního nodu [autor]

Výpis 7.1: Soubor project_specific.h

```
#define DATA_WIDTH 16
#define DATA_COUNT 1

#define ACCESS_TYPE READ  //Nabyva hodnoty READ nebo WRITE
#define LATENCY DELAYED //Nabyva hodnoty NOW nebo DELAYED
#define MIN_DELAY 1

#define FW_VERSION 1
extern const code unsigned char fw_description[];

#define CHANNEL_COUNT 2

#define PIC16F690
```

Pole `fw_description` musí být definováno v souboru `project_specific.c` s maximální délkou 30 znaků. Delší popisek je zkrácen.

V souboru `common.c` jsou pak implementovány metody společné pro všechny firmwary, jako proces prvotní adresace, inicializace časovače, inicializace UART, a metody pro běžnou komunikaci. Dále pak obsluha resetovacího tlačítka, správa FLASH nebo EEPROM paměti (dle modelu mikrokontroléru), algoritmy pro výpočet a verifikaci CRC8 a detekci nevalidního paketu. Pokud je přijatý paket vyhodnocen jako validní a adresa odpovídá danému nodu, je nejprve zjištěno, zdali se nejedná o zprávu `IDENTIFY_ON` nebo `IDENTIFY_OFF` sloužící k zapnutí nebo vypnutí blikání identifikační SLED nebo o zprávu `PROPERTIES_REQUEST` pro vyžádání charakteristik. Pokud ne a přijatá zpráva je buď `REQUEST`, `SET` nebo `MEASURE`, je zavolána metoda `handle_data_packets`, která musí být implementována dle konkrétního firmwaru. Veškerý vývoj mikrokontrolérů probíhá ve vývojovém prostředí MikroC společnosti Mikroelektronika a pro jejich programování se využívá programátor PicKit verze 3. Nyní budou popsány vyvinuté firmwary využívající hardwaru standardního nodu.

Firmware digital_input_fast

Jedná se firmware pro obsluhu digitálních (respektive binárních) vstupů, jako jsou tlačítka, pohybová čidla, přepínače a magnetické kontakty. Podle definice `CHANNEL_COUNT` vyčítá firmware 1 až 8 svorek nodu. Protože firmware je rychlého typu, stav pinů je čten a odeslán zpět subsystému ihned po obdržení zprávy `REQUEST`.

Firmware digital_input_delayed

Firmware pracuje obdobně jako `digital_input_fast` s tím rozdílem, že vyčtení stavu pinů probíhá při přijetí zprávy `MEASURE` a po přijetí zprávy `REQUEST` se odesílá již před tím vyčtený stav. Charakteristika firmware je také obdobná jako u `digital_input_fast`, ale minimální prodleva mezi čtením stavů je 100 ms. Jeho využití je tam, kde není třeba stavy vyčítat s vysokou frekvencí.

Firmware digital_output

S tímto firmwarem se hardware nodu chová jako digitální výstupy, kde definice `CHANNEL_COUNT` ovlivňuje počet svorek, které se podle přijaté zprávy `SET` nastavují. Po nastavení firmware odpovídá subsystému zpět zprávou `GET`.

Firmware analog_input_fast

Cílem tohoto firmware je podle definovaného počtu kanálů vyčítat napětí na jednotlivých svorkách přes zabudovaný ADC modul mikrokontroléru 16F690. Rozlišení ADC modulu je 10 bitů, ovšem firmare se subsystému ohlašuje šírkou dat 16 bitů a jednou složkou. Je to z důvodu možné konstrukce nodu s 16 bitovým ADC pro přesnější měření. Tento firmware je využitelný pro různé analogové senzory jako jsou teploměry, fotoodpory, vlhkoměry, mikrofonní moduly a ovládací prvky jako tahové a rotační potenciometry. Díky možnostem matematického převodu nodu na komponenty lze kombinovat analogové i binární vstupy. Převodovým výrazem `rint(a / 512)` lze docílit toho, aby binární vstup vracel korektně buď hodnotu 1, nebo 0.

Firmware analog_input_delayed

Jedná se o podobný firmware jako `analog_input_fast` s rozdílem rychlosti měření. Napěťové stavy jsou měřeny po přijetí zprávy `MEASURE` a po přijetí zprávy `REQUEST` je pouze vrácen dříve naměřený výsledek. Firmware tedy využívá pomalého typu a je vhodný k měření senzorů, kde je vzorkovací frekvence 10 Hz dostatečná. Také lze kombinovat s binárními vstupy.

Firmware ds18b20

DS18B20 je teplotní čidlo komunikující skrze sběrnici 1-Wire. Díky teplotnímu rozsahu −55 °C až +125 °C je čidlo vhodné i pro venkovní měření, případně měření kotlů a tepelných čerpadel. Přesnost měření je ±0,5 °C a data lze vyčítat s rozlišením 9 až 12 bitů. Implementovaný firmware nastavuje automaticky nejvyšší přesnost 12 bitů. Na sběrnici 1-Wire lze provozovat desítky identických senzorů v sérii, díky zabudovaným unikátním adresám, ovšem tento firmware senzory v sérii nepodporuje. Je ale možné na každou svorku nodu připojit jedno čidlo a jedním nodem jich obsluhovat až 8. Pokud libovolný senzor neodpovídá na pokyn k naměření dat, je na tomto kanálu nodu vrácena hodnota 32 768, indikující chybový stav. Doporučený převod dat nodu na komponentu ve stupních Celsia se provede výrazem
`a == 32768 ? 0/0 : a < 32768 ? a/16 : (a - 65535)/16`.

Firmware dht22

Tento firmware slouží pro získávání dat z čidla teploty a vlhkosti DHT22 (alternativní označení AM2302). Čidlo využívá vlastní jednovodičový komunikační protokol a celá komunikace proto nespoléhá na žádnou knihovnu a je implementována přímo v kódu firmwaru. Měřený rozsah teplot je −40 °C až +80 °C, při přesnosti ±0,2 °C. Relativní vlhkost je měřena v plném rozsahu 0 až 100 %, při maximální toleranci 5 %. Podle definovaného počtu kanálů je možné získávat data z až osmi senzorů. Po přijetí zprávy `MEASURE` proběhne vyčtení dat ze senzorů a pokud některý neodpoví, je místo teploty (nultý kanál) a vlhkosti (první kanál) vráceno číslo 65535. Firmware čte teplotu jako surová celočíselná data v 15-bitovém doplňkovém kódu. Pro přepočet na teplotu a vlhkost s přesností jednoho desetinného čísla je nutné aplikovat výraz
`a == 65535 ? 0/0 : a < 32768 ? a/10 : (a-32767)/10`,
která provádí korektní přepočet i pro hodnoty pod bodem mrazu.

Firmware bh1750

Pro měření intenzity osvětlení lze využít kombinaci fotorezistoru s pull-up nebo pull-down rezistorem. Ovšem měřený stav je pak pouze napětí a různé fotorezistory mají různé charakteristiky (přesnost a linearitu). Pro převod na luxy, jednotky intenzity osvětlení je nutné naměřit několik hodnot napětí a dle hodnot změřených kalibrovaným luxmetrem lze stanovit převodní výraz a ten aplikovat mezi nod a komponentu. Druhou možností je využít firmware `bh1750` pro obsluhu stejnojmenného senzoru od

společnosti Rohm Semiconductor. Jedná se o kalibrovaný senzor intenzity osvětlení komunikující sběrnicí I^2C.

Tento firmware je nutné zkombinovat s mikrokontrolérem PIC16F15345, protože původní PIC16F690 neumožňuje přemapovaní periferních pinů, a pin RB6 hodin I^2C se využívá pro řízení odpovědí subsystému. Firmware pak mapuje piny I^2C na piny RC0 (SDA) a RC1 (SCL), které na standardním nodu odpovídají svorkám AD8 a AD7. Samotný senzor je realizován jako SMD součástka a lze využít široce dostupný modul přidávající pull-up rezistory, konverzi napěťových úrovní a výstupními piny s roztečí 2,54 mm.

Firmware nastavuje senzor do režimu nejpomalejšího měření s nejvyšší přesností $\pm$0,5 lux. Data jsou vyčítána na celé luxy a v případě problému se senzorem je odesíláno číslo 6 535. Převod na komponentu je doporučen výrazem `a != 65535 ? a : 0/0`. Firmware oznamuje subsystému jednu složku o šířce 16 bitů s periodou měření 1.5 s.

Firmware tcs34725

Senzor TCS34725 obsahuje barevné filtry a měří poměr červené, zelené a modré složky obsažené ve světle dopadající na senzor. Lze ho využít ke zjišťování původu světelného zdroje (přirozený nebo umělý) a také k jeho intenzitě, i když se nejedná přímo o luxy. Protože senzor komunikuje přes sběrnici I^2C, je nutné využít mikrokontrolér PIC16F15345. Ohlašuje se se subsystému jedním kanálem se čtyřmi složkami o šířce 16 bitů, kde data jsou úroveň světla, červená složka, zelená složka a modrá složka světla. V případě problémů v komunikaci se senzorem je každá složka nastavena na hodnotu 65 535.

Firmware bme280

Bosh BME280 je populární environmentální senzor pro měření teploty, relativní vlhkosti a barometrického tlaku. Díky rozměrům 2,5$\times$2,5$\times$0,9 mm je často používán i v chytrých telefonech jako výškoměr. Komunikuje sběrnicí I^2C, tudíž je pro tento firmware vyžadován mikrokontrolér PIC16F15345. Subsystému se identifikuje třemi složkami s šířkou 24 bitů a čtením každé 2 sekundy. Senzor BME280 vyžaduje vyčtení 18 kalibračních konstant, které jsou unikátní pro každý senzor. Firmware pak dle doporučení datasheetu senzoru nastavuje nejvyšší přesnost měření a po čtení surových hodnot ze zabudovaného AD převodníku senzoru aplikuje konstanty pro získání skutečných environmentálních hodnot.

Firmware stepper_motor

Firmware `stepper_motor` lze využít pro ovládání jednoho nebo dvou unipolárních krokových motorů. Pro přijetí zprávy `SET` provede firmware potřebný počet kroků. Motorem lze otáčet oběma směry, protože firmware interpretuje přijaté 16 bitové číslo jako znaménkové v doplňkovém kódu. Protože krokové motory často vyžadují napájecí napětí alespoň 12 V, je nutné k jejich ovládání využít tranzistory nebo integrovaný obvod s bipolárním tranzistorovým polem, jako například ULN2803.

Firmware wattmeter

Tento firmware umožňuje na svorku AD1 připojit přes pomocný obvod neinvazivní proudovou sondu SCT-013-015 pro měření proudů až 15 A střídavého napětí. Firmware pro přijetí zprávy `MEASURE` naměří na vstupu svorky 2 000 hodnot, ze kterých po centrování dat vypočítává lokální minima a maxima pro zjištění proudu protékajícího sondou. Subsystému pak na zprávu `REQUEST` vrací aktuální spotřebu ve wattech vynásobením konstantou 230. Nod lze vyčítat subsystémem každých 1,5 s a data jsou jedné složky s šířkou 16 bitů.

7.1.2 Nod s relé

Cílem tohoto hardwaru je umožnit spínání silových prvků. Schéma zapojení nodu je znázorněno na obrázku 1, výkres plošného spoje na obrázku 7 a fotografie na obrázku 8 v příloze. Obsahuje dvě bipolární relé se stejnosměrným napájením 5 V. Maximální spínaný proud vybraného relé může dosahovat až 10 A při 250 V střídavého napětí nebo 30 V stejnosměrného napětí. Protože spínací proud cívky činí 70 mA, což je více než maximální možný odběr na pinu mikrokontroléru, jsou cívky spínány bipolárním NPN tranzistorem. Paralelně s kontakty cívky jsou umístěný indikační diody LED1 a LED2. Hardware využívá 14 pinového mikrokontroléru PIC16F15324, kterému dodává kmitočet krystal Q1 o frekvenci 18,432 MHz. Pro použití nodu je nutné využít firmware `relay`. Tento firmware se chová identicky jako firmware `digital_output`, ovšem je přizpůsoben mikrokontroléru `PIC16F15324`.

7.1.3 Nod pro stmívání RGBW LED pásků

Možnost nejenom spínání, ale i plynulé regulace intenzity svitu (stmívání) LED pásků je velmi častý požadavek v instalacích inteligentních domů. Místo standardního mikrokontroléru `PIC16F690` je využit pinově kompatibilní mikrokontrolér `PIC16F1579`, protože obsahuje čtyři nezávislé pulzně šířkové moduly (PWM). Čtyři moduly jsou využity, protože tříbarevné RGB pásky jsou čím dál častěji nahrazovány RGBW pásky, kde je přítomna vyhrazená bílá LED. Jako hardware nodu lze využít standardní nod, kde je výstup PWM buzení vyveden na svorky AD5 až AD8. Připojení výkonových obvodů případně nízkopříkonových LED je nutné realizovat mimo hardware nodu.

Druhou možností je využít hardware přímo navržený pro tuto specifickou úlohu. Obvod je navržen pro běžně dostupné LED pásky s napájením 12 až 24 V. Schéma zapojení tohoto nodu je znázorněno na obrázku 2, výkres plošného spoje na obrázku 7 a fotografie na obrázku 8 v příloze. Pro buzení LED pásku je využit N-FET tranzistor IRFZ44N[2] s maximálním Source-Drain napětím až 55 V při krátkodobém odběru až 49 A. Pro takto vysoké proudové odběry ovšem není hardware nodu dimenzován, a není doporučeno překročit výkon 20 W na jeden tranzistor. Pro chlazení tranzistorů byl vybrán chladič SK09-37SA220. Vstupní napájení nodu je nutné přivést na svorky PWR a musí být shodné s požadavky LED pásků, protože jsou realizovány jako diody se společnou anodou. K napájení mikrokontroléru a řadičů RS-485 je využit lineární stabilizátor 7805 s přítomným chladičem.

Pro správnou funkčnost nodu, ať už se se standardním nebo specializovaným obvodem, je nutné využít firmware `rgbw_pwm`, který korektně inicializuje `PIC16F1579` a nastavuje PWM na frekvenci 4 kHz. Po přijetí zprávy `SET` pak přenastavuje střídu PWM a okamžitě odpovídá zprávou `GET`. Nod s tímto firmwarem se subsystému ohlašuje čtyřmi složkami s šířkou 8 bitů.

7.1.4 Nod pro měření pevných částic v ovzduší

Pevné částice (anglicky: particulate matter, PM) jsou částice pevného skupenství tak malých rozměrů. že mohou být unášeny vzduchem a pro lidské oko nejsou viditelné [91]. Tyto částice mohou vznikat z lidských činností jako je spalování fosilních paliv nebo těžba, mohou být produkovány dieselovými motory, dokonce i během vaření. Krátkodobá expozice může pro lidský organismus znamenat zvýšení počtu zánětlivých onemocnění, dlouhodobá expozice potom může vést k chorobám dolních cest dýchacích, případně zvýšení výskytu chronické obstrukční plicní nemoci [131]. Je dokázáno, že střední délka života

[2] Je možné využít libovolný N-FET tranzistor v pouzdře TO220 s kompatibilním rozložením vývodů.

se v méně rozvinutých zemích snižuje o více než jeden rok právě kvůli těmto částicím. Obvykle se stanovují částice PM10, PM2,5 a PM1, kde číslo udává počet mikrometrů. PM10 tedy označuje částice menší než $10\,\mu\mathrm{m}$.

Tento nod slouží pro měření pevných mikročástic a využívá senzor Honeywell HPMA115S0, měřící PM10 a PM2,5 částice. Senzor pracuje na principu rozptylu laserového paprsku přes komoru s částicemi. Schéma zapojení nodu je znázorněno na obrázku 3, výkres na obrázku 7 a fotografie na obrázku 8 v příloze. Hardware využívá mikrokontroléru `PIC16F15324`, vybaveného dvěma rozhraními UART a možností mapování periferií a pinů. Vstupní napětí nodu je 5 V, ovšem senzor komunikuje v napěťové logice 3,3 V, a proto je mikrokontrolér napájen přes stabilizátor.

Firmware nodu s názvem `hpma115S0` se subsystému prezentuje dvěma složkami s šířkou dat 16 bitů a možností čtení dat každou minutu. Na první složce vrací firmware subsystému úroveň PM2,5, na druhé pak PM10. Po nastavení oscilátoru a periferií posílá příkaz senzoru k vypnutí automatického měření a zastavení ventilátoru pro úsporu energie a zvýšení životnosti ventilátoru. Pokud senzor neodpoví zpět, případně neodpovídá kontrolní součet odpovědi, je nastavena globální proměnná `sensor_ok` na nulu. Při přijetí `MEASURE` se pak mikrokontrolér snaží reinicializovat senzor. V případě úspěchu se `sensor_ok` nastavuje na jedničku. Díky tomu je možné vyměnit starý senzor za nový bez nutnosti vypnutí a zapnutí nodu. V korektním stavu se po přijetí zprávy `MEASURE` pošle senzoru příkaz na sepnutí ventilátoru. Následně se čeká 8 sekund a je poslán příkaz pro vyčtení PM hodnot a vypnutí ventilátoru. Případné problémy se senzorem indikuje firmware hodnotou 65 535 na obou kanálech. Doporučené mapování na komponentu se provádí výrazem:
`a != 65535 ? a : 0/0`.

7.1.5 Nod s precizním Sigma-Delta ADC převodníkem

Po adresaci nod již ví svoji adresu a dokáže předpovídat, jaké CRC budou mít pakety se zprávami `REQUEST`, `IDENTIFY_ON` a `IDENTIFY_OFF`. Po první adresaci zapíná přerušení vyvolané při přijatém bajtu od subsystému. V obsluze přerušení udržuje kruhový zásobník o délce 7 bajtů, což odpovídá délce paketů. Pokud nastane shoda se zprávou `REQUEST`, je nastaven příznak `response_flag`, pro zprávy identifikační je změn stav proměnné `has_identify` a odeslána odpověď. Odesílání je implementováno asynchronně. Nejprve je naplněno pole `send` cílovým paketem, je vynulována proměnná `send_index` a proměnná `to_send` je nastavena na požadovaný počet odchozích bajtů. Pak je spuštěn první časovač s periodou 30s, což je dostatečný čas pro odeslání bajtu přes sběrnici zpět subsystému. Když dojde k přerušení prvním časovačem, je odeslán bajt na dle indexu proměnné `send_index`. Poté je index inkrementován a proměnná `to_send` dekrementována. Pokud je proměnná `to_send` rovna nule, nejsou žádné další bajty k odeslání a první časovač je vypnut.

Měřené vstupy AD převodníku se střídají podle požadovaného počtu kanálů. V případě jednokanálové varianty je vstup AD převodníku měřen s frekvencí 660 Hz, dva kanály s frekvencí 330 Hz, tři kanály s frekvencí 220 Hz a čtyři pak s frekvencí 165 Hz. V příloze je na na obrázku 4 zobrazeno schéma, výkres na obrázku 7 a fotografie na obrázku 8. Využívá mikrokontrolér PIC16F15345 a AD převodník je připojen jako hotový modul od společnosti Adafruit.

7.1.6 Shrnutí

V tabulce 7.1 jsou shrnuty všechny vyvinuté firmwary, jejich charakteristiky, podporované typy zařízení, připojitelné limity a doporučený hardware nodů. Firmwary `ads1115`, `bh1750`, `bme280` a `tcs34725` vyžadují mikrokontrolér PIC16F15324, firmwary `hpma115S0` a `relay` mikrokontrolér PIC16F15324, firmware pro řízení led pásků pak PIC16F1579. Ostatní firmwary pak vyžadují model PIC16F690. Vzhledem k jejich omezeným parametrům je jasné, že navržený protokol neklade vysoké nároky na prostředky mikrokontrolérů. Implementovaná funkcionalita podporuje všechny běžné požadavky na sběr environmentálních hodnot i požadavky na rychlý sběr surových dat.

Tabulka 7.1: Firmwary pro kabelové nody [autor]

Firmware	**Č/Z**	**Lat.**	**Data**	**Podporovaný HW**	**Maxima**	**PCB**
ads1115	Č	0	(2-5)x16b	Texas Instruments ADS1115 16-bit S-D ADC	1..4 vstupy	standard ads1115
bh1750	Č	1,5s	1x16b	Rohm Semi I2C lux senzor světla	1 senzor	standard
bme280	Č	2s	3x24b	Bosh BME280 I2C senzor teploty, vlhkosti a tlaku	1 senzor	standard
dht22	Č	4s	2x16b	DHT22 a AM2302 senzor teploty a vlhosti	1..8 senzorů	standard
ds18b20	Č	4s	2x16b	Maxim DS18B20 1-wire teplotní senzor	1..8 senzorů	standard
ad_input_delay	Č	0,1s	1x16b	Libovolný analogový nebo digitální vstup	1..8 vstupů	standard
ad_input_fast	Č	0	1x16b	Libovolný analogový nebo digitální vstup	1..8 vstupů	standard
dig_input_delay	Č	0,1s	1x1b	Libovolný binární vstup	1..8 vstupů	standard
dig_input_fast	Č	0	1x1b	Libovolný binární vstup	1..8 vstupů	standard
dig_output	Z	0	1x1b	Libovolný binární výstup	1..8 výstupů	standard
hpma115S0	Č	60s	2x16b	Honeywell HPMS115S0 senzor PM částic	1 senzor	pm
relay	Z	0	1x1b	Relé	2 relé	relay
rgbw_pwm	Z	0	4x8b	RGBW LED pásek	4 PWM výstupy	standard pwm
stepper_motor	Z	0	1x16b	Unipolární krokové motory	1..2 motory	standard
tcs34725	Č	0,1s	4x16b	Taos TCS34725 RGB senzor světla	1 senzor	standard
watttmeter	Č	1,5s	1x16b	SCT-013-015 proudová sonda	1 sonda	standard

7.2 Bluetooth Low Energy

Existuje mnoho společností (například Microchip, Texas Instruments, Sillicon Labs, NXP, ST atd.) vyrábějících mikrokontroléry se zabudovaným rádiem a firmwarem implementujícím technologii Bluetooth

Low Energy. Pro vývoj bezdrátových nodů byl zvolen mikrokontrolér nRF52832 společnosti Nordic Semiconductor, zejména pro svoji nízkou spotřebu, široké množství podporovaných sběrnic a kvalitu poskytnuté dokumentace a ukázek. Je postaven na architektuře ARM Cortex-M4 s 512 kB paměti Flash, 64 kB RAM, rozhraními UART, SPI a I^2C a 12 bitovými AD převodníky. Čip je vyráběn pouze v pouzdře QFN s rozměry 6×6 mm a 48 piny. Pro vývoj byl použit vývojový kit nRF52 DK.

Programování je možné v několika vývojových prostředích, stejně tak lze využít komerční placené kompilátory, nebo svobodný kompilátor GCC. Nordic Semiconductor poskytuje kompletní SDK pro komunikaci se zabudovaným rádiem a jednodušší obsluhu sběrnic, časovačů, přerušení atd. Spolu s SDK je nutné využít tzv. SoftDevice, neboli předkompilovaný binární firmware, který je nutné nahrát do mikrokontroléru. Uživatelská část firmwaru se umisťuje za koncovou adresu SoftDevice. Pokud metoda obsažená v SDK vyžaduje volání metody v SoftDevice (například zapsání nové hodnoty charakteristiky a odeslání), je v SDK uvedena absolutní adresa obslužné rutiny a je proveden skok. Když metoda SoftDevice vrací návratovou hodnotu nebo ukazatel, je využito softwarové přerušení. Podle požadavků na funkcionalitu zařízení je k dispozici několik různých SoftDevice. Ty, které například podporují jak periferní, tak i centrální roli mají větší velikost a nechají nižší část volné paměti pro uživatelský firmware. Nody využívají základní SoftDevice S110. Pro bezdrátové nody byly vyvinuty tři odlišné firmwary.

7.2.1 Firmware pro binární vstup

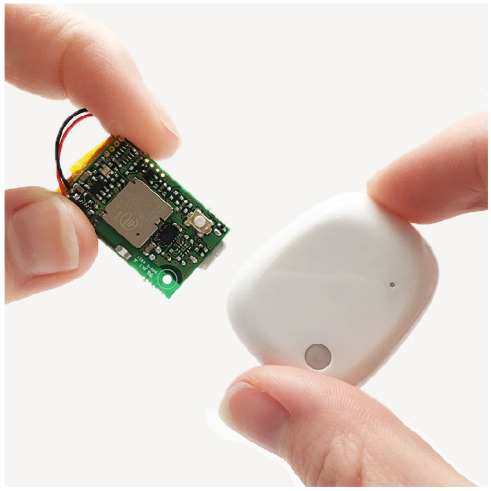
Tento firmware nastavuje GPIO piny mikrokontroléru jako výstupní a implementuje čtecí službu `READ_SERVICE` spolu s její charakteristikou `READ_CHARACTERISTICS`. Firmware se subsystému identifikuje subsystému jako nod pro okamžité čtení, s jedním bitem, jednou složkou 8 kanály. Po přijetí datového bajtu jsou nastaveny piny vývojového kitu P0.11 až P0.18 na požadovanou logickou úroveň.

7.2.2 Firmware pro binární výstup

Nahráním tohoto firmwaru do vývojového kitu se začne subsystému identifikovat jako nod pro zápis, s jedním bitem a jednou složkou, tedy jako digitální vstup s 8 kanály. Implementuje službu pro zápis `WRITE_SERVICE` a její charakteristiku `WRITE_CHARACTERISTICS`. Využívá stejné piny jako firmware pro vstup. Aby nebylo nutné periodicky kontrolovat stav pinů, je využito hardwarové přerušení. Firmware jinak mikrokontrolér uspává pro snížení spotřeby elektrické energie.

7.2.3 Nositelné zařízení MetaMotionR

Tento firmware nevyužívá vývojový kit nRF52 DK a je cílen pro nositelné zařízení MetamotionR od společnosti mbientlab. Podoba zařízení je znázorněna na obrázku 7.2. Jedná se o zařízení určené pro vědecké projekty využívající environmentální nebo prostorová data. Zařízení i přes svoje miniaturní rozměry obsahuje nabíjecí LiPo baterii, tlačítko, indikační RGB LED, akcelerometr, gyroskop, magnetometr, barometr, senzor ambietního osvětlení a 8 mbit paměť. Přes doprovodnou aplikaci MetaBase pro operační systém Android a iOS lze konfigurovat jaké senzory při jaké frekvenci se mají přes BLE přenášet v reálném čase, nebo ukládat do zabudované paměti. Lze tak například se vzorkovací frekvencí 100 Hz přenášet data buď z akcelerometru, gyroskopu nebo magnetometru. Pokud je nutné měřit všechny senzory souběžně, klesá frekvence vzorkování na 50 Hz, nebo lze vyčítat softwarově dopočítané Eulerovy úhly.



Obrázek 7.2: Nositelný senzor MetaMotionR [143]

Software zařízení, ani BLE služby a charakteristiky nejsou otevřené, ovšem je k dispozici schéma zapojení zařízení a na jeho PCB jsou vyvedeny programovací piny. Po připojení JTAG lze do mikrokontroléru nahrát vlastní firmware přes programátor ST-Link. Zařízení využívá stejný mikrokontrolér nRF52832 jako předešlé nody. Byl proto vytvořen firmware, který dokáže přenášet data ze zabudovaného SPI gyroskopu Bosh BMI160 v reálném čase se vzorkovací frekvencí 200 Hz a zařízení přeměňuje na MM_Gyro. Ostatní senzory jsou při startu firmwaru deaktivovány pro úsporu spotřeby zařízení. Firmware implementuje BLE služby `READ_SERVICE`, `IDENTIFY_SERVICE` a také službu monitorování baterie. Při adresaci se nod ohlašuje jako okamžitý čtecí, s šířkou dat 16 bitů a 10 složkami.

Aby bylo možné přenášet data se vzorkovací frekvencí 200 Hz, jsou v jednom datovém BLE rámci odeslána 3 měření gyroskopu spolu s číslem vzorku. Data jsou z nodu do subsystému přenášena v tomto pořadí:
`X1 Y1 Z1 X2 Y2 Z2 X3 Y3 Z3 CISLO_MERENI`,
kde `X`, `Y` a `Z` značí osu gyroskopu. Subsystém tedy přijímá data s třetinovou frekvencí (66 Hz) oproti frekvenci vzorkovací. Dvojnásobná vzorkovací frekvence oproti originálnímu firmwaru výrobce umožňuje měření rychlejších jevů.

7.3 Z-Wave

V tabulce 7.2 jsou vypsána všechna v současné době podporovaná a otestovaná Z-Wave zařízení. Jedná se o chytré zásuvky, jelikož spínání síťového napětí 230 V pomocí kabelových nodů bez potřebných certifikací je doporučenou pouze na vlastní nebezpečí. Dále jsou podporována různá senzorická zařízení, žárovka, stmívač, termostat a chytrý zámek dveří. Podpora dalších zařízení bude zahrnována podle budoucích požadavků.

7.3.1 Vývoj vlastních Z-Wave zařízení

Jakákoliv zařízení určená ke komerční distribuci musí projít certifikačním procesem společnosti Sigma Designs, vlastníkem ochranné známky Z-Wave a Z-Wave plus. Je však možné vyvíjet vlastní prototypy s využitím vývojového kitu Z-Uno [200]. Obdobně jako BLE mikrokontrolér nRF52832 obsahuje pro-

Tabulka 7.2: Podporované Z-Wave zařízení [autor]

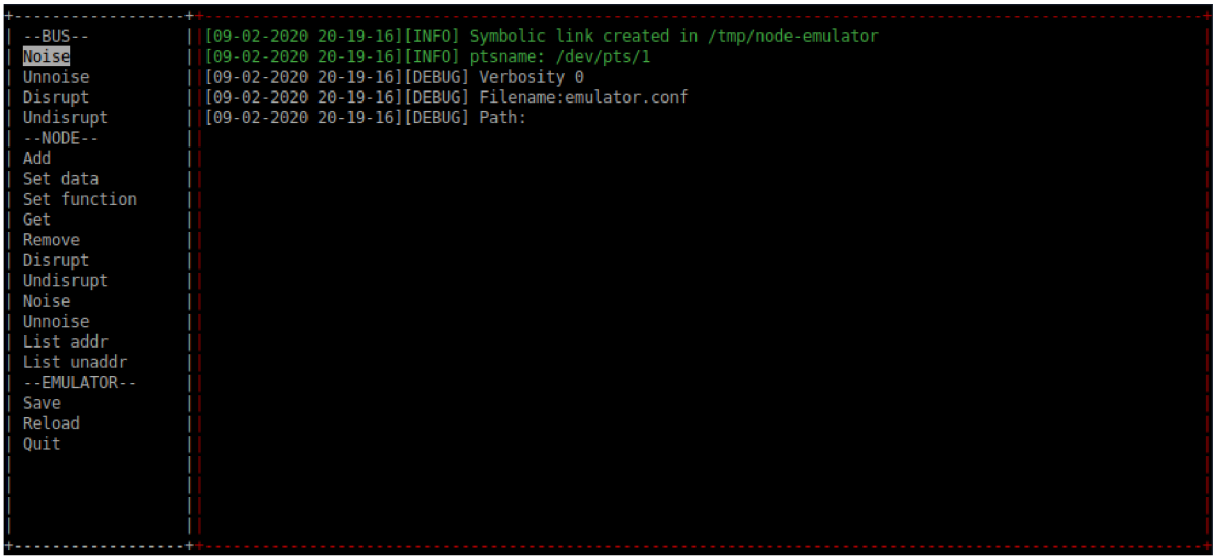
Výrobce	Model	Typ zařízení	Čtená data	Zapisovaná data
Aeotec	Plug Gen5	Zásuvka	Vypnuto / zapnuto, watty, volty, ampéry	Vypni / zapni
Aeotec	Plug Gen6	Zásuvka	Vypnuto / zapnuto, watty, volty, ampéry	Vypni / zapni
Aeotec	Bulb6	RGBW E27 žárovka		Barva světla
Aeotec	Multisenzor Gen6	Kombinovaný senzor	Teplota, vlhkost, pohyb, intenzita světla	
Aeotec	Minimote	Dálkový ovladač	ID tlačítka	
Fibaro	Smart plug	Zásuvka	Vypnuto / zapnuto, watty	Vypni / zapni
Fibaro	Mini dimmer	Stmívač světel	Intenzita svitu	
Fifthplay	ZD2102EU-5	Dveřní nebo okenní senzor	Otevřeno / zavřeno	
Go Controll	GC-TBZ48	Termostat	Teplota, limity, stav klimatizace	Teplota, limity, stav klimatizace
Go Controll	LB60-Z1	Stmívatelná E27 žárovka		Intenzita svitu
Heiman	HS1HT	Kombinovaný senzor	Teplota, vlhkost	
Kwikset	SmartCode 916	Zámek dveří	Odemčeno / zamčeno	Odemkni / zamkni

prietární firmware, který pomocí softwarového přerušení komunikuje s uživatelským kódem. Pro vývoj uživatelské části je využito prostředí Arduino. Firmware podporuje pouze omezené množství programových tříd, ovšem je zahrnuta podpora třídy `COMMAND_CLASS_MULTI_CHANNEL`, určené pro realizaci takových zařízení, která neodpovídají žádné již existující standardní třídě. Lze tak implementovat podporu téměř libovolného zařízení, pokud uživatelský kód nezabere více jak 30 kB paměti Flash a 2 kB RAM v nezabezpečeném přenosu, nebo 6 kB paměti Flash a 2 kB RAM při šifrovaném AES-128 spojení mezi vývojovým kitem a řídící jednotkou.

7.4 Podpůrná aplikace Emulátor

Aplikace Node Emulator vznikla původně pro potřeby vývoje subsystému, jelikož není možné mít ve všech situacích přítomen hardware nodů a subsystému pro vývoj. Cílem emulátoru je co nejvěrněji napodobit chování RS-485 nodů programově. Aplikaci je možné využívat pouze na operačním systému Linux a stejně jako Server Simulator je distribuována pod otevřenou licencí GNU/GPLv3.

Node Emulator pracuje ve dvou režimech. Textový grafický režim je vyobrazen na 7.3 a je možné v něm simulovat různé poruchy komunikace, přidat nové nody a adresovat je, nastavovat stav nodu pro čtení, který bude následně vyčten subsystémem nebo sledovat jaké stavy nodu pro zápis byly nastaveny. Alternativním režimem je běh aplikace jako demón, kdy je vytvořena pojmenovaná roura a aplikaci lze přes dokumentované rozhraní ovládat z libovolného programovacího jazyka. Tímto přístupem lze vytvořit například virtuální nod, který místo hardwaru sděluje IoTIM různé systémové informace subsystému, nbo lze takto implementovat podporu senzorů s dostupnými ovladači pro systém Linux.



Obrázek 7.3: Aplikace Node Emulator [autor]

8

Diskuze a zhodnocení

V této kapitole bude nejprve pojednáno o možnostech alternativních implementací navržené architektury, poté zhodnoceno splnění cílů disertační práce a závěrem budou představeny projekty ve kterých byla již implementace nebo její vybrané části využity.

8.1 Možnosti alternativních implementací

V kapitole 5 byla představena implementace IoTIM, v kapitole 6 pak implementace subsystému a v sedmé kapitole vlastní implementace nodů s metalickou technologií RS-485, bezdrátovou technologií Bluetooth Low Energy a komerční Z-Wave zařízení. Díky flexibilně navržené architektury mikroslužeb IoTIM, aplikace subsystému a nodů se jedná pouze o jednu z možných variant implementace platformy.

V případě IoTIM lze například zvolit odlišné implementační jazyky mikroslužeb než Rust, C++ a Python. Apache Kafka podporuje desítky dalších jazyků, jako Java, C#, Google Go, PHP, Ruby a další. Samotný Apache Kafka není jedinou využitelnou technologií pro data streaming a je možné využít Redis, RabbitMQ, ActiveMQ a další. Stejně tak místo technologií Telegraf, InfluxDB a Grafana lze využít téměř identické služby Metricbeat, ElasticSearch a Kibana. Subsystémy nemusejí spoléhat výhradně na MQTT protokol. CoAP, WebSocket či vlastní protokol implementovaný nad TCP spojením dokáže splnit požadavky na komunikaci subsystémů s IoTIM. Dle požadavků mohou subsystémy využívat alternativních technologií než RS-485, BLE a Z-Wave. Například IoT platforma specializovaná osvětlení a monitoring prostředí může využít kombinaci technologií IQRF, DALI a KNX. Díky abstrakci dat nodů lze implementovat v podstatě jakoukoliv myslitelnou komunikační technologii.

8.2 Splnění cílů disertační práce

V kapitole 2 bylo stanoveno několik dílčích cílů práce. Seznam bude nyní zhodnocen.

- **Vymezení základní terminologie a obecného modelu senzorických IoT architektur a různé pohledy na oblast IoT** - v kapitole 3.2 je popsán doménový model IoT spolu s terminologií jednotlivých entit, dále pak funkční model v kapitole 3.3. V navazující kapitole 3.4 je představena čtyř-vrstvá architektura IoT platforem. Aplikace spadající do Internetu věcí jsou popsány z pohledu nasazení do provozu v kapitole 3.8 a ekonomických přínosů pro firmy a uživatele v kapitole 3.9.

- **Představení technologií využitelných v oblasti Internetu věcí** - v kapitole 3.5 jsou popsány využitelné v IoT. Z důvodu vysokého počtu technologických oblastí je kapitola zaměřena na přenosové technologie a protokoly.
- **Popis existujících IoT platforem** - komerční i open-source platformy jsou popsány v kapitole 3.11.
- **Provedení rešerše současného stavu výzkumu v oblasti Internetu věcí a nalezení řešených problémů** - v kapitole 3.10 je nalezeno několik oblastí Internetu věcí s otevřenými výzkumnými otázkami.
- **Porovnání existujících řešení z hlediska nalezených problémů** - srovnání IoT platforem z hlediska výzkumných problémů v kapitole 3.12 ukazuje, že neexistuje takové řešení, které by realizovalo všechny popsané problémy.
- **Návrh architektury nového řešení** - návrhem nové IoT platformy, rozdělené na integrační middleware, subsystémy a nody popisuje kapitola 4. Nejprve je popsán datový model nového řešení a poté případy užití a blokové architektury jednotlivých vrstev.
- **Implementace vybraných softwarových a hardwarových částí nového systému** - částečnou ukázkovou implementaci IoTIM prezentuje kapitola 5, plnou implementaci subsystému pak kapitola 6 a v kapitole 7 je představen implementovaný hardware a firmwary nodů s technologiemi RS-485 a Bluetooth Low Energy a představeny subsystémem podporovaná komerční Z-Wave zařízení.
- **Diskuze a představení výzkumných projektů využívajících nové řešení** - výzkumné projekty jsou představeny v této kapitole.

V kapitole 3.10 bylo představeno šest vybraných problémů, řešených v současných výzkumných publikacích z oblasti IoT. Po provedení rešerše současných komerčních a open-source platforem v kapitole 3.11 a jejich srovnání v kapitole 3.12 bylo zjištěno, že neexistuje takové řešení, které by je souhrnně implementovalo. Neznamená to ovšem jakoukoliv nedospělost či nepoužitelnost těchto platforem. Ovšem v kapitole 3.13 je představen důvod nutnosti návrhu a implementace nové architektury. Důvodem je usnadnění realizace a verifikace senzorických výzkumných experimentů. Architektura implementující řešení všech šesti vybraných problémů přinese možnost využít IoT platformu skrze celý životní cyklus experimentu od výběru senzorů, přes sběr a vyhodnocení dat až po návrh a testování edge compute algoritmů a nasazení ve skutečném prostředí. Se současnými platformami toto není možné a je nutné využít kombinace různých programů, případně určitých některých požadavků (jako je např. přesná synchronizace času) není možné dosáhnout bez využití dedikovaných měřicích přístrojů. Nyní bude zhodnoceno dosažení jednotlivých požadavků na nové řešení vypsaných v 3.13.

Synchronizace mezi času mezi subsystémem a IoTIM

Jedním ze stanovených požadavků bylo umožnit synchronizovat čas bran (v implementaci nazývané subsystémy) s IoT integračním middlewarem. Algoritmus, který zároveň detekuje přítomnost subsystému je představen v podkapitole 4.4.2. Detekce pracuje inverzně oproti přístupu v článku [67], protože subsystém aktivně detekuje IoTIM. Není tak zapotřebí mít veřejně dostupný subsystém z internetu při nasazení IoTIM na vzdálený server nebo aplikační cloud. Implementace subsystému umožňuje

mechanismus synchronizace času nezahrnout do sestavení aplikace, jak je popsáno v kapitole 6.9. Bez tohoto sestavení se musí subsystém spoléhat na přítomnost externích služeb poskytujících aktuální čas v dostatečné přesnosti, jinak nemohou být data nodů analyzována historicky. Dopady špatného času tří bran jsou v demonstrovány v kapitole 3.10.1, kdy rozdíl pouhých jednotek sekund dokáže změnit interpretaci naměřených hodnot. Důležitost synchronizace času se odlišuje v závislosti na aplikačním využitím. V prostředí domácí automatizace je implementovaný mechanismus téměř nadbytečný, ovšem ve scénáři sběru surových dat z několika zdrojů souběžně je zcela kritický pro evaluaci měření.

Offline pravidla subsystému

Struktura pravidla je navržena v kapitole 4.2.2. Je zde odlišen rozdíl mezi pravidlem subsystému a pravidlem IoTIM, kdy IoTIM dokáže do svého pravidlového systému začlenit i vstupy z externích zdrojů. Implementace pravidlového systému je popsána v kapitole 6.5. Vytvořený systém dokáže přijímat stavy komponent okamžitě po přechodu subsystému do offline stavu. V případě zaplnění fronty pravidel čekajících na zpracování je implementován mechanizmus balancování zátěže aplikace (viz 6.11, který dokáže frekvenci vyčítání nodů krátkodobě zpomalit.

Ze zkoumaných komerčních a open-source řešení implementuje podporu offline pravidel v bráně pouze platforma Samsung Smart Things, ovšem pouze velmi omezeně s cílem umožnit alespoň ovládání osvětlení domu v případě výpadku internetového připojení. Nově implementovaný přístup umožňuje skrze API IoTIM vytvořit v subsystému pravidla využívající jako vstupy nativní i edge komponenty. Pravidla lze spouštět na základě nového stavu komponenty, periodicky nebo v přesně daném časovém okamžiku. Díky tomu lze například v případě aplikační domény domácí automatizace zachovat částečnou funkcionalitu jednotlivých místností v případě problémů s IoTIM. Dalším možným využitím je využívat jeden subsystém jako zcela soběstačnou řídící jednotku, která je při prvotní konfiguraci jednorázově nastavena skrze IoTIM nebo aplikace Server Simulator (viz kapitola 5.2).

Offline logování událostí v subsystému

V kapitole 4.3.1, popisující vzájemné propojení mikroslužeb a aplikací IoTIM byla popsána aplikace Offline logovací backend. Jedná se o aplikaci, která umožňuje přijímat logy vzniklé v subsystému během nepřítomnosti Externího zprávového brokera nebo síťového spojení. Kapitola 4.4.1 v blokovém diagramu subsystému představuje modul Logování offline událostí. Robustní implementace přijímání offline logů na IoTIM je popsána v 5.1.2. Protistrana v podobě implementace subsystému je představena v 6.8.

Přínosem offline logování v subsystému je schopnost uchovat naměřené stavy komponent v případě náhlé nedostupnosti internetového připojení nebo při nutnosti modifikace či aktualizace IoTIM. Subsystém, v případě ukázkové implementace tedy Raspberry Pi s SD kartou jako systémovým úložištěm, lze vybavit dostatečně velkou kapacitou a použít jej pro měření v lokalitách bez internetového připojení. Je nutné pouze zajistit prvotní synchronizaci časové známky. Protože subsystém neustále detekuje přítomnost IoTIM, budou nové stavy komponenty ukládány dokud bude dostupné volné místo v úložišti. Historická data komponent se po znovupřipojení IoTIM odesílají díky technologiím rsyslog a Telegraf na pozadí souběžně s aktuálními daty komponent. Z existující platforem implementují tuto funkcionalitu pouze Microsoft Azure IoT a IBM Watson, ale postrádají implementace jiných požadovaných funkcionalit.

Edge Computing v subsystému a nodech

V návrhu subsystému v kapitole 4.4.1 je popsán modul který spravuje externí moduly implementující algoritmy. Moduly jsou v spustitelné skripty či aplikace a data nativních komponent přijímají od subsystému skrze TCP spojení a stejným způsobem i odesílají výsledky výpočtu. Implementace je popsána v kapitole 6.3. Edge computing v nodech je díky transparentnosti dat velmi jednoduchý. Nod se chová identicky jako nod se senzorem, ovšem v popisku firmwaru či jeho identifikátoru zmiňuje jaké hodnoty poskytuje. Možnosti edge computingu v nodech jsou tak limitovány pouze vybranými hardwarovými a softwarovými prostředky nodu. Při implementaci byly využity jednoduché a cenově velmi dostupné osmibitové mikrokontroléry Microchip řady 16F, vybavené operační pamětí o velikosti jednotek kilobajtů a maximálním taktem 32 megahertzů. Na trhu jsou k dispozici mikrokontroléry s řádově lepšími parametry, jako například řada H7 společnosti ST [201]. Nejvyšší dvou jádrový model této řady disponuje jádrem architektury ARM Cortex-M7 s maximálním taktem 480 MHz a jádrem Cortex-M4 s taktem až 240 MHz. V kombinaci s operační pamětí o velikost 1 MB a FLASH pamětí 2 MB umožňuje realizaci náročných aritmetických operací, které bylo možné dříve realizovat pouze na počítačích nebo na specializovaných signálových procesorech či programovatelných hradlových polích.

Edge computing v subsystému představuje možnost signifikantní úspory počtu přenášených dat směrem k IoTIM i jejího požadovaného výkonu. Raspberry Pi čtvrté generace disponuje až 4 gigabajty operační paměti a 4 procesorovými jádry. Lze tak implementovat edge modul v jazycích nižší úrovně, které mohou realizovat komplexní výpočty. Publikace, které využívají různé algoritmy a senzorů v podobě gyroskopů nebo akcelerometrů jsou popsány v kapitole 3.13. Tyto algoritmy mohou být právě díky edge computingu implementovány přímo v subsystému a na IoTIM poskytovat pouze vypočtené údaje.

Externí datové zdroje

V kapitole 4.3.1 byly popsány zodpovědnosti mikroslužeb Management objektů a Management dat. Ty jsou využity k vytvoření externích komponent bez mapování na fyzické nody, umožňující perzistenci dat z klientských aplikací. Implementace perzistence externích dat je popsána v kapitole 5.1.2. Využívá identickou databázi jako data nativních komponent, pouze místo identifikátoru subsystému je pro identifikaci vyžito klíčové slovo `external`.

Perzistence externích datových zdrojů umožňuje sbírat data i z nepřímo podporovaných technologií. V kapitole 3.5.2 jsou popsány úsporné bezdrátové technologie s dlouhým dosahem SigFox a LoraWAN. V případě SigFox zařízení odesílá data přes infrastrukturu národního poskytovatele a data jsou ukládána do jejich proprietárního úložiště. LoraWAN umožňuje provoz vlastních radiových stanic, ovšem pouze v nezbytných případech, jinak je též doporučenou strategií využít existující pokrytí. Pokud IoTIM neobsahuje prostředky pro uložení dat z externích zdrojů, není možné takové technologie integrovat. V případě potřeby integrace těchto technologií je možné nové řešení rozšířit o klientskou aplikaci, které bude v periodických intervalech volat API externích služeb a nová data ukládat pomocí služby Managementu dat.

Tok dat v reálném čase

V digramu kompozice mikroslužeb uvedeném v kapitole 4.3.1 je definována samostatná mikroslužba poskytující data komponent v reálném čase klientským aplikacím. Služba přijímá stavy skrze externí

zprávový broker a poskytuje je přes vhodné komunikační prostředky. Implementace služby je popsána v kapitole 5.1.4. Po ověření oprávnění a uvedení seznamu komponent skrze REST API jsou data asynchronně poskytována přes WebSocket spojení.

Z existujících platforem nabízí podporu poskytování dat v reálném čase platforma Amazon Web Services IoT Core. Technicky se ovšem nejedná o plný real-time přístup, jelikož je stále nutné periodicky pokládat dotazy na nová data namísto asynchronního přijímání. Ostatní platformy tuto funkcionalitu nepodporují a nebo je její implementace omezená na využití pouze ve vybraných proprietárních technologiích. Implementovaná mikroslužba poskytuje data skutečně asynchronně díky využití WebSocket. Data komponent ze subsystému jsou transformovány z úsporného binárního formátu FlatBuffers na JSON, který je podporovaný v široké škále jazyků. Real-time data umožňují testování různých algoritmů a zrychlují jejich ladění. Druhým aplikačním využitím je možnost vytvořit specializované grafické přehledy a zobrazovat například analogová měření obdobným způsobem jako na osciloskopu. Tuto funkcionalitu již podporuje také vyvinutá aplikace Server Simulator.

8.2.1 Shrnutí

V tabulce 8.1 jsou znovu uvedeny IoT platformy zkoumané v kapitole doplněné o nové řešení. Z tabulky je zřejmé, že nově navržená architektura úspěšně řeší hlavní cíle disertační práce a jedná se tak o jediné současné řešení nabízející možnost realizace senzorických výzkumných projektů bez nutnosti využití kombinací různých platforem a hardwarových prostředků. Tohoto cíle lze komfortně docílit pouze platformou která uceleně implementuje všechny popsané problémy. Které vlastnosti mohou být využity v konkrétních krocích realizace senzorického experimentu jsou popsány v kapitole 3.13.

Tabulka 8.1: Přímé srovnání vlastností IoT platforem [autor]

	Vlastní hostování	**Offline pravidla**	**Offline události**	**Synch. času**	**RT API**	**Edge computing**	**Externí zdroje**
Azure	Ne	Ne	Ano	Ne	Prop.	Zařízení, Brána	Ne
Amazon	Ne	Ne	Ne	Ne	Ano	Zařízení, Brána	Ne
SmartThings	Ne	Ano	Ne	Ne	Ne	Ne	Ne
SiteWhere	Ano	Ne	Ne	Ne	Ne	Ne	Ne
openMTC	Ano	Ne	Ne	Ne	Ne	Ne	Ne
Webinos	Nezjištěno	Ne	Ne	Ne	Ne	Nezjištěno	Ne
FIWARE	Ano	Ne	Ne	Ne	Ne	Zařízení, Brána	Ne
OpenIoT	Ano	Ne	Ne	Ne	Ne	Ne	Ne
IoT-Framework	Ano	Ne	Ne	Ne	Ne	Ne	Ano
IBM Watson	Ne	Ne	Ano	Ne	Prop.	Zařízení, Brána	Ano
ThingSpeak	Ne	Ne	Ne	Ne	Prop.	Ne	Ne
Node-RED	Ano	Ne	Ne	Ne	Ne	Ne	Ne
Nové řešení	Ano	Ano	Ano	Ano	Ano	Nod, Subsystém	Ano

8.3 Nasazení implementovaného řešení v projektech

Implementované řešení bylo již úspěšně v různých rozsazích nasazeno ve výzkumných projektech.

8.3.1 Smart Furniture

Cílem projektu v rámci TA ČR GAMA TG02010020 bylo vytvoření prototypu systému pro zjišťování frekvence dechu a tepu osob pomocí senzorů umístěných v nábytku, například kancelářské židli, lůžku nebo pohovce apod. Jedná se tedy o tzv. „Smart Furniture“, chytrý nábytek se značnou přidanou hodnotou – senzory v nábytku poskytují data, která jsou pomocí vytvořených algoritmů převedena na informace o stavu osoby (tepová frekvence, dechová frekvence). Vytvořením komplexnějších algoritmů bude možné zjišťovat i úrovně stresu nebo únavy. Využití tohoto systému se předpokládá jak v domácím prostředí (kde je z prudkého nárůstu prodeje nositelných zařízení zřejmý zájem uživatelů o informace o stavu jejich fyziologie), tak i v prostředí komerčním, neboť v mnoha firmách se zvyšuje péče o zaměstnance a společnosti se snaží předcházet stresu zaměstnanců a maximalizovat efektivitu jejich práce. Oproti nositelným řešením, které kvůli omezené kapacitě baterií nemohou monitorovat uživatele dlouhodobě a přesně, přináší tento systém detailnější informace o uživateli, z kterých lze zjistit právě zmíněný stav únavy a stresu.

Celý systém pracuje v téměř reálném čase [1] a implementované řešení je využito v celé míře. V židli jsou přítomny 2 nody s převodníkem ADS1115 s firmwarem pro obsluhu dvou kanálů převodníku, jeden v části opěrky, druhý v sedáku. Jsou k nim připojeny přes nábojový zesilovač měřící senzory. Tyto nody jsou připojeny k subsystému komunikujícím se IoTIM (viz obrázek 8.1). Výpočet dechové a tepové frekvence probíhá komplexním algoritmem v programovém prostředí MATLAB, které přes REST API serveru zažádá o otevření WebSocket spojení a data analyzuje, provádí výpočet a na základě své konfigurace může zpětně přes server provádět různé akce, jako stmívat LED pásek v rytmu tepu, zobrazovat data na displayi, spínat zařízení při překročení spodní či horní meze dechu, atd. Vytvoření prototypu v rámci tohoto projektu bylo úspěšné, a projekt GAMA je již u konce.

Obrázek 8.1: Fyzická architektura chytrého nábytku [autor]

[1] Vytvořený systém vyžaduje alespoň 10 sekund měření v interním zásobníku.

8.3.2 Healthy Aging in Industrial Environment

Healthy Aging in Industrial Environment (HAIE) je výzkumným projektem Univerzity Ostrava. Cílem výzkumu je nalezení vztahů mezi environmentálními podmínkami, životním stylem, zdravotním stavem, kvalitou života a stárnutím [202]. V rámci subdodávky byly na Fakultu Kinantropologie instalovány 4 lůžka s prototypem SmartFurniture. Každý večer mimo neděle spí na lůžkách probandi, kteří předtím procházejí detailní fyzickou prohlídkou. Během spánku mají na zápěstí chytrý fitness náramek FitBit. Surová data z nodů jsou perzistovány na IoTIM a analyzovány. Výzkumníkům jsou poskytovány reporty o dechové, tepové frekvenci a spánkových fázích.

8.3.3 GAMA2 Covid

Během pandemie viru SARS-CoV-2 způsobující onemocnění COVID-19 byl prototyp Smart Furniture instalován na plicní oddělení Fakultní nemocnice Hradec Králové s cílem dlouhodobého sběru a vyhodnocování dat s cílem implementace algoritmů pro monitoring progrese nemoci. Druhotným cílem bylo otestování systému ve skutečném prostředí a sbírání zpětné vazby a praktických poznatků.

Využívá specifického hardwaru přímo pro monitoring vitálních funkcí, výpočty v mikrokontroléru a nový proprietární aplikačně specifický IoT middleware, optimalizovaný pro běh algoritmů pro výpočty vitálních charakteristik, jako dechová a tepová frekvence, variabilita srdečního tepu, výskyt arytmií a spankových apnoe a také predikcí progrese specifických onemocnění jako např. parkinson, alzheimer nebo COVID-19.

8.3.4 SmartLab

V rámci projektu OP PIK byl vytvořen ve spolupráci se společností C2P s.r.o. systém pro monitoring prostor s použitím pokročilých rozhodovacích metod pomocí komunikace se systémem pro správu senzorů umístěných v prostředí. Řešení využívá jak IoTIM tak subsystémy a vystavuje API pro rozhodovací aplikaci.

8.3.5 CHOPN

Cílem projektu bylo vytvoření komplexního podpůrného systému pro monitorování průběhu Chronické obstrukční plicní nemoci (CHOPN), skládající se z statické části se senzory polétavých PM částic umístěných prostředí. Nositelná část se pak skládá ze senzoru umožňující detekci tepové a dechové frekvence a dále kašlání či chrápání. Data jsou odesílány do centrálního hubu, kde jsou k dispozici k další analýze lékařem. Další částí je pak podpůrná aplikace pro mobilní telefony a tablety, provádějící geolokaci a na základě otevřených dat ČHMÚ doporučuje vycházet či nevycházet ven, případně větrat nebo ne. Druhou funkcí aplikace je zobrazování certifikovaného dotazníku CAT [40] pro pacienty s CHOPN.

Sběr dat byl rozdělen na dvě fáze, v té první probíhalo měření v Fakultní nemocnici Hradce Králové (FNHK) postupně na několika pacientech dobrovolnících. Druhou fází je nasazení řešení v obydlích pacientů. Zde je již využit centrální hub i podpůrná aplikace. Implementované řešení je v projektu využito pro celou senzorickou část projektu. Pro měření v nemocnici byl do instalační krabice namontován subsystém, dva nody pro měření PM částic a standardní nod s firmwarem BME280. Jeden senzor Honeywell HPMA115S0 byl přítomen v krabici s průduchy a druhý na venkovním parapetu v zateplené instalační krabici spojený datovým kabelem. Subsystém dále komunikoval s BLE zařízením MetaMotionR,

umístěným na hrudi pacienta. Data byla ze subsystému odesílána do notebooku s aplikací Server Simulator a skriptem synchronizujícím data na FTP server pro analýzu.

V domě je pak server nahrazen miniaturním počítačem Intel NUC se IoTIM, který data odesílá přes real-time API do centrálního hubu, kde jsou analyzována algoritmy pro detekci vitálních hodnot a slučovány s daty z dotazníku a prezentovány lékaři přes webovou aplikaci. Projekt TA ČR GAMA je již ukončen a bude na něj navázáno projektem s cílem zlepšení algoritmu o detekci pískotů, sípotů, vrzotů a chrůpek.

8.3.6 Autorův dům

V tomto případě se jedná se o entusiasmus autora práce, s cílem co nejvíce odladit a dlouhodobě testovat celou platformu, a také realizovat některé nápady, které usnadní žití v domě. V současné době je v domě přítomno pět subsystémů, využívajících implementovaný hardware se sestavením aplikace pro RS-485, Z-Wave a zabudované teplotní senzory. Rozmístění nodů a Z-Wave zařízení je ukázáno v tabulce 8.2.

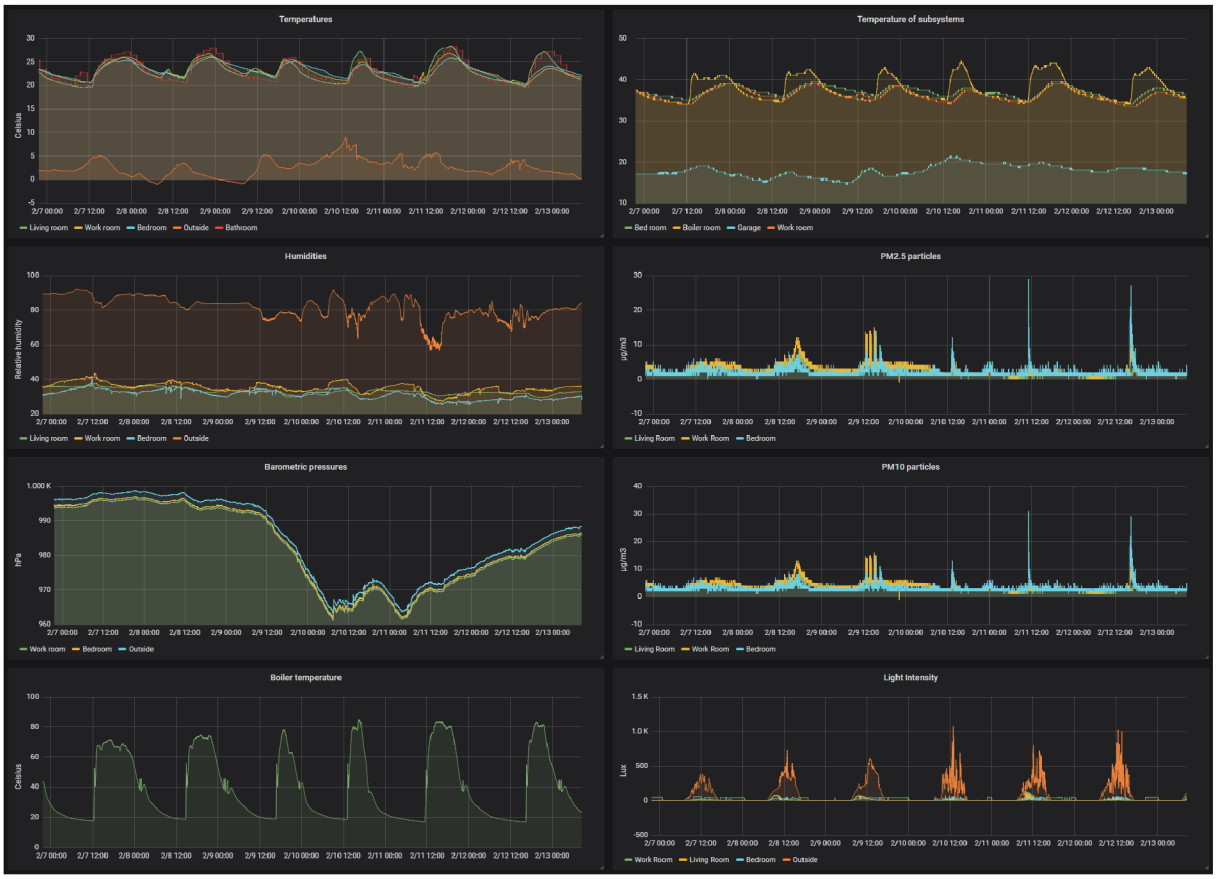
Tabulka 8.2: Nody a zařízení v domě [autor]

Místnost	**Nody a měřené veličiny**
Pracovna	Nod BME280 (teplota, vlhkost a barometrický tlak) Nod HPMA (prachové částice PM10 a PM2,5) Nod BH1750 (světlo prostředí) Nod s AD vstupem a připojeným mikrofonním modulem
Obývací pokoj	Nod DHT22 (teplota, vlhkost) Nod HPMA (prachové částice PM10 a PM2,5) Nod BH1750 (světlo prostředí) Nod s relé (spínání podsvětlení nábytku) Nod s RGB led páskem Nod s firmwarem wattmetru (spotřeba TV a receiveru) 2x Z-Wave žárovka Aeotec Bulb6
Koupelna	Z-Wave zásuvka Fibaro Z-Wave senzor teploty a vlhkosti Heinman
Venkovní prostory	Nod s AD vstupem a připojeným mikrofonním modulem Nod BH1750 (světlo prostředí) Nod BME280 (teplota, vlhkost a barometrický tlak)
Kotelna	Nod s teplotním senzor DS18B20
Ložnice	Nod BH1750 (světlo prostředí) Nod BME280 (teplota, vlhkost a barometrický tlak) Nod digitálním vstupem s připojením PIR čidlem Nod HPMA (prachové částice PM10 a PM2,5)

V domě je pak přítomen úsporný server s procesorem AMD Athlon 5350, 4GB RAM a 250GB SSD na kterém jsou nasazeny mikroslužby IoTIM. Na obrázku 8.2 jsou ukázány data z domu od 7.2.2020 do 13.2.2020. Nad IoTIM jsou implementována pravidla provádějící automatizační úlohy. LED pásek v ložnici je připevněný uvnitř postele. Ve spodní části postele je také pohybové čidlo HC-SR501 v krytu, který záměrně blokuje zorné pole čidla tak, aby došlo pouze k jeho sepnutí chozením v místnosti a v žádném případě během spánku. Pokud dojde k sepnutí čidla a v místnosti není dostatek světla, pravidlo rozsvítí LED pásek a nechá jej rozsvícený po dobu 30 sekund.

Druhá sada pravidel se týká obývacího pokoje a implementována jako klientská aplikace nad real-time daty. Pokud je v místnosti tma a není zapnutý televizor, je sepnuté podsvícení barové skříně pro ambientní efekt. Když je televizor zapnutý a v místnosti je zhasnuto, svítí místo podsvícení baru LED pásek, který je připevněn na zadní tělo televizoru. Barvy RGB složek se velmi pomalu mění každých 5 sekund. Jednotlivé složky se regulují funkcí sinus a jsou vůči sobě fázově posunuty o 120 stupňů. Jakmile je v místnosti dostatek přirozeného světla, nebo je v místnosti rozsvíceno, LED pásek i osvětlení baru není aktivní.

Další činnost, kterou skript provádí je zaslání notifikační zprávy na chatovací síť Telegram, pokud jsou překročeny přijatelné limity PM částic. Vytápění domu uhlím přináší v současné době pouze jedinou výhodu, a to finanční úsporu. Jinak se jedná o zastaralý přístup vyžadující nadměrné množství práce a případná zdravotní rizika. Pokud se v kotli nevytvoří hořením dostatečná teplota, tak v kombinaci s nízkým tlakem prostředí začne kotel dehtovat a přes přítomnost izolace v podobě skelné vaty propouštět exhaláty do domácnosti. V tomto případě je nutné v kotli okamžitě přiložit a otevřít okna.



Obrázek 8.2: Vizualizační prostředí s daty senzorů [autor]

9

Závěr

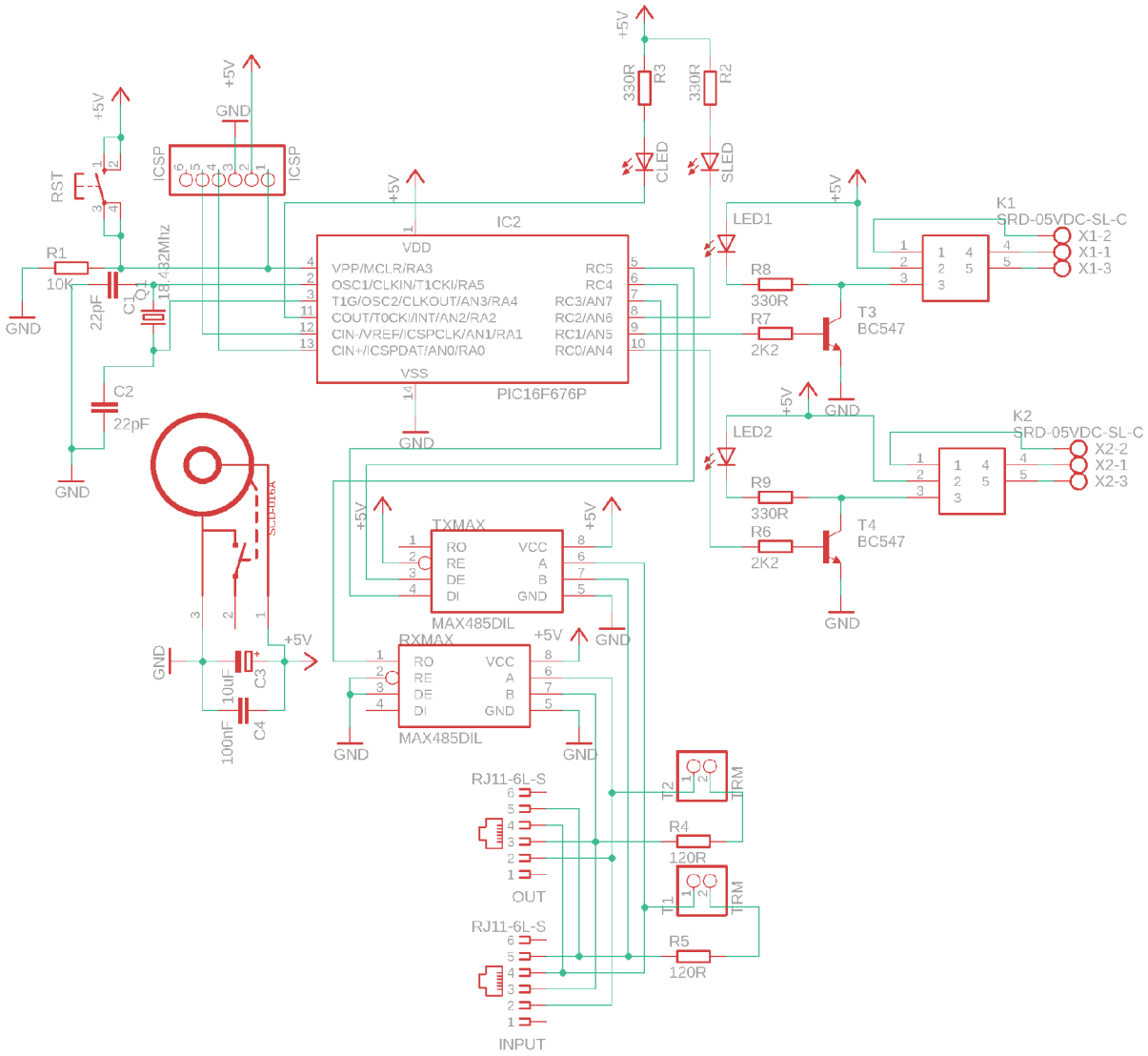
V této práci byla představena architektura Internetu věcí, systémů vzájemně propojených a komunikujících zařízení detailně popsaných doménovým a funkčním modelem. Dále je představena referenční fyzická architektura IoT platforem, sestávající se z senzorů a aktuátorů připojených na zařízení, komunikující s integračním middlewarem přímo, nebo skrze bránu. Klientské aplikace se připojují k middlewaru a vytvářejí tak IoT řešení. Následně je problematika dekomponována z hlediska nasazení a ekonomie a jsou představeny komerční a open-source IoT platformy a popsány jejich blokové architektury.

Po prozkoumání současného stavu výzkumu v oblasti Internetu věcí bylo zjištěno, že se jedná o oblast s mnoha výzkumnými problémy a příležitostmi. Bylo vybráno šest řešených aplikačních problémů, jmenovitě problematika synchronizace času mezi branami a integračním middlewarem, podpory přenosu informací do klientských aplikací v reálném čase, rozšířením schopností bran o možnost provádět uživatelská pravidla, logování událostí v při nedostupnosti IoTIM, výpočty na hraně systému a v neposlední řadě možnost perzistence dat v IoTIM nepocházejících z nižších vrstev. Při srovnání existujících IoT platforem z hlediska výzkumných problémů došlo ke zjištění, že neexistuje řešení splňující veškerou funkcionalitu a proto bylo přistoupeno k dalšímu kroku, návrhu nové architektury IoT platformy s cílem nabídnout možné řešení stanovených problémů při zachování funkcionality existujících platforem.

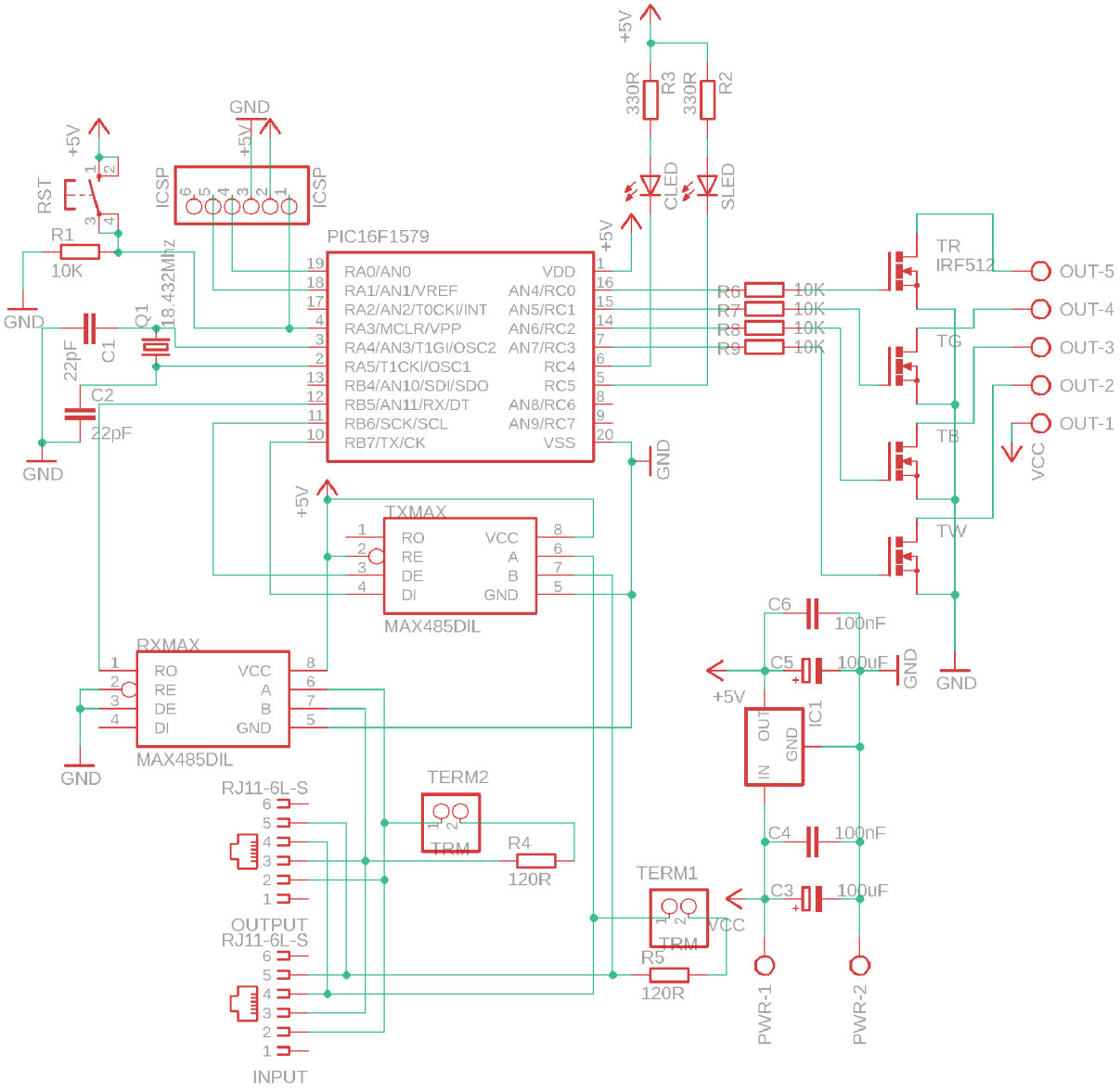
Nové řešení sdílí obdobnou fyzickou architekturu s existujícími platformami s rozdílem nahrazení protokolové brány subsystémem a koncových zařízení nody. Nody umožňují realizovat hardware kombinující odlišné senzory a aktuátory na jednom fyzickém zařízení. Subsystémy díky implementaci prostředků pro dynamické zavádění a obsluhu výpočetních modulů, možnosti provádět uživatelsky definovaná pravidla při absenci spojení s IoTIM a logovat všechny nastalé události výrazně rozšiřují existující implementace bran. IoTIM architektura je implementována jako sada kontejnerizovaných mikroslužeb a aplikací s cílem umožnit škálování od běžně dostupného stolního počítače až po výkonné multiprocesorové serverové systémy. Lze tak realizovat komplexní projekty od prvotní fáze sběru surových senzorických dat, přes využití IoTIM k implementaci, testování a nasazení algoritmů až po využití principů edge computingu ke snížení datové zátěže a latence. Díky systému oprávnění IoTIM je možné vytvářet klientské role s přesně definovaným rozsahem přístupu do systému a povolit klientům pouze nezbytné informace ke splnění jejich záměru.

Představená ukázková implementace IoTIM, subsystému a nodů je již aktivně využívána ve výzkumných projektech. Je ovšem ukázáno, že navrženou architekturu lze implementovat s využitím alternativních technologií a to jak aplikačních tak komunikačních.

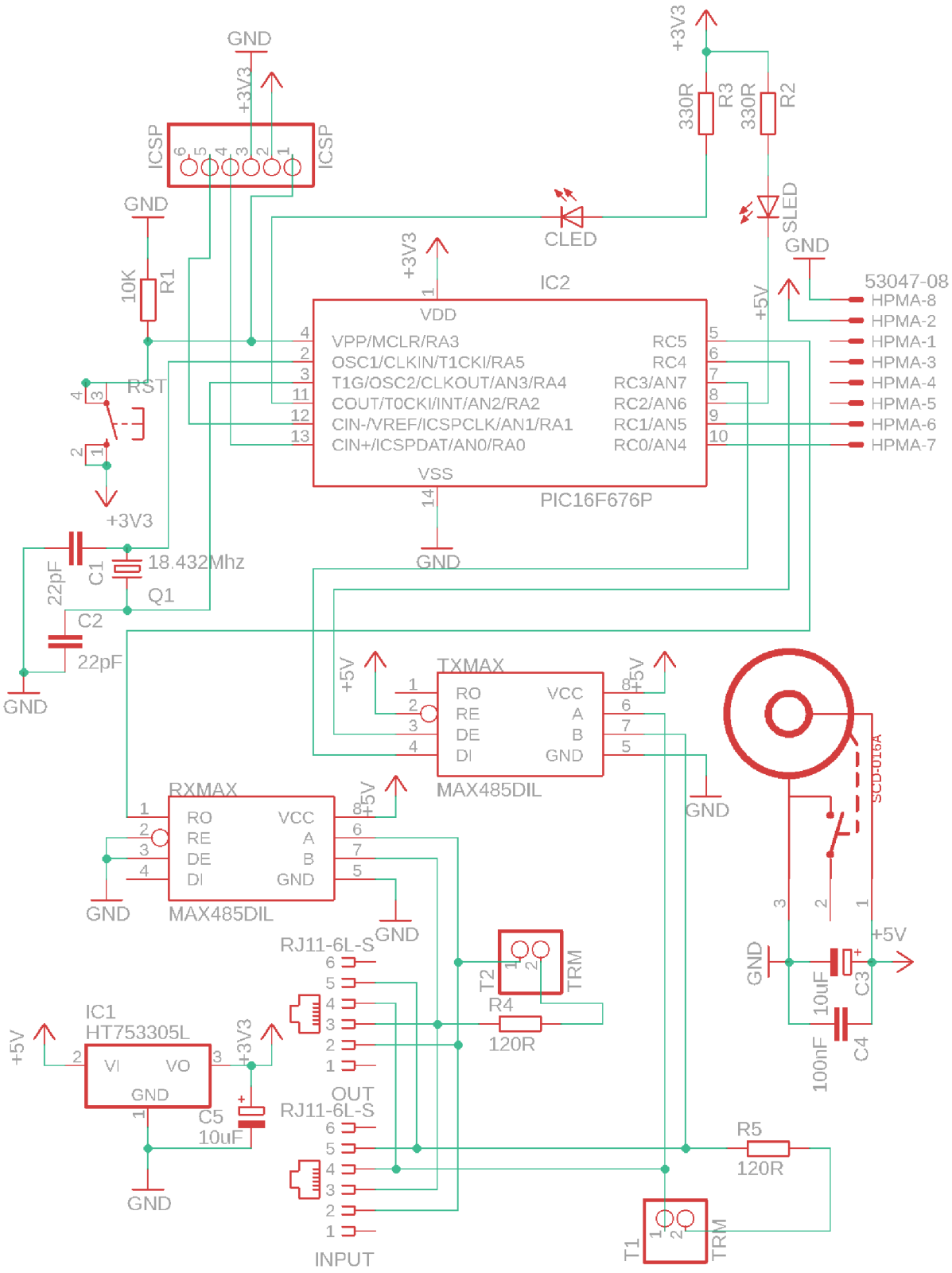
Přílohy



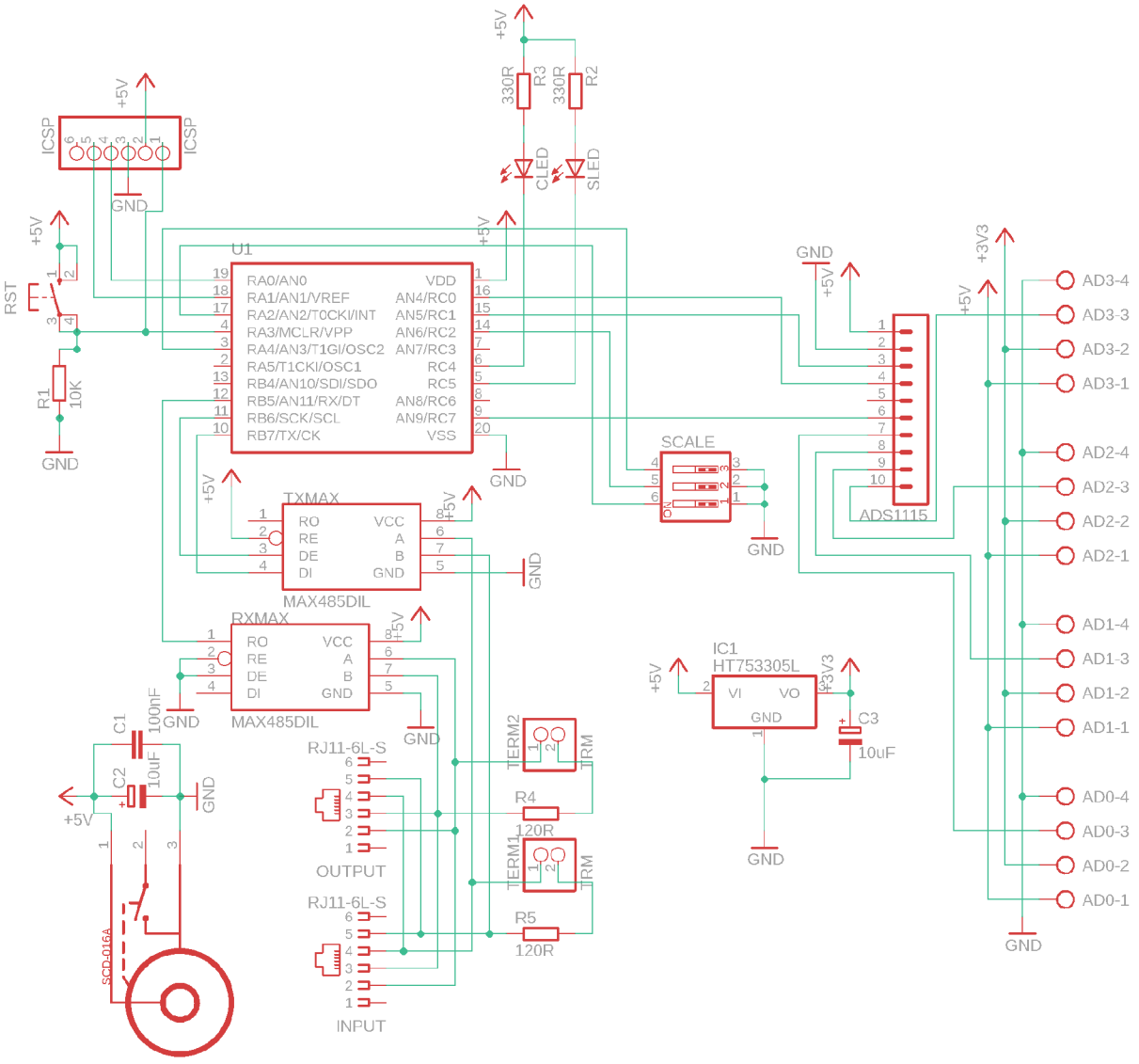
Obrázek 1: Schéma zapojení nodu s relé [autor]



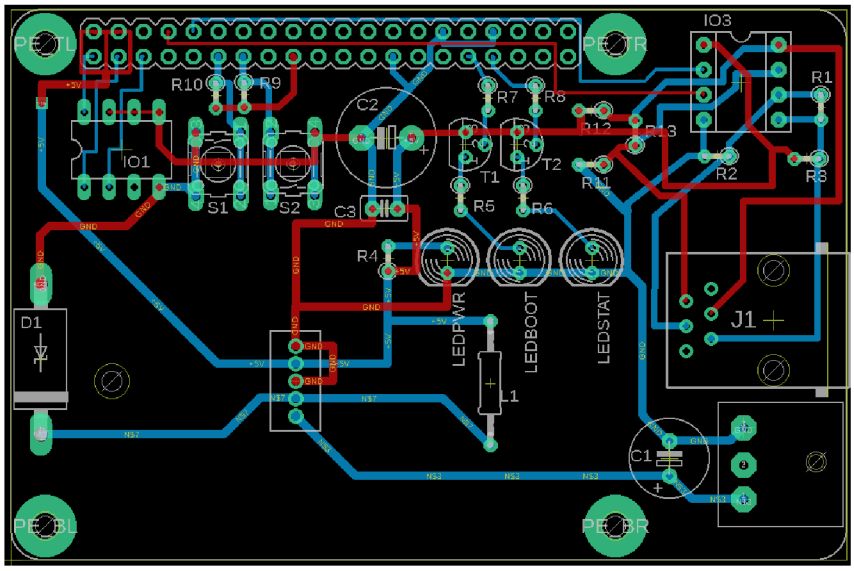
Obrázek 2: Schéma zapojení nodu pro buzení LED pásků [autor]



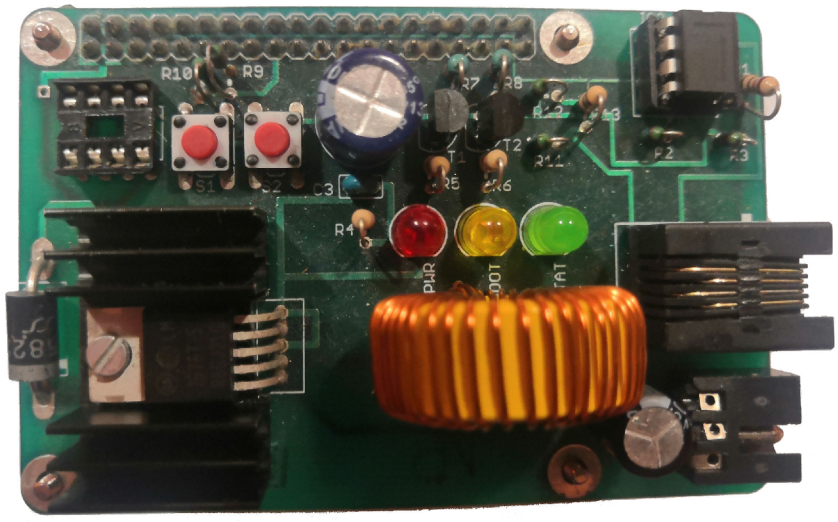
Obrázek 3: Schéma zapojení nodu pro měření PM částic [autor]



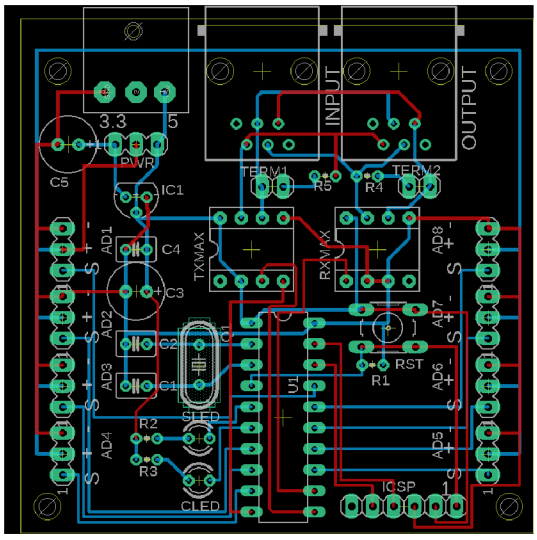
Obrázek 4: Schéma zapojení nodu s převodníkem ADS1115 [autor]



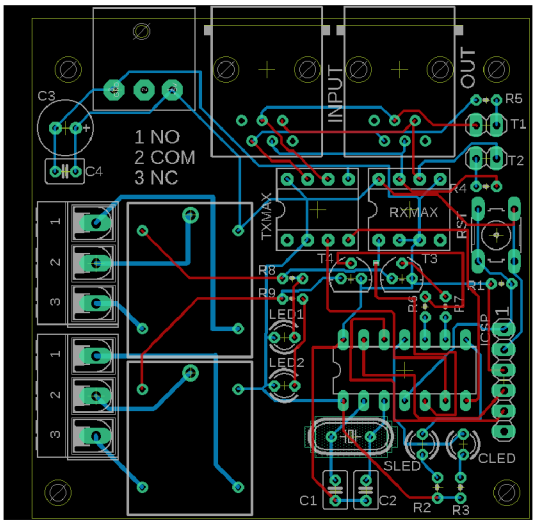
Obrázek 5: Výkres plošného spoje subsystému [autor]



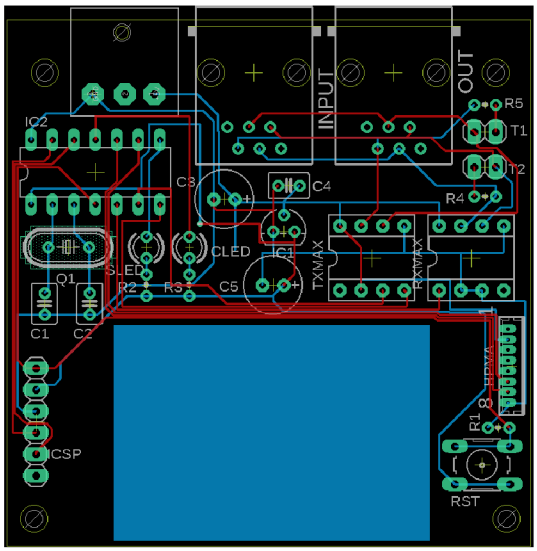
Obrázek 6: Fotografie subsystému [autor]



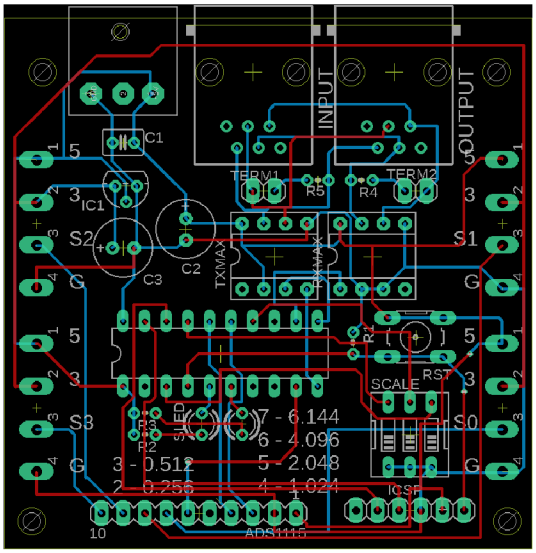
(a) Standardní nod



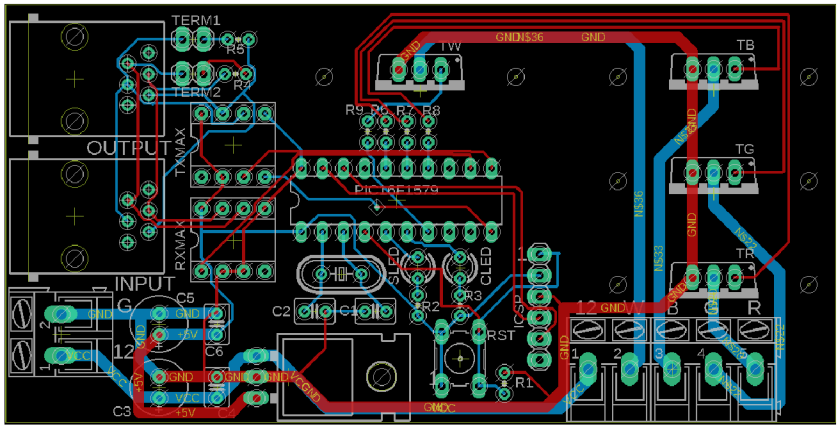
(b) Nod s relé



(c) PM senzor nod

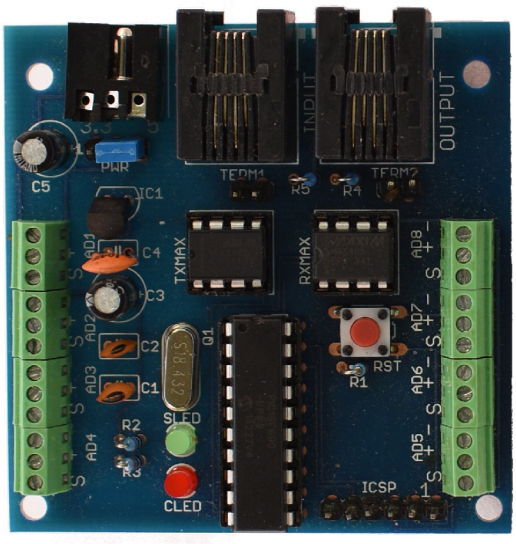


(d) ADC nod

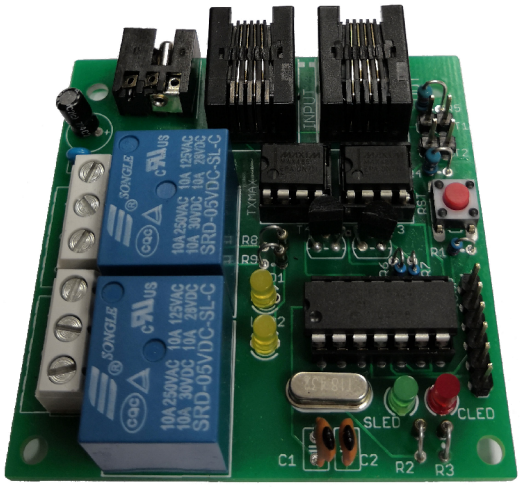


(e) PWM nod

Obrázek 7: Výkresy plošných spojů nodů [autor]

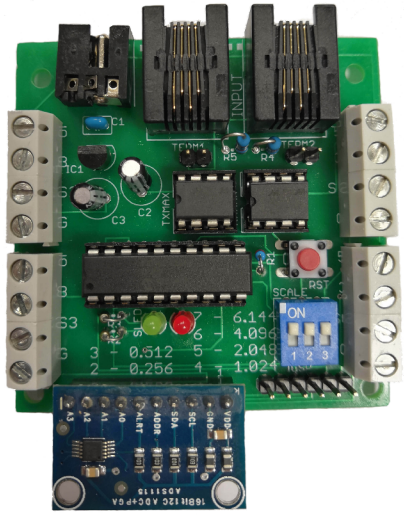


(a) Standardní nod

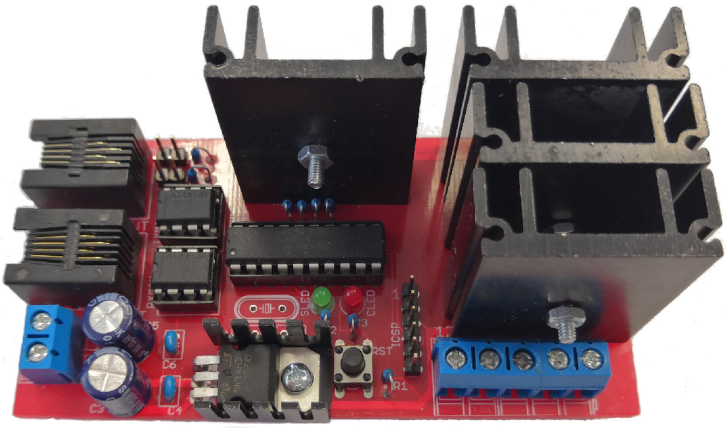


(b) Nod s relé

(c) PM senzor nod



(d) ADC nod



(e) PWM nod

Obrázek 8: Fotografie nodů [autor]

Literatura

[1] Big iot data analytics: Architecture, opportunities, and open research challenges. *IEEE Access 5* (2017), 5247–5261.

[2] ABDUROHMAN, M., HERUTOMO, A., SURYANI, V., ELMANGOUSH, A., AND MAGEDANZ, T. Mobile tracking system using openmtc platform based on event driven method. *38th Annual IEEE Conference on Local Computer Networks - Workshops* (2013).

[3] ADELANTADO, F., VILAJOSANA, X., TUSET-PEIRO, P., MARTINEZ, B., MELIA-SEGUI, J., AND WATTEYNE, T. Understanding the limits of lorawan. *IEEE Communications Magazine 55*, 9 (2017), 34–40.

[4] ADJIH, C., BACCELLI, E., FLEURY, E., HARTER, G., MITTON, N., NOEL, T., PISSARD-GIBOLLET, R., SAINT-MARCEL, F., SCHREINER, G., AND VANDAELE, J. E. A. Fit iot-lab: A large scale open experimental iot testbed. *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)* (2015).

[5] AGGARWAL, R., AND DAS, M. L. Rfid security in the context of"internet of things". In *Proceedings of the First International Conference on Security of Internet of Things* (2012), pp. 51–56.

[6] AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., ALEDHARI, M., AND AYYASH, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials 17*, 4 (2015), 2347–2376.

[7] AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., ALEDHARI, M., AND AYYASH, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials 17*, 4 (2015), 2347–2376.

[8] AL-QASEEMI, S. A., ALMULHIM, H. A., ALMULHIM, M. F., AND CHAUDHRY, S. R. Iot architecture challenges and issues: Lack of standardization. *2016 Future Technologies Conference (FTC)* (2016).

[9] ALEKSY, M., TROOST, M., SCHEINHARDT, F., AND ZANK, G. T. Utilizing hololens to support industrial service processes. *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)* (2018).

[10] AMARAL, M., POLO, J., CARRERA, D., MOHOMED, I., UNUVAR, M., AND STEINDER, M. Performance evaluation of microservices architectures using containers. *2015 IEEE 14th International Symposium on Network Computing and Applications* (2015).

[11] ARIELY, D. *Predictably irrational.* HarperCollins, 2010.

[12] ARLOW, J., NEUSTADT, I., AND KISZKA, B. *UML 2 a unifikovaný proces vývoje aplikací*, 2 ed. Computer Press, 2007.

[13] ASHTON, K. J. That 'internet of things' thing.

[14] ATZENI, A., LYLE, J., AND FAILY, S. Developing secure, unified, multi-device, and multi-domain platforms. *Application Development and Design* (2018), 539–564.

[15] BADENHOP, C. W., RAMSEY, B. W., MULLINS, B. E., AND MAILLOUX, L. O. Extraction and analysis of non-volatile memory of the zw0301 module, a z-wave transceiver. *Digital Investigation 17* (2016), 14–27.

[16] BAUER, M., BOUSSARD, M., BUI, N., CARREZ, F., JARDAK, C., DE LOOG, J., MAGERKURTH, C., MEISSNER, S., NETTSTRATER, A., OLIVEREAU, A., TOMA, M., JOACHIM, W., STEFA, J., AND SALINAS, A. Internet of things – architecture iot-a deliverable d1.5 – final architectural reference model for the iot v3.0, 2013.

[17] BERNSTEIN, D. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing 1*, 3 (2014), 81–84.

[18] BONOMI, F., MILITO, R., ZHU, J., AND ADDEPALLI, S. Fog computing and its role in the internet of things. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12* (2012).

[19] BÁRTA, J. Diskuze o open-source licensích, 2018.

[20] CHAQFEH, M. A., AND MOHAMED, N. Challenges in middleware solutions for the internet of things. *2012 International Conference on Collaboration Technologies and Systems (CTS)* (2012).

[21] COLAKOVIC, A., AND HADZIALIC, M. Internet of things (iot): A review of enabling technologies, challenges, and open research issues. *Computer Networks 144* (2018), 17–39.

[22] CVAR, N., TRILAR, J., KOS, A., VOLK, M., AND STOJMENOVA DUH, E. The use of iot technology in smart cities and smart villages: Similarities, differences, and future prospects. *Sensors 20*, 14 (2020), 3897.

[23] DA CRUZ, M. A. A., RODRIGUES, J. J. P. C., AL-MUHTADI, J., KOROTAEV, V. V., AND DE ALBUQUERQUE, V. H. C. A reference model for internet of things middleware. *IEEE Internet of Things Journal 5*, 2 (2018), 871–883.

[24] DALWADI, N., AND PADOLE, M. An insight into time synchronization algorithms in iot. *Data, Engineering and Applications* (2019), 285–296.

[25] DARROUDI, S. M., AND GOMEZ, C. Bluetooth low energy mesh networks: A survey. *Sensors 17*, 7 (2017), 1467.

[26] DAVIS, G. 2020: Life with 50 billion connected devices. *2018 IEEE International Conference on Consumer Electronics (ICCE)* (2018).

[27] DE JONG, J., AND MANSFIELD, E. Math.js: An advanced mathematics library for javascript. *Computing in Science & Engineering 20*, 1 (2018), 20–32.

[28] DESRUELLE, H., ISENBERG, S., BOTSIKAS, A., VERGORI, P., AND GIELEN, F. Accessible user interface support for multi-device ubiquitous applications: architectural modifiability considerations. *Universal Access in the Information Society 15*, 1 (2016), 5–19.

[29] DORSEMAINE, B., GAULIER, J.-P., WARY, J.-P., KHEIR, N., AND URIEN, P. Internet of things: A definition & taxonomy. *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies* (2015).

[30] DRAGONI, N., GIALLORENZO, S., LAFUENTE, A. L., MAZZARA, M., MONTESI, F., MUSTAFIN, R., AND SAFINA, L. Microservices: Yesterday, today, and tomorrow. *Present and Ulterior Software Engineering* (2017), 195–216.

[31] D'SILVA, G. M., KHAN, A., GAURAV, AND BARI, S. Real-time processing of iot events with historic data using apache kafka and apache spark with dashing framework. *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)* (2017).

[32] DURAIRAJAN, R., MANI, S. K., BARFORD, P., AND SOMMERS, J. Mntp: Enhancing time synchronization for mobile devices. *Proceedings of the 2016 ACM on Internet Measurement Conference - IMC '16* (2016).

[33] DURKOP, L., CZYBIK, B., AND JASPERNEITE, J. Performance evaluation of m2m protocols over cellular networks in a lab environment. *2015 18th International Conference on Intelligence in Next Generation Networks* (2015).

[34] ELMANGOUSH, A. Openmtc platform a generic m2m communication platform.

[35] EVANS, J. M., O'NEIL, J. T., LITTLE, J. L., ALBUS, J. S., BARBERA, A. J., AND FIFE, D. W. *Standards for Computer Aided Manufacturing*. U.S. Department of Commerce, 1976.

[36] FALESSI, D., CANTONE, G., KAZMAN, R., AND KRUCHTEN, P. Decision-making techniques for software architecture design. *ACM Computing Surveys 43*, 4 (2011), 1–28.

[37] FLEISCH, E. What is the internet of things? an economic perspective. *Auto-ID Labs White Paper* (2010).

[38] FRANCESCO, P. D., MALAVOLTA, I., AND LAGO, P. Research on architecting microservices: Trends, focus, and potential for industrial adoption. *2017 IEEE International Conference on Software Architecture (ICSA)* (2017).

[39] FUHRHOP, C., LYLE, J., AND FAILY, S. The webinos project. *Proceedings of the 21st international conference companion on World Wide Web - WWW '12 Companion* (2012).

[40] GARCÍA-SIDRO, P., NAVAL, E., RIVERA, C. M., BONNIN-VILAPLANA, M., GARCIA-RIVERO, J. L., HERREJÓN, A., DE MOLINA, R. M., MARCOS, P. J., MAYORALAS-ALISES, S., ROS, J. A., VALLE, M., ESQUINAS, C., BARRECHEGUREN, M., AND MIRAVITLLES, M. The CAT (COPD assessment test) questionnaire as a predictor of the evolution of severe COPD exacerbations. *Respiratory Medicine 109*, 12 (dec 2015), 1546–1552.

[41] GOMATHINAYAGAM, P., AND JAYANTHY, S. Implementation of mesh network using bluetooth low energy devices. *Lecture Notes in Electrical Engineering 446* (2017), 205–213.

[42] GUOQIANG, S., YANMING, C., CHAO, Z., AND YANXU, Z. Design and implementation of a smart iot gateway. *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing* (2013).

[43] GUPTA, A., AND JHA, R. K. A survey of 5g network: Architecture and emerging technologies. *IEEE Access 3* (2015), 1206–1232.

[44] GUPTA, A., AND MUKHERJEE, N. Can the challenges of iot be overcome by virtual sensors. *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (2017).

[45] GUTH, J., BREITENBÜCHER, U., FALKENTHAL, M., FREMANTLE, P., KOPP, O., LEYMANN, F., AND REINFURT, L. *A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences*. Springer, 2018, pp. 81–101.

[46] HERUTOMO, A., ABDUROHMAN, M., SUWASTIKA, N. A., PRABOWO, S., AND WIJIUTOMO, C. W. Forest fire detection system reliability test using wireless sensor network and openmtc communication platform. *2015 3rd International Conference on Information and Communication Technology (ICoICT)* (2015).

[47] HILLAR, G. C. *MQTT Essentials - A Lightweight IoT Protocol*, 1 ed. Packt Publishing, 2017.

[48] IRMAK, E., AND BOZDAL, M. Internet of things (iot): The most up-to-date challenges, architectures, emerging trends and potential opportunities. *International Journal of Computer Applications 179*, 40 (2018), 20–27.

[49] IVANOV, K. *KVM Virtualization Cookbook*, 1 ed. Packt Publishing, 2017.

[50] JALAL, A., QUAID, M. A. K., AND SIDDUQI, M. A. A triaxial acceleration-based human motion detection for ambient smart home system. *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)* (2019).

[51] JEZBERA, D., KRIZ, J., STUDNICKA, F., AND SEBA, P. Unobtrusive monitoring of blood pressure variability and pulse wave velocity. *Recent Advances in Energy, Environment, Biology and Ecology* (2018), 124 – 127.

[52] JI, W., XU, J., QIAO, H., ZHOU, M., AND LIANG, B. Visual iot: Enabling internet of things visualization in smart cities. *IEEE Network 33*, 2 (2019), 102–110.

[53] JUNG, D., ZHANG, Z., AND WINSLETT, M. Vibration analysis for iot enabled predictive maintenance. *2017 IEEE 33rd International Conference on Data Engineering (ICDE)* (2017).

[54] JUNG, D., ZHANG, Z., AND WINSLETT, M. Vibration analysis for iot enabled predictive maintenance. *2017 IEEE 33rd International Conference on Data Engineering (ICDE)* (2017).

[55] KAMILARIS, A., GAO, F., PRENAFETA-BOLDU, F. X., AND ALI, M. I. Agri-iot: A semantic framework for internet of things-enabled smart farming applications. *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)* (2016).

[56] KANAWADAY, A., AND SANE, A. Machine learning for predictive maintenance of industrial machines using iot sensor data. *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)* (2017).

[57] KANG, B., KIM, D., AND CHOO, H. Internet of everything: A large-scale autonomic iot gateway. *IEEE Transactions on Multi-Scale Computing Systems 3*, 3 (2017), 206–214.

[58] KANG, B., KIM, D., AND CHOO, H. Internet of everything: A large-scale autonomic iot gateway. *IEEE Transactions on Multi-Scale Computing Systems 3*, 3 (2017), 206–214.

[59] KIM, H., MA, X., AND HAMILTON, B. R. Tracking low-precision clocks with time-varying drifts using kalman filtering. *IEEE/ACM Transactions on Networking 20*, 1 (2012), 257–270.

[60] KIM, J.-Y., LIU, N., TAN, H.-X., AND CHU, C.-H. Unobtrusive monitoring to detect depression for elderly with chronic illnesses. *IEEE Sensors Journal 17*, 17 (2017), 5694–5704.

[61] KORKUA, S., JAIN, H., LEE, W.-J., AND KWAN, C. Wireless health monitoring system for vibration detection of induction motors. *2010 IEEE Industrial and Commercial Power Systems Technical Conference - Conference Record* (2010).

[62] KRAMP, T., MEISSNER, S., KRANENBURG, R., LANGE, S., FIEDLER, M., BAUER, M., AND BASSI, A. *Enabling Things to Talk*, 1 ed. Springer-Verlag GmbH, 2013.

[63] KUSEK, M. Internet of things: Today and tomorrow. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (2018).

[64] LANZA, J., SOTRES, P., SÁNCHEZ, L., GALACHE, J. A., SANTANA, J. R., GUTIÉRREZ, V., AND MUÑOZ, L. Managing large amounts of data generated by a smart city internet of things deployment. *International Journal on Semantic Web and Information Systems 12*, 4 (2016), 22–42.

[65] LAURIDSEN, M., NGUYEN, H., VEJLGAARD, B., KOVACS, I. Z., MOGENSEN, P., AND SORENSEN, M. Coverage comparison of gprs, nb-iot, lora, and sigfox in a 7800 km^2 area. *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)* (2017).

[66] LEA, P. *Internet of Things for Architects*, 1 ed. Packt publishing Ltd., 2018.

[67] LEE, Y.-H., AND NAIR, S. A smart gateway framework for iot services. *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (2016).

[68] LUCERI, L., CARDOSO, F., PAPANDREA, M., GIORDANO, S., BUWAYA, J., KUNDIG, S., ANGELOPOULOS, C. M., ROLIM, J., ZHAO, Z., AND CARRERA, J. L. E. A. Vivo: A secure, privacy-preserving, and real-time crowd-sensing framework for the internet of things. *Pervasive and Mobile Computing 49* (2018), 126–138.

[69] MA, K., ABRAHAM, A., YANG, B., AND SUN, R. Intelligent web data management of websocket-based real-time monitoring. *Intelligent Web Data Management: Software Architectures and Emerging Technologies* (2016), 105–124.

[70] MADAKAM, S., LAKE, V., LAKE, V., LAKE, V., ET AL. Internet of things (iot): A literature review. *Journal of Computer and Communications 3*, 05 (2015), 164.

[71] MANI, S. K., DURAIRAJAN, R., BARFORD, P., AND SOMMERS, J. An architecture for iot clock synchronization. *Proceedings of the 8th International Conference on the Internet of Things - IOT '18* (2018).

[72] MANI, S. K., DURAIRAJAN, R., BARFORD, P., AND SOMMERS, J. A system for clock synchronization in an internet of things. *ACM* (2018).

[73] MASSÉ, M. *REST API design rulebook*, 1 ed. O'Reilly, 2012.

[74] MENYCHTAS, A., DOUKAS, C., TSANAKAS, P., AND MAGLOGIANNIS, I. A versatile architecture for building iot quantified-self applications. *2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS)* (2017).

[75] MEYER, S., SPERNER, K., MAGERKURTH, C., AND PASQUIER, J. Towards modeling real-world aware business processes. In *Proceedings of the Second International Workshop on Web of Things* (New York, NY, USA, 2011), WoT '11, Association for Computing Machinery.

[76] MILOSLAVSKAYA, N., NIKIFOROV, A., PLAKSIY, K., AND TOLSTOY, A. Standardization issues for the internet of things. *Advances in Intelligent Systems and Computing* (2019), 328–338.

[77] MIN, D., XIAO, Z., SHENG, B., AND SHIYA, G. Design and implementation of the multi-channel rs485 iot gateway. *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)* (2012).

[78] MINERAUD, J., MAZHELIS, O., SU, X., AND TARKOMA, S. A gap analysis of internet-of-things platforms. *Computer Communications 89-90* (2016), 5–16.

[79] MIORANDI, D., SICARI, S., DE PELLEGRINI, F., AND CHLAMTAC, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks 10*, 7 (2012), 1497–1516.

[80] MORABITO, R., PETROLO, R., LOSCRĚ, V., AND MITTON, N. Legiot: A lightweight edge gateway for the internet of things. *Future Generation Computer Systems 81* (2018), 1–15.

[81] NASAR, M., AND KAUSAR, M. A. Suitability of influxdb database for iot applications. *International Journal of Innovative Technology and Exploring Engineering 8*, 10 (2019), 1850–1857.

[82] NATH, B., REYNOLDS, F., AND WANT, R. Rfid technology and applications. *IEEE Pervasive Computing 5*, 1 (2006), 22–24.

[83] NGU, A. H. H., GUTIERREZ, M., METSIS, V., NEPAL, S., AND SHENG, M. Z. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal* (2016), 1–1.

[84] NOLIN, J., AND OLSON, N. The internet of things and convenience. *Internet Research 26*, 2 (2016), 360–376.

[85] NUGUR, A., PIPATTANASOMPORN, M., KUZLU, M., AND RAHMAN, S. Design and development of an iot gateway for smart building applications. *IEEE Internet of Things Journal* (2018), 1–1.

[86] NURATCH, S. Design and implementation of microcontroller-based platform-independent real-time websocket server for monitoring and control applications. *2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)* (2017).

[87] NURGAZY, M., ZASLAVSKY, A., JAYARAMAN, P. P., KUBLER, S., MITRA, K., AND SAGUNA, S. Cavisap: Context-aware visualization of outdoor air pollution with iot platforms. In *in International Conference on High Performance Computing and Simulation (HPCS)* (2019).

[88] PALAVALLI, A., KARRI, D., AND PASUPULETI, S. Semantic internet of things. *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)* (2016).

[89] PARK, Y., CHOI, J., AND CHOI, J. Conceptual metadata model for sensor data abstraction in iot environments. *IOP Conference Series: Materials Science and Engineering 383* (2018), 012013.

[90] PETROV, A. *Database internals*, 1 ed. O'Reilly Media, 2019.

[91] POPE, C. A., I., AND DOCKERY, D. W. Health effects of fine particulate air pollution: Lines that connect. *Journal of the Air and Waste Management Association 56*, 6 (2006), 709–742.

[92] QIU, T., LIU, X., HAN, M., NING, H., AND WU, D. O. A secure time synchronization protocol against fake timestamps for large-scale internet of things. *IEEE Internet of Things Journal 4*, 6 (2017), 1879–1889.

[93] RAJITH, A., SOKI, S., AND HIROSHI, M. Real-time optimized hvac control system on top of an iot framework. *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)* (2018).

[94] RAYMOND, K. *Reference Model of Open Distributed Processing (RM-ODP): Introduction*. Springer US, Boston, MA, 1995, pp. 3–14.

[95] RAZZAQUE, M. A., MILOJEVIC-JEVRIC, M., PALADE, A., AND CLARKE, S. Middleware for internet of things: A survey. *IEEE Internet of Things Journal 3*, 1 (2016), 70–95.

[96] REESE, G. *Cloud Application Architectures*, 1 ed. O'Reilly Media, Inc., 2009.

[97] ROMER, K., MATTERN, F., DUBENDORFER, T., AND SENN, J. Infrastructure for virtual counterparts of real world objects.

[98] SAHADEVAN, A., MATHEW, D., MOOKATHANA, J., AND JOSE, B. A. An offline online strategy for iot using mqtt. *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)* (2017).

[99] SANCHEZ, L., MUNOZ, L., GALACHE, J. A., SOTRES, P., SANTANA, J. R., GUTIERREZ, V., RAMDHANY, R., GLUHAK, A., KRCO, S., AND THEODORIDIS, E. E. A. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks 61* (2014), 217–238.

[100] SEFLOVA, P., SULC, V., POC, J., AND SPINAR, R. Iqrf wireless technology utilizing iqmesh protocol. *2012 35TH INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS AND SIGNAL PROCESSING (TSP)* (2012), 101–104.

[101] SETHI, P., AND SARANGI, S. R. Internet of things: Architectures, protocols, and applications. *Journal of Electrical and Computer Engineering 2017* (2017), 1–25.

[102] SHI, W., AND DUSTDAR, S. The promise of edge computing. *Computer 49*, 5 (2016), 78–81.

[103] SHINHO, L., HYEONWOO, K., DONG-KWEON, H., AND HONGTAEK, J. Correlation analysis of mqtt loss and delay according to qos level. *The International Conference on Information Networking 2013 (ICOIN)* (2013).

[104] SHRIVASTAV, V., LEE, K. S., WANG, H., AND WEATHERSPOON, H. Globally synchronized time via datacenter networks. *IEEE/ACM Transactions on Networking 27*, 4 (2019), 1401–1416.

[105] SHWETA, S. A., MUKESH, D. P., AND JAGDISH, B. N. Implementation of controller area network (can) bus (building automation). *Communications in Computer and Information Science* (2011), 507–514.

[106] SIEKKINEN, M., HIIENKARI, M., NURMINEN, J. K., AND NIEMINEN, J. How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE* (2012), IEEE, pp. 232–237.

[107] SILVA, J., RODRIGUES, J., AL-MUHTADI, J., RABÊLO, R., AND FURTADO, V. Management platforms and protocols for internet of things: A survey. *Sensors 19*, 3 (2019), 676.

[108] SON, S.-C., KIM, N.-W., LEE, B.-T., CHO, C. H., AND CHONG, J. W. A time synchronization technique for coap-based home automation systems. *IEEE Transactions on Consumer Electronics 62*, 1 (2016), 10–16.

[109] SONG, Z., CARDENAS, A. A., AND MASUOKA, R. Semantic middleware for the internet of things. *2010 Internet of Things (IOT)* (2010).

[110] SOTRES, P., SANTANA, J. R., SANCHEZ, L., LANZA, J., AND MUNOZ, L. Practical lessons from the deployment and management of a smart city internet-of-things infrastructure: The smartsantander testbed case. *IEEE Access 5* (2017), 14309–14322.

[111] STUDNICKA, F., SEBA, P., JEZBERA, D., AND KRIZ, J. Continuous monitoring of heart rate using accelerometric sensors. *2012 35th International Conference on Telecommunications and Signal Processing (TSP)* (2012).

[112] SUMI, L., AND RANGA, V. Sensor enabled internet of things for smart cities. *2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)* (2016).

[113] SUNARDI, S., AFIF, A., AND NOVIYANTO, F. Real time monitoring and irrigation control using the websocket protocol. *Proceedings of the International Conference of Science and Technology for the Internet of Things* (2019).

[114] SURYANI, V., RIZAL, A., HERUTOMO, A., ABDUROHMAN, M., MAGEDANZ, T., AND ELMANGOUSH, A. Electrocardiagram monitoring on openmtc platform. *38th Annual IEEE Conference on Local Computer Networks - Workshops* (2013).

[115] TAVARES BRUSCATO, L., HEIMFARTH, T., AND PIGNATON DE FREITAS, E. Enhancing time synchronization support in wireless sensor networks. *Sensors 17*, 12 (2017), 2956.

[116] TESSARO, L., RAFFALDI, C., ROSSI, M., AND BRUNELLI, D. Lightweight synchronization algorithm with self-calibration for industrial lora sensor networks. *2018 Workshop on Metrology for Industry 4.0 and IoT* (2018).

[117] TIRADO-ANDRÉS, F., ROZAS, A., AND ARAUJO, A. A methodology for choosing time synchronization strategies for wireless iot networks. *Sensors 19*, 16 (2019), 3476.

[118] TORKAMAN, A., AND SEYYEDI, M. Analyzing iot reference architecture models. *International Journal of Computer Science and Software Engineering 5*, 8 (2016), 154–160.

[119] TUDORICA, B. G., AND BUCUR, C. A comparison between several nosql databases with comments and notes. In *2011 RoEduNet International Conference 10th Edition: Networking in Education and Research* (2011), pp. 1–5.

[120] VASISHT, D., KAPETANOVIC, Z., WON, J., JIN, X., CHANDRA, R., SINHA, S., SUDARSHAN, M., AND STRATMAN, S. *FarmBeats: An IoT Platform for Data-Driven Agriculture.* 03 2017.

[121] VEJLGAARD, B., LAURIDSEN, M., NGUYEN, H., KOVACS, I. Z., MOGENSEN, P., AND SORENSEN, M. Coverage and capacity analysis of sigfox, lora, gprs, and nb-iot. *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)* (2017).

[122] VERMA, S., KAWAMOTO, Y., FADLULLAH, Z. M., NISHIYAMA, H., AND KATO, N. A survey on network methodologies for real-time analytics of massive iot data and open research issues. *IEEE Communications Surveys & Tutorials 19*, 3 (2017), 1457–1477.

[123] VILLAMIZAR, M., GARCES, O., OCHOA, L., CASTRO, H., SALAMANCA, L., VERANO, M., CASALLAS, R., GIL, S., VALENCIA, C., AND ZAMBRANO, A. E. A. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (2016).

[124] VOGLER, M., SCHLEICHER, J., INZINGER, C., NASTIC, S., SEHIC, S., AND DUSTDAR, S. Leonore – large-scale provisioning of resource-constrained iot deployments. *2015 IEEE Symposium on Service-Oriented System Engineering* (2015).

[125] WANG, G., KOSHY, J., SUBRAMANIAN, S., PARAMASIVAM, K., ZADEH, M., NARKHEDE, N., RAO, J., KREPS, J., AND STEIN, J. Building a replicated logging system with apache kafka. *Proceedings of the VLDB Endowment 8*, 12 (2015), 1654–1655.

[126] WANG, J., PAN, J., ESPOSITO, F., CALYAM, P., YANG, Z., AND MOHAPATRA, P. Edge cloud offloading algorithms. *ACM Computing Surveys 52*, 1 (2019), 1–23.

[127] WEISER, M. The computer for the 21st century. *Scientific American 265(3)* (1991), 94–104.

[128] WU, L., XU, C., XU, Y.-J., AND WANG, F. Plug-configure-play service-oriented gateway for fast and easy sensor network application development. *Proceedings of the 2nd International Conference on Sensor Networks* (2013).

[129] XU, T., ZHOU, Y., AND ZHU, J. New advances and challenges of fall detection systems: A survey. *Applied Sciences 8*, 3 (2018), 418.

[130] YASUMOTO, K., YAMAGUCHI, H., AND SHIGENO, H. Survey of real-time processing technologies of iot data streams. *Journal of Information Processing 24*, 2 (2016), 195–202.

[131] YF, X., YH, X., MH, S., AND YX, L. The impact of pm2.5 on the human respiratory system. *Journal of Thoracic Disease 8*, 1 (2016), 69 – 74.

[132] ZAMBONELLI, F. Key abstractions for iot-oriented software engineering. *IEEE Software 34*, 1 (2017), 38–45.

[133] ZATWARNICKI, K., PLATEK, M., AND ZATWARNICKA, A. A cluster-based quality aware web system. *Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology – ISAT 2015 – Part II* (2016), 15–24.

Internetové zdroje

[134] “nke watteco sigfox temperature-humidity sensor,” 2020. [Online]. Available: https://partners.sigfox.com/products/temperature-humidity-sensor

[135] “Srg-3352 standard iot gateway system,” 2020. [Online]. Available: https://www.aaeon.com/en/p/iot-gateway-arm-cortex-risc-srg-3352

[136] “Co je saas? software jako služba,” 2020. [Online]. Available: https://azure.microsoft.com/cs-cz/overview/what-is-saas/

[137] “Amazon aws iot,” 2019. [Online]. Available: https://docs.aws.amazon.com/iot/index.html

[138] “Microsoft azure iot hub,” 2019. [Online]. Available: https://docs.microsoft.com/cs-cz/azure/iot-hub/about-iot-hub

[139] “Smartthings,” 2019. [Online]. Available: https://docs.smartthings.com/en/latest/architecture/index.html

[140] “Openmtc,” 2019. [Online]. Available: http://www.open-mtc.org/index.html

[141] “Sitewhere documentation,” 2019. [Online]. Available: https://sitewhere.io/docs/2.1.0/

[142] C. Daffara, “Open source license selection in relation to business models,” 2018. [Online]. Available: https://timreview.ca/article/416

[143] “Metamotionr,” 2019. [Online]. Available: https://mbientlab.com/metamotionr/

[144] “The "only"coke machine on the internet,” 2020. [Online]. Available: https://www.cs.cmu.edu/~coke/history_long.txt

[145] M. Armstrong, “Infographic: All of the data created in 2018 is equal to...” 2020. [Online]. Available: https://www.statista.com/chart/17723/the-data-created-last-year-is-equal-to/

[146] “Buzzword definition - merriam-webster,” 2020. [Online]. Available: https://www.merriam-webster.com/dictionary/buzzword

[147] “Architecture definition - merriam-webster,” 2020. [Online]. Available: https://www.merriam-webster.com/dictionary/architecture

[148] “Sensor definition - merriam-webster,” 2020. [Online]. Available: https://www.merriam-webster.com/dictionary/sensor

[149] “Digital illumination interface,” 2018. [Online]. Available: https://www.digitalilluminationinterface.org/dali/standards.html

[150] “Knx association,” 2018. [Online]. Available: https://www.knx.org/knx-en/for-your-home/index.php

[151] “Sigfox,” 2018. [Online]. Available: https://www.sigfox.com/en

[152] “Lora alliance,” 2018. [Online]. Available: https://www.lora-alliance.org/

[153] “3gpp - a global initiative,” 2018. [Online]. Available: http://www.3gpp.org/

[154] “Bluetooth alliance,” 2018. [Online]. Available: https://www.bluetooth.com/

[155] “Wi-fi alliance,” 2018. [Online]. Available: https://www.wi-fi.org/

[156] “Zigbee,” 2018. [Online]. Available: http://www.zigbee.org/

[157] “Iqrf,” 2018. [Online]. Available: https://www.iqrf.org/

[158] “Z-wave alliance,” 2018. [Online]. Available: https://z-wavealliance.org/

[159] “Coap technology,” 2020. [Online]. Available: https://coap.technology/

[160] “Json,” 2020. [Online]. Available: https://www.json.org/json-en.html

[161] “Xml standard,” 2020. [Online]. Available: https://www.w3.org/TR/2008/REC-xml-20081126/

[162] “Flatbuffers: Overview,” 2018. [Online]. Available: https://google.github.io/flatbuffers/

[163] K. Khare, “Json vs protocol buffers vs flatbuffers – codeburst,” 2019. [Online]. Available: https://codeburst.io/json-vs-protocol-buffers-vs-flatbuffers-a4247f8bda6f

[164] “Protocol buffers,” 2019. [Online]. Available: https://developers.google.com/protocol-buffers/

[165] “grpc,” 2019. [Online]. Available: https://grpc.io/

[166] “Cap’n proto,” 2019. [Online]. Available: https://capnproto.org/

[167] “Apache thrift,” 2019. [Online]. Available: https://thrift.apache.org/

[168] “Improving facebook’s performance on android with flatbuffers,” 2019. [Online]. Available: https://code.fb.com/android/improving-facebook-s-performance-on-android-with-flatbuffers/

[169] “memcached,” 2020. [Online]. Available: https://memcached.org/

[170] “Mongodb,” 2020. [Online]. Available: https://www.mongodb.com/

[171] “Elasticsearch,” 2020. [Online]. Available: https://www.elastic.co/elasticsearch/

[172] “Apache hadoop,” 2020. [Online]. Available: https://hadoop.apache.org/

[173] “neo4j,” 2020. [Online]. Available: https://neo4j.com/

[174] “Redis,” 2020. [Online]. Available: https://redis.io/

[175] “Influxdb - open source time series database,” 2020. [Online]. Available: https://www.influxdata.com/products/influxdb-overview/

[176] “Prometheus,” 2020. [Online]. Available: https://prometheus.io/

[177] “Graphite,” 2020. [Online]. Available: https://graphiteapp.org/

[178] “Dedikované servery - wedos.cz,” 2020. [Online]. Available: https://www.wedos.cz/dedikovane-servery

[179] “Google workspace: Business collaboration tools,” 2020. [Online]. Available: https://workspace.google.com/

[180] “Amazon dynamo,” 2020. [Online]. Available: https://aws.amazon.com/dynamodb/

[181] “Amazon kinesis,” 2020. [Online]. Available: https://aws.amazon.com/kinesis/

[182] “Amazon lambda,” 2020. [Online]. Available: https://aws.amazon.com/lambda/

[183] “Amazon s3,” 2020. [Online]. Available: https://aws.amazon.com/s3/

[184] “Amazon sns,” 2020. [Online]. Available: https://aws.amazon.com/sns/

[185] “Microsoft azure iot device catalog,” 2020. [Online]. Available: https://catalog.azureiotsolutions.com/

[186] N. Transformation, “onem2m - home,” 2020. [Online]. Available: http://www.onem2m.org/

[187] “Openmtc documentation,” 2020. [Online]. Available: http://www.open-mtc.org/doc.html

[188] “Sitewhere,” 2020. [Online]. Available: https://sitewhere.io/en/

[189] “Webinos repository,” 2020. [Online]. Available: https://github.com/webinos

[190] “Fiware,” 2020. [Online]. Available: https://www.fiware.org/

[191] “Openiot,” 2020. [Online]. Available: http://www.openiot.eu/

[192] “Ericsson iot framework,” 2020. [Online]. Available: https://github.com/EricssonResearch/iot-framework-engine

[193] “Thingspeak,” 2020. [Online]. Available: https://thingspeak.com/

[194] “Nodered,” 2020. [Online]. Available: https://nodered.org/

[195] “Telegraf - open source server agent,” 2020. [Online]. Available: https://www.influxdata.com/time-series-platform/telegraf/

[196] “The rocket-fast syslog server - rsyslog,” 2020. [Online]. Available: https://www.rsyslog.com/

[197] “Openzwave,” 2019. [Online]. Available: http://www.openzwave.com/

[198] “Lightning memory-mapped database,” 2018. [Online]. Available: https://symas.com/lmdb/

[199] “Cereal.” [Online]. Available: http://uscilab.github.io/cereal/

[200] “Z-uno,” 2019. [Online]. Available: https://z-uno.z-wave.me/

[201] “Stm32h7 series,” 2021. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html

[202] “Healthy aging in industrial environment,” 2020. [Online]. Available: https://haie.osu.cz/projekt-haie/

Publikace autora v periodiku s přiznaným IF nebo SJR

HORALEK, J., MATYSKA, J., STEPAN, J., VANCL, M., CIMLER, R., AND SOBESLAV, V. Lower layers of a cloud driven smart home system. *Studies in Computational Intelligence 598* (2015), 219–228.

STEPAN, J., CIMLER, R., AND MATYSKA, J. Using ipv6 over bluetooth low energy on low costs platforms. *Journal of Engineering and Applied Sciences 13*, 9 (2018), 6986–6991.

STEPAN, J., MATYSKA, J., CIMLER, R., AND HORALEK, J. Low level communication protocol and hardware for wired sensor networks. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC) 9*, 2-4 (2017), 53–57.

STUDNICKA, F., STEPAN, J., AND SLEGR, J. Low-cost radon detector with low-voltage air-ionization chamber. *Sensors 19*, 17 (2019), 3721.

Publikace autora v indexovaném sborníku konference

SEC, D., CIMR, D., STEPAN, J., CIMLER, R., AND KUHNOVA, J. Automatic address assigning problem in smart homes. In *IEEE World Congress On Computational Intelligence* (2018), IEEE, pp. 1–6.

SEC, D., CIMR, D., STEPAN, J., CIMLER, R., AND KUHNOVA, J. Optimized algorithm for node address assigning in large scale smart automation environment. In *Lecture Notes in Computer Science* (2018), vol. 11056, Springer, pp. 311–321.

STEPAN, J., CIMLER, R., AND KREJCAR, O. Automation system architecture for smart hotel. In *Lecture Notes in Computer Science* (2018), vol. 11056, Springer, pp. 457–466.

STEPAN, J., CIMLER, R., MATYSKA, J., AND KREJCAR, O. Design of universal hardware node board for smart-home automation and the iot. In *Studies in Computational Intelligence* (2018), vol. 769, Springer, pp. 465–475.

STEPAN, J., CIMLER, R., MATYSKA, J., SEC, D., AND KREJCAR, O. Lightweight protocol for m2m communication. In *Lecture Notes in Computer Science* (2017), vol. 10449, Springer, pp. 335–344.

STEPAN, J., DANICEK, M., CIMLER, R., MATYSKA, J., AND KREJCAR, O. Wildlife presence detection using the affordable hardware solution and an ir movement detector. In *Lecture Notes in Computer Science* (2017), vol. 10449, Springer, pp. 345–354.

VOJTAS, P., STEPAN, J., SEC, D., CIMLER, R., AND KREJCAR, O. Voice recognition software on embedded devices. In *Lecture Notes in Computer Science* (2018), vol. 10751, Springer, pp. 642–650.

Shrnutí publikační činnosti autora a účasti na projektech

Shrnutí publikačních výstupů:

	publikací	h-index	citací celkem
WoS	7	1	11
Scopus	11	3	25
Google Scholar	9	4	31

Účast na projektech:

GAČR

2015–2017

GA15-11724S	Člen	DEPIES – Rozhodovací procesy v inteligentních prostředích

MPO

2017				
OPPIK	Člen	Řešení pro úsekové měřidlo	Silponix s.r. o.	Návrh a tvorba HW pro úsekové měřidlo
2018				
OPPIK	Člen	DeltaControl	Trilab group s.r.o.	Návrh a implementace rozšíření základové desky, návrh a tvorba programátoru
2019				
OPPIK	Člen	SmartLab	C2P s.r.o.	Nasazení systému pro sběr senzorických dat
TRIO	Člen	FV40231	ESSENCE LINE, s.r.o.	Návrh platformy pro stanovení bioaktivních látek

TAČR

2017		
GAMA TG02010020	Člen	Zařízení pro detekci živých organismů v uzavřených prostorech
2018		
GAMA TG02010020	Člen	Diagnostický systém monitorování fitness s vizí využití pacienty CHOPN
GAMA TG02010020	Člen	Smart Furniture
Epsilon TH03010448	Člen	Zhotovení moderního stavebnicového systému pro výuku mechatroniky v souladu s výzvou Průmysl 4.0
2019		
GAMA TG02010020	Člen	Zařízení pro autodiagnostiku progrese artrózy a rozvoje femoroacetabulárního impingement syndromu u rizikových skupin
ZÉTA TJ02000155	Člen	Smart Vet
2020		
GAMA TP01010032	Člen	Využití pokročilých metod zpracování signálů ke kontinuálnímu zjišťování stavu mostních konstrukcí
GAMA TP01010032	Člen	Systém pro monitoring progrese Parkinsonovy choroby
GAMA TP01010032	Člen	Hospital bed sensors for COVID diagnosis
GAMA TP01010032	Člen	Zařízení pro monitoring hladiny v uzavřených nádobách
GAMA TP01010032	Člen	Portable Patron
ETA TL03000520	Člen	Chytrá řešení napříč kontinuální péčí o seniory

Specifický výzkum

2016	Člen	Chytrá řešení ve všudypřítomných počítačových prostředích
2017	Člen	Modely a algoritmy pro optimalizační problémy v podmínkách neurčitosti III
2017	Člen	Informační a znalostní management a kognitivní věda v cestovním ruchu I
2017	Člen	Chytrá řešení ve všudypřítomných počítačových prostředích
2018	Člen	Socio-ekonomické modely a autonomní systémy
2018	Člen	Informační a znalostní management a kognitivní věda v cestovním ruchu II
2018	Člen	Počítačové sítě pro cloud, distribuované výpočty a internet věcí
2019	Člen	Informační a znalostní management a kognitivní věda v cestovním ruchu III
2019	Člen	Socio-ekonomické modely a autonomní systémy II
2019	Člen	Počítačové sítě pro cloud, distribuované výpočty a internet věcí II