Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Sandi Šemrov

Samodejna zaznava in nastavitev senzorjev v internetu stvari

MAGISTRSKO DELO

ŠTUDIJSKI PROGRAM DRUGE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: prof. dr. Marko Bajec

Ljubljana, 2016

Avtorske pravice. Rezultati magistrskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov magistrskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Zahvala

Zahvaljujem se mentorju prof. dr. Marku Bajcu in as. dr. Slavku Žitniku za usmerjanje in pomoč pri izdelavi pričujočega dela. Hvala staršem za podporo tekom študija. Za lektoriranje dela se zahvaljujem sestri Sabini.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Opis problema 2
1.2 Prispevki dela 2
1.3 Raziskovalni pristop 3
1.4 Pregled sorodnih del 3
1.5 Organiziranost dela 5

2 Tehnologije in koncepti **7**

2.1 M2M in internet stvari (IoT) 7
2.2 oneM2M 8
2.3 OM2M 13
2.4 Bluetooth 20
2.5 Z-Wave 26

3 Knjižnice, orodja in naprave **29**

3.1 Bluecove 29
3.2 Zwave4J 30
3.3 Orodja in knjižnice za BLE 31
3.4 Naprave 35

4 Razširitev OM2M **37**
4.1 Implementacija vmesnikov 37
4.2 Integracija s platformo OM2M 48

5 Testiranje vtičnikov **59**
5.1 Priprava okolja 59
5.2 Preizkus vtičnika za Bluetooth 59
5.3 Preizkus vtičnika za BLE 66
5.4 Preizkus vtičnika za Z-Wave 71

6 Semantika **75**
6.1 Ontologija 75
6.2 Ontologija v oneM2M 77
6.3 Semantika v OM2M 83

7 Sklepni del **85**

Apendiks **91**
A Specifikacija oneM2M 93
B Specifikacija GATT 119

Seznam uporabljenih kratic

kratica	angleško	slovensko
IoT	Internet of Things	internet stvari
M2M	Machine-to-Machine	komunikacija med napravami
API	Application Programming Interface	programski vmesnik
WSDL	Web Service Description Language	opisni jezik spletnih storitev
XML	Extensible Markup Language	razširljivi označevalni jezik
MAC	Media Access Control	krmiljenje dostopa do medija
URL	Uniform Resource Locator	enotni naslov vira
URI	Uniform Resource Identifier	enotni identifikator vira
ATT	Attribute Protocol	atributni protokol
GATT	Generic Attributes	splošni atributi
BLE	Bluetooth Low Energy	nizkoenergijski Bluetooth
HFP	Hands-Free Profile	profil prostoročne uporabe
BLP	Blood Pressure Profile	profil za merjenje krvnega tlaka
SPP	Serial Port Profile	profil serijske povezave
UUID	Universally Unique Identifier	globalno unikaten identifikator
CSE	Common Services Entity	entiteta skupnih storitev
AE	Application Entity	aplikacijska entiteta
MN	Middle Node	vmesno vozlišče
IN	Infrastructure Node	infrastrukturno vozlišče
ASN	Application Service Node	vozlišče storitev aplikacije

IPE	Interworking Proxy Application Entity	vzajemno delujoča aplikacijska entiteta
SP	Service Provider	ponudnik storitev
RDF	Resource Desciption Framework	ogrodje za opis vira
HTTP	HyperText Transfer Protocol	protokol za izmenjavo hiperteksta
IP	Internet Protocol	internetni protokol
CoAP	Constrained Application Protocol	omejeni aplikacijski protokol
OWL	Web Ontology Language	jezik za ontologije
REST	representational state transfer	prenos stanj kot predstavitev
SOA	Service Oriented Architecture	storitveno orientirana arhitektura
IEEE	Institute of Electrical and Electronics Engineers	svetovno združenje inženirjev elektronike in elektrotehnike

Povzetek

Naslov: Samodejna zaznava in nastavitev senzorjev v internetu stvari

Pomanjkanje standardizacije na področju M2M povzroča nezmožnost povezovanja in slabšo interoperabilnost med napravami v internetu stvari. V delu je preučen globalni standard oneM2M, ki razvija standarde na področju M2M in specificira skupno storitveno plast, ki bo napravam M2M omogočala komunikacijo na globalni ravni. Preučena je platforma OM2M, ki temelji na specifikacijah oneM2M. OM2M je odprtokodna platforma za komunikacijo M2M, ki je razvita na razširljivi platformi OSGi. Prikazana je razširitev platforme OM2M z vtičniki treh brezžičnih tehnologij, ki so popularne v internetu stvari: Bluetooth, Bluetooth Low Energy in Z-Wave. Vtičniki so razviti na način, ki omogoča samodejno zaznavo naprav v posameznem komunikacijskem protokolu. Prikazani so koraki, potrebni za razširjanje platforme OM2M s tehnologijo, ki ni del sistema oneM2M. Opisana je semantika, ki jo določa sistem oneM2M, in način poteka prevedbe ontologije zunanjega sistema v ontologijo sistema oneM2M.

Ključne besede: internet stvari, M2M, oneM2M, OM2M, Bluetooth, Bluetooth Low Energy, Z-Wave, vtičnik.

Abstract

Title: Automatic detection and configuration of sensors in IoT

Lack of Machine-to-Machine standardization is causing the inability of connecting and poor interoperability between devices in IoT. We examine a global standard oneM2M for M2M. It defines a common services layer, which is going to enable communications between M2M devices on a global scale. We study OM2M platform, which is based on oneM2M standard. In this paper, we show how to extend the OM2M platform with plugins of three wireless technologies that are common in IoT: Bluetooth, Bluetooth Low Energy and Z-Wave. Plugins are developed in such a way, that they enable automatic detection in every chosen communication protocol. We show the steps needed to extend the OM2M platform with a technology, which is not a part of an oneM2M system. We explain semantics in oneM2M system and procedures to translate ontologies of non-oneM2M systems to oneM2M ontology.

Keywords: IoT, M2M, oneM2M, OM2M, Bluetooth, Bluetooth Low Energy, Z-Wave, plugin.

Poglavje 1

Uvod

Internet stvari (ang. *Internet of Things* – IoT) predstavlja enega najpomembnejših tehnoloških trendov današnjega časa. Temelji na pričakovanju, da bo v prihodnosti možno v internet povezati praktično sleherni objekt in da bodo tako povezani objekti znali med seboj komunicirati ter se v določenih okoliščinah tudi avtonomno odločati. Eden izmed ključnih konceptov, ki se za to uporablja, je machine-to-machine (krajše M2M). M2M predstavlja komunikacijo med dvema napravama, ki lahko poteka preko kakršnegakoli komunikacijskega medija.

Trg naprav, ki so se sposobne priključiti v internet, je izjemno velik. Naprave so po večini specializirane in zaradi različnih proizvajalcev in sektorjev uporabe temeljijo na različnih tehnologijah ter uporabljajo različne komunikacijske protokole. To povzroča številne težave pri povezovanju in onemogoča interoperabilnost [24], ki pa je ključnega pomena za realizacijo mnogih scenarijev v internetu stvari.

Dandanes se vse več industrij zanaša na vertikalne, specifične rešitve M2M, ki vključujejo specifično strojno in programsko opremo. Ta je ponavadi izdelana za točno določene industrijske storitve. Izdelava, nameščanje in vzdrževanje takšnih rešitev zahteva veliko denarja. Zato je velika potreba po oblikovanju skupne horizontalne platforme, ki bi bila skupna vsem vertikalnim industrijam in bi omogočala lažje razvijanje rešitev M2M [25].

Zaradi pomanjkanja standardizacije na področju M2M, so se pri evropskem inštitutu za standardizacijo na telekomunikacijskem področju (Eurpoean Telecommunications Standards Institue) odločili, da izdajo nabor specifikacij za enotno storitveno platformo M2M [1], ki je kasneje prerasla v globalni standard oneM2M.

1.1 Opis problema

V mnogih scenarijih IoT pričakujemo, da se bodo naprave samodejno zaznale in začele komunicirati med seboj. Težava je v raznovrstnosti komunikacijskih protokolov, saj je potrebno za vsak protokol posebej razviti rešitev, ki omogoča samodejno povezovanje z napravami. Rešitve za posamezne protokole se omejijo na specifične platforme IoT, za katere pa vemo, da ne sledijo standardom. Zato smo v tem delu za osnovo vzeli platformo OM2M, ki sledi standardu oneM2M, in preizkusili, kako bi lahko samodejno zaznavali naprave, ki komunicirajo po nekaterih najbolj uporabljenih protokolih.

1.2 Prispevki dela

Glavni prispevek magistrskega dela je razvoj razširitve za odprtokodno platformo OM2M, ki omogoča samodejno zaznavo naprav. Platformo smo nadgradili z moduli nekaj najpopularnejših komunikacijskih protokolov brezžičnih osebnih omrežij (ang. *Wireless Personal Area Network*, WPAN). Za implementacije vseh izbranih brezžičnih tehnologij smo uporabili odprtokodne knjižnice. Moduli znajo preiskovati okolico na način, kot ga uporabljajo izbrani protokoli in zaznati nov senzor oz. napravo v prostoru. Znajo se povezati z zaznano napravo in z nje brati podatke, ki jih naprava pošilja.

V delu je prikazana razširitev platforme OM2M z vtičniki, ki omogočajo uporabo protokolov, ki niso del standarda oneM2M oz. niso narejeni po specifikacijah oneM2M (t. i. elementi *non-oneM2M*).

Prikazan je tudi preizkus možnosti samodejne zaznave naprav, ki komuni-

cirajo po izbranih protokolih.

Predstavimo ontologijo standarda oneM2M in prevedbo med ontologijami non-oneM2M in oneM2M.

1.3 Raziskovalni pristop

Namestili in preučili smo odprtokodno platformo OM2M ter njeno delovanje. Preučili smo literaturo o standardu oneM2M, komunikacijah M2M ter omrežjih tipa WPAN. Na osnovi analize smo izbrali protokole, ki se danes pogosto uporabljajo v okviru WPAN: Bluetooth, Bluetooth Low Energy in Z-Wave. Za izbrane protokole smo poiskali brezplačne odprtokodne implementacije in jih prilagodili za potrebe razširitve OM2M. Za platformo OM2M smo za vsako od izbranih brezžičnih tehnologij razvili vtičnike za avtomatsko zaznavanje naprav v prostoru. Razvite vtičnike smo testirali. Postopek evaluacije je opisan v poglavju 5.

1.4 Pregled sorodnih del

Med sorodna dela spadajo dela, ki se nanašajo na komunikacije M2M, interoperabilnost med napravami in sposobnost samodejne zaznave naprav. Iskali smo odprtokodno platformo za komunikacije M2M, ki sledi standardu M2M, omogoča samodejno zaznavo naprav v prostoru in podpira različne brezžične komunikacijske tehnologije.

Avtorja S. in J. Zhang [30] predlagata platformo za komunikacije M2M, ki temelji na programskem jeziku Java EE (*Enterprise Edition*), in sledi konceptu storitveno usmerjene arhitekture SOA (*Service Oriented Architecture*), vendar ne upoštevata nobenega standarda s področja komunikacij M2M.

V članku [6] naslovijo potrebne zahteve in oblikovni vidik platforme M2M, ki omogoča interoperabilnost naprav v velikem obsegu. Platforma bi se uporabljala za rešitve pametnih mest, kjer je potrebna interoperabilnost veliko heterogenih tehnologij. Predlagajo standardne vmesnike ETSI M2M

za povezovanje naprav, vendar ne opišejo zaznavanja naprav.

Članek [14] predstavi platformo M2M, kjer je omrežna arhitektura sestavljena iz naprave M2M, storitvene platforme M2M in uporabnika. V visokonivojski arhitekturi sistema M2M se osredotočajo na funkcionalnosti storitvenega dela. Sistemska arhitektura je narejena po standardu ETSI TC M2M in omogoča razširljivost z omrežji WPAN. Gre za prototipno implementacijo platforme, ki ni na voljo širši javnosti in ne opredeljuje samodejne zaznave naprav.

Avtorji dela [19] za interoperabilnost senzorjev predlagajo metodo, ki uporablja spletne storitve. Interoperabilnost je zahtevana na omrežnem in aplikacijskem nivoju. Na omrežnem nivoju se zagotovi z uporabo protokola IP. Na aplikacijskem nivoju se pričakuje, da razvijalec pozna semantike senzorjev in protokole, po katerih se prenašajo podatki. Zaradi različnih specifikacij senzorjev (različnih proizvajalcev) predlagajo uporabo spletnih storitev, kjer je vsako vozlišče definirano s svojo specifikacijo WSDL (*Web Service Description Language*). Na ta način lahko različne aplikacije berejo oz. pišejo podatke iz/v senzorjev. To delo predlaga uporabo protokola IP za zagotavljanje interoperabilnosti, v našem primeru pa imamo tehnologije, ki ne podpirajo tega protokola.

V delu [4] za integracijo naprav z različno strojno opremo uporabijo protokol CoAP. Njihova rešitev naslavlja senzorje in aktuatorje, ki komunicirajo preko protokolov IPv4, IPv6 ali Zigbee in za izmenjavo podatkov uporabljajo CoAP. Predlagajo in implementirajo tudi mehanizem za samodejno nastavljanje brezžičnih naprav in odkrivanje njihovih zmožnosti zaznavanj. Za zaznavanje novih naprav oz. za odstranjevanje starih naprav se na aplikacijskem nivoju periodično izprašuje razširitveno ploščico. V primeru sprememb se le-te sporočijo po protokolu CoAP. Zaznane naprave se nato javijo še omrežnemu prehodu, ki omogoči dostop do naprav ostalim vozliščem v omrežju. Rešitev ni narejena po standardih za komunikacijo M2M.

1.5 Organiziranost dela

V delu najprej opišemo tehnologije in koncepte, ki so predstavljeni v poglavju 2. Potrebni so za razumevanje rešitev, ki jih poda magistrsko delo. Implementirane rešitve so dosežene s pomočjo orodij, knjižnic in naprav, ki so opisane v poglavju 3. Sledi podrobna predstavitev implementacije vmesnikov in vtičnikov ter integracija s platformo OM2M v poglavju 4. V poglavju 5 testiramo razvite vtičnike na platformi OM2M in prikažemo njihovo delovanje. Na koncu dela v poglavju 6 predstavimo semantiko, ki so jo specificirali za standard oneM2M, in pripadajočo ontologijo.

Poglavje 2

Tehnologije in koncepti

V tem poglavju predstavimo koncepte, ki so temelj tega magistrskega dela. Najprej predstavimo paradigmo interneta stvari in komunikacij v njem, ki jih opisuje M2M. Nato opišemo specifikacije standarda oneM2M, na katerem je zgrajena platforma OM2M, ki smo jo nadgradili. Na koncu opišemo še brezžične tehnologije, ki smo jih uporabili v okviru samodejnega zaznavanja naprav v prostoru.

2.1 M2M in internet stvari (IoT)

Machine to machine ali krajše M2M je izraz, ki opisuje mehanizme, algoritme in tehnologije za žično in/ali brezžično komunikacijo med napravami ali storitvami [3]. M2M uporabi napravo (npr. senzor ali merilec), da izmeri količino (npr. temperaturo ali nivo zaloge), ki se pošlje po žičnem, brezžičnem ali hibridnem omrežju do aplikacije (programske opreme), ki prevede dobljene podatke v uporabne informacije [7].

Internet stvari je omrežje naprav - stvari, ki med sabo komunicirajo in znajo med sabo izmenjevati podatke. Za komunikacijo se v večini uporablja M2M. Najbolj znani primeri IoT/M2M so e-zdravje, inteligentni transportni sistemi, pametna električna omrežja, pametne hiše in pametna mesta. Napoveduje se, da bo do leta 2020 povezanih skoraj 20 milijard naprav po celem svetu [11].

Med naprave oz. *stvari* uvrščamo elektronske naprave, ki zajemajo podatke iz okolice preko senzorjev oz. vplivajo na okolico preko aktuatorjev. Naprave morajo biti povezane v omrežje, da lahko med seboj izmenjujejo podatke in se na njih odzivajo. Med *stvari* uvrščamo tudi avtomobile, zgradbe in druge objekte, ki vsebujejo senzorje in aktuatorje.

Internet stvari omogoča, da z napravami v omrežju upravljamo oddaljeno in tako dobimo tesno integracijo med fizičnimi napravami in računalniškimi sistemi. Na tem temeljijo številne rešitve s področja interneta stvari: pametne stavbe, pametna mesta, pametna električna omrežja itn.

2.2 oneM2M

OneM2M Global Initiative je mednarodni projekt, ustanovljen z namenom globalnega sodelovanja pri izgradnji specifikacije za dostopno neodvisno storitveno plast M2M. Skupna storitvena plast je namenjena vgradnji v različno strojno in programsko opremo, ki bo napravam M2M zagotavljala komunikacijo na globalni ravni [25].

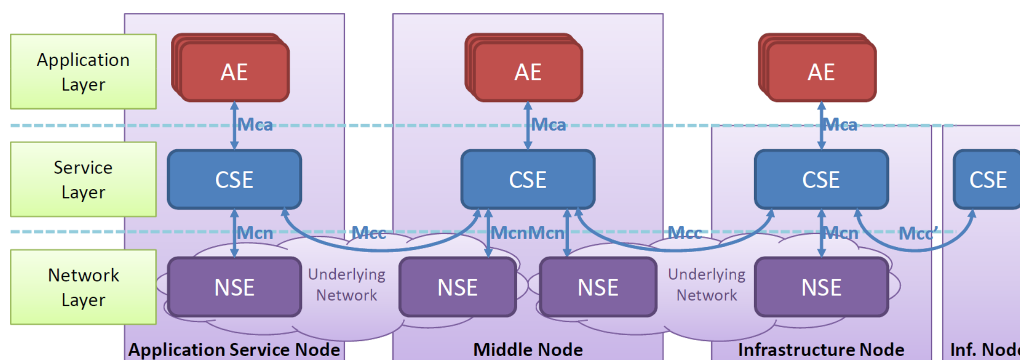
OneM2M se je začel s pobudo sedmih telekomunikacijskih organizacij: japonski *Association of Radio Industries and Businesses* (ARIB) in *Telecommunication Technology Committee* (TTC), ameriški *Alliance for Telecommunications Industry Solutions* (ATIS) in *Telecommunications Industry Association* (TIA), kitajska *China Communications Standards Association* (CCSA), evropska *European Telecommunications Standards Institute* (ETSI) in korejska *Telecommunications Technology Association* (TTA). Njihove družbe članice, ki jih je okrog 270, dejavno prispevajo v oneM2M. Poleg tega so zveze in industrijski forumi *Open Mobile Alliance* (OMA), *Broadband Forum* (BBF), *Continua Health Alliance* in *Home Gateway Initiative* (HGI) postali partnerji oneM2M [25].

Glavni cilj oneM2M je minimiziranje fragmentacije standardov storitvene plasti M2M z združevanjem trenutno izoliranih standardov storitvene plasti M2M in razvijanjem globalnih specifikacij[25].

2.2.1 Specifikacije oneM2M

OneM2M specificira nivojski model sestavljen iz treh nivojev: aplikacijski nivo (ang. *Application Layer*), nivo skupnih storitev (ang. *Common Services Layer*) in nivo omrežnih storitev (ang. *Network Services Layer*).

Na sliki 2.1 je prikazana arhitektura oneM2M, kjer so vidni nivoji, entitete in referenčne točke.



Slika 2.1: oneM2M arhitektura po nivojih (pridobljeno iz [5]).

2.2.1.1 Entitete

Arhitektura je sestavljena iz naslednjih entitet:

- Application Entity - AE: *aplikacijska entiteta* je entiteta na aplikacijskem nivoju, ki implementira logiko aplikacijske storitve M2M. Vsa logika aplikacijske storitve je lahko nameščena na več vozliščih M2M (lahko tudi več kot enkrat). Vsaka instanca logike aplikacijske storitve je poimenovana kot aplikacijska entiteta in je enolično določena z identifikatorjem aplikacijske entitete (AE-ID). Primer aplikacijske entitete bi bila instanca aplikacije, ki oddaljeno nadzoruje sladkor v krvi, aplikacija, ki meri porabo moči, ali kontrolna aplikacija.

- Common Services Entity - CSE: *entiteta skupnih storitev* predstavlja en primer iz množice funkcij skupnih storitev v okolju M2M. Funkcije

storitev so na voljo ostalim entitetam preko referenčnih točk Mca in Mcc. Referenčna točka Mcn se uporablja za dostop do entitet omrežnih storitev. Vsaka entiteta skupnih storitev je enolično določena z identifikatorjem CSE-ID. Primeri funkcij storitev, ki jih ponuja CSE so: upravljanje s podatki, upravljanje naprav, upravljanje z naročninami na storitve M2M in lokacijske storitve. Podfunkcije, ki jih ponuja CSE imenujemo tudi *skupne funkcije storitev* oz. CSF (*Common Services Functions*), ki so opisane v dodatku A.9. Normativi, ki implementirajo funkcije storitev v CSE, so lahko obvezni ali opcijski.

- Network Services Entity - NSE: *entitete omrežnih storitev* ponujajo omrežne storitve entitetam skupnih storitev. Primer takih storitev so upravljanje z napravami, lokacijske storitve in proženje naprav. Omrežja zagotavljajo prenos podatkov med entitetami v sistemu oneM2M. Storitve prenosa podatkov niso vključene v NSE.

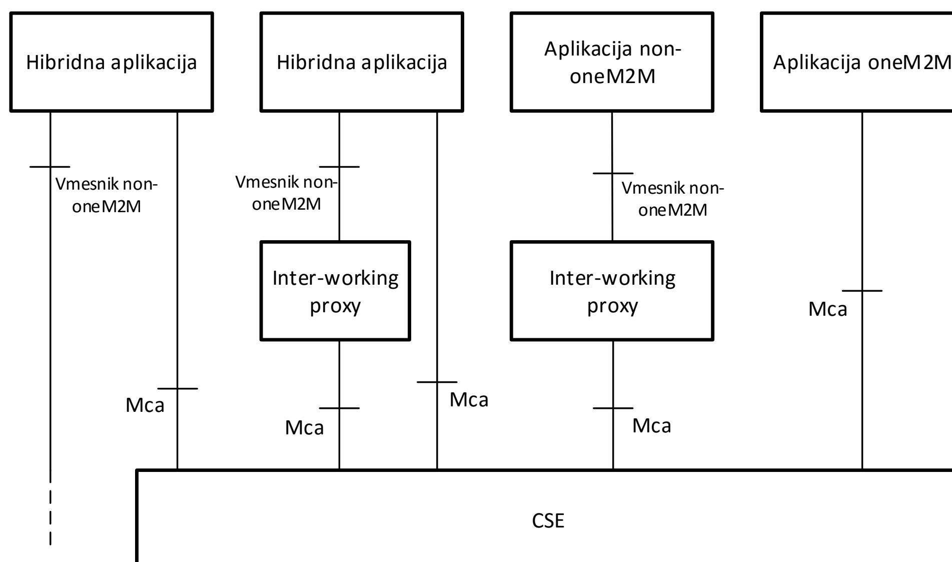
Ostali deli specifikacije so podrobno opisani v dodatku A. Opisana so vozlišča, referenčne točke, identifikatorji, formati identifikatorjev, viri, tipi virov in naslavljanje v sistemu oneM2M.

2.2.2 Integracija rešitev non-oneM2M

Rešitve non-oneM2M so prisotne in se bodo izdelovale tudi v prihodnosti. Niso zgrajene po specifikacijah oneM2M, zato obstaja način, kako te rešitve postanejo del sistema oneM2M. Za vse rešitve non-oneM2M mora oneM2M zagotoviti sredstva, da se omogoči:

- mešano postavitev (ang. *mixed deployment*, ki je deloma skladna z oneM2M, kjer sistem oneM2M zagotovi integracijo več tehnologij, npr. dodajanje novih tehnologij na obstoječe namestitve;

- hibridno postavitev, ki še vedno uporablja protokole non-oneM2M (lastniške ali standardne) in hoče dostop do funkcionalnosti oneM2M; tipičen



Slika 2.2: Podprti scenariji v sistemu oneM2M (povzeto po [17]).

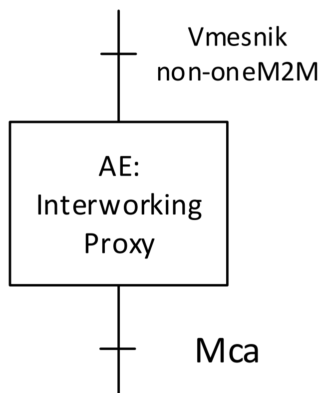
primer je izmenjava velike količine podatkov izven CSE (video nadzor) in uporaba storitev CSE (npr. upravljanje s kamerami).

Na sliki 2.2 so prikazani tipični scenariji, ki so podprti v arhitekturi oneM2M v kontekstu vzajemnega delovanja. Kombinacija različnih scenarijev dovoljuje mešane postavitve.

Vozlišče NoDN (glej A.2) nima informacij o sistemu oneM2M, zato se za povezovanje sistemov uporabi vozlišče ASN ali ADN. Imenujemo jih *Interworking Proxy Entities* (IPE).

2.2.2.1 Interworking Proxy Application Entity (IPE)

IPE so posebne, vzajemno delujoče aplikacijske entitete, ki so na CSE povezane preko referenčne točke Mca. Takim specializiranim aplikacijam pravimo *interworking proxy* (prikazana na sliki 2.3).



Slika 2.3: Aplikacijska entiteta Interworking Proxy, ki nastopa med referenčno točko Mca in vmesnikom, ki ni del sistema oneM2M (povzeto po [17]).

IPE je opredeljena s podporo referenčne točke non-oneM2M in z zmožnostjo preslikave zunanjega podatkovnega modela v vire oneM2M, ki so izpostavljeni preko referenčne točke Mca.

Prevedba podatkovnih modelov non-oneM2M v podatkovne modele oneM2M je podrobneje opisana v razdelku 6.2.3.

Naloge IPE:

- ustvari vire oneM2M, ki predstavljajo območno omrežje M2M (naprave, njihove aplikacije in vmesnike) na nivoju storitev, dostopnih preko referenčne točke Mca;

- upravljanje z viri oneM2M, če se struktura območnega omrežja M2M spremeni;

- samodejno odkrivanje sprememb v strukturi območnega omrežja M2M, če omrežje to podpira.

Specifikacije oneM2M so povzete iz [17].

2.3 OM2M

OM2M je odprtokodna platforma za komunikacijo M2M. Gre za projekt Eclipse, ki so ga začeli razvijati v francoskem Nacionalnem centru za znanstvene raziskave, v Laboratoriju za analizo in arhitekturo sistemov (LAAS-CNRS[1]). Je odprtokodna implementacija standardov oneM2M in smartM2M. Prva različica OM2M (verzije 0.8.0) je izšla aprila 2015 in je narejena po specifikacijah standarda ETSI smartM2M. Druga različica OM2M (verzije 1.0.0), ki je zgrajena po specifikacijah standarda oneM2M, je izšla konec junija 2016. Naša rešitev temelji na slednji.

OM2M je RESTful aplikacija in ponuja horizontalno storitveno platformo M2M za razvoj storitev, ki niso odvisne od omrežja. S tem naj bi olajšali nameščanje vertikalnih aplikacij in raznovrstnih naprav [9]. Arhitektura je modularna in teče na plasti OSGi, kar naredi platformo razširljivo z vtičniki. Podpira povezave med različnimi protokoli kot sta HTTP in CoAP. Na voljo so različni interoperabilni posredniki, ki omogočajo nemoteno komunikacijo med različnimi tehnologijami, kot so v našem primeru Bluetooth, BLE in Z-Wave.

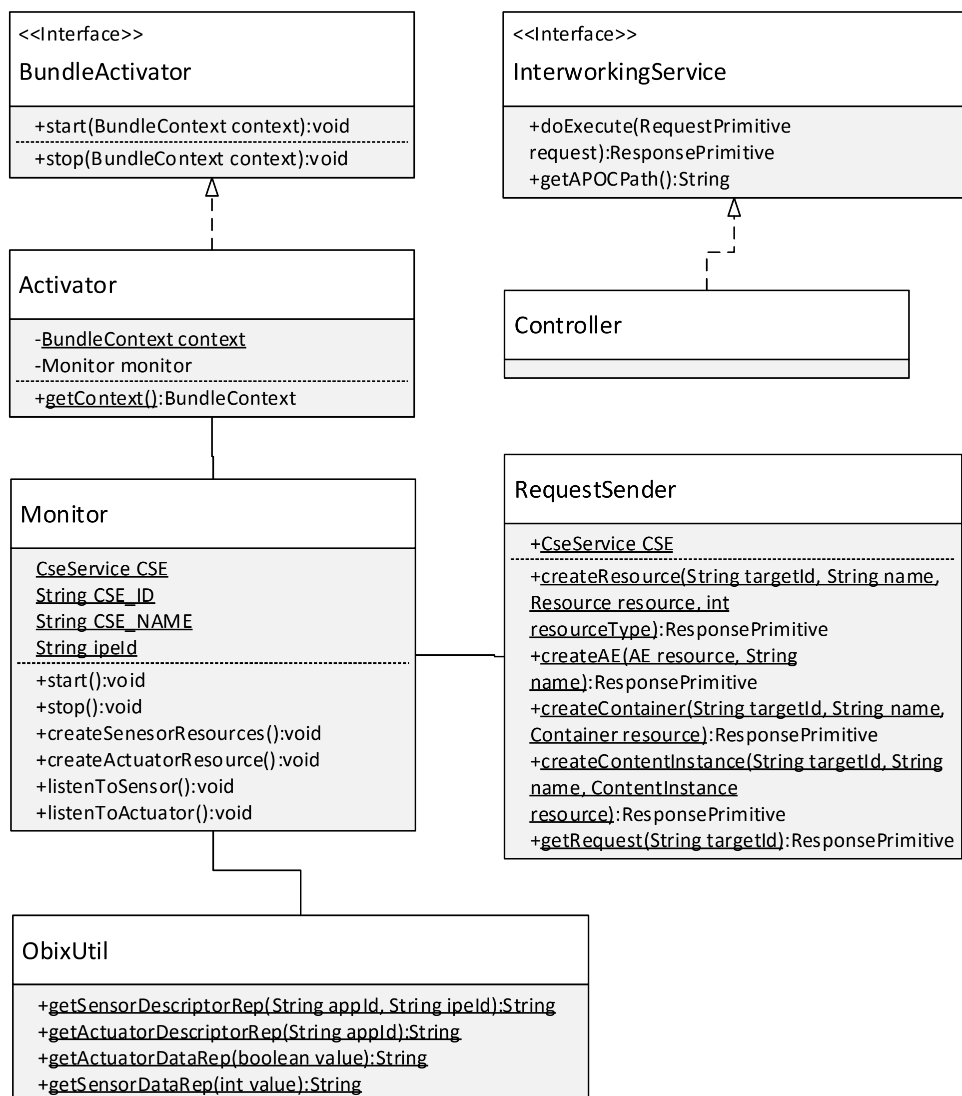
2.3.1 Razvoj vtičnikov

Za razvijanje vtičnikov za platformo OM2M imamo na voljo primer dobre prakse razvoja vtičnika, ki so ga razvili razvijalci platforme OM2M[2]. Razredna shema tega primera je prikazana na sliki 2.4.

Platforma OM2M nudi tri tipe vozlišč: IN, MN in ASN. Vtičnik je v tem primeru AE in ga zato dodamo v MN ali ASN.

[1] `https://www.laas.fr`

[2] Celotna izvorna koda se nahaja na spletni strani `http://wiki.eclipse.org/OM2M/one/Developer` (dostopano 9.8.2016)



Slika 2.4: Diagram UML primera dobre prakse razvijanja vtičnikov za platformo OM2M.

2.3.1.1 Razred Activator

Vsak vtičnik mora vsebovati razred `Activator`, ki se požene ob zagonu vtičnika. Razred `Activator` implementira vmesnik `BundleActivator`, ki je del OSGi (nahaja se v paketu org.osgi.framework). Vmesnik `BundleActivator` definira metodi *start()* in *stop()*, ki se pokličeta ob zagonu vtičnika oz. ob ustavitvi vtičnika. Metodi kot argument sprejmeta kontekst, ki ga predstavlja razred `BundleContext`. V metodi *start()* se ustvari nov objekt `ServiceTracker`, ki poenostavi uporabo storitev iz storitvenega registra ogrodja in omogoča spremljanje storitev. Ob kreiranju objekta podamo naslednje argumente:

- kontekst, ki smo ga dobili v metodo *start()*,
- ime razreda storitev, ki ga želimo spremljati,
- instanco vmesnika `ServiceTrackerCustomizer`, ki omogoča manipulacijo s storitvami, ki jih sledimo. Pokliče se ob dodajanju (metoda *addingService()*), urejanju (metoda *modifiedService()*) ali brisanju (metoda *removedService()*) storitev iz objekta tipa `ServiceTracker`.

Ob kreaciji objekta lahko spustimo zadnji argument (podamo *null*), ker že razred `ServiceTracker` implementira vmesnik `ServiceTrackerCustomizer`. Metode *addingService()*, *modifiedService()*, *removedService()* lahko implementiramo na način, kot je prikazano v odseku kode 2.1.

```
ServiceTracker<Object, Object> cseServiceTracker = new ServiceTracker<
    Object, Object>(bundleContext, CseService.class.getName(), null) {
  public void removedService(ServiceReference<Object> reference, Object
      service) {
    //implementacija metode removedService
  }
  public Object addingService(ServiceReference<Object> reference) {
    //implementacija metode addingService
    return cseService;
  }
  public void modifiedService(ServiceReference<S> reference, T service) {
    //implementacija metode modifiedService
  }
};
```

Izsek kode 2.1: Primer kreiranja objekta tipa ServiceTracker.

V metodi *addingService()* ustvarimo novo nit, v kateri ustvarimo instanco razreda Monitor. Na koncu metode *start()* v razredu Activator pokličemo metodo *open()* razreda ServiceTracker, ki začne spremljati storitve.

2.3.1.2 Razred Monitor

Monitor je razred, v katerem se ustvarijo viri. V konkretnem primeru se ustvarita dve aplikacijski entiteti: MY_SENSOR in MY_ACTUATOR. Vsaki AE se dodata dva vsebnika: DESCRIPTOR in DATA.

V vsebnikih DESCRIPTOR se ustvarita instanci vsebine, ki vsebujeta različne operacije. MY_SENSOR vsebuje metodi za pridobivanje zadnjega vnesenega podatka (*latest*) in za pridobivanje trenutnega stanja senzorja. MY_ACTUATOR vsebuje, poleg metod za pridobivanje vrednosti iz aktuatorja, še metodi za interakcijo z aktuatorjem, s katerima ga vklopimo in izklopimo.

Na koncu se zaženeta niti senzorja in aktuatorja, ki poskrbita za ustvarjanje podatkov, ki se vnašajo kot instance vsebine pod vsebnik DATA.

2.3.1.3 Razred RequestSender

Razred `RequestSender` se uporablja za pošiljanje zahtevkov na CSE. Zahtevek je predstavljen z razredom `RequestPrimitive`. Kreiranemu objektu nastavimo naslednje atribute:

- *From*: določimo pošiljatelja zahtevka. V tem primeru se sklicujemo na konstanto ADMIN_REQUESTING_ENTITY v razredu `Constants` (v paketu org.eclipse.om2m.commons.constants). Ta konstanta ima vrednost sistemske spremenljivke org.eclipse.om2m.adminRequestingEntity v kateri sta shranjena uporabniško ime in geslo.

- *TargetId* predstavlja pot do ciljnega vira, ki ga želimo ustvariti. Vrednost tega atributa, v primeru ustvarjanja instance vsebine za DESCRIPTOR aktuatorja, bi bila "/CSE_ID/CSE_NAME/actuatorId/-DESCRIPTOR".

- *ResourceType* je tipa `BigInteger`, zato so v razredu `ResourceType` (v paketu org.eclipse.om2m.commons.constants) določene konstante, da vemo, katera vrednost pomeni kateri tip vira. Za določanje tipa vira AE, nastavimo atribut na vrednost *ResourceType.AE*.

- *RequestContentType*: s tem atributom nastavimo, v kakšni obliki pošiljamo vsebino. Konstante lahko pridobimo iz `MimeMediaType`, kjer so vnaprej določene vrste vsebine. Lahko izbiramo med XML, JSON, objektom in oBIX.

- *ReturnContentType*: enako kot pri `RequestContentType`, le da v tem primeru določimo obliko, v kateri želimo prejeti odgovor na zahtevek.

- *Content*: ta atribut predstavlja vsebino zahtevka. Če smo `RequestContentType` nastavili na *MimeMediaType.OBJ*, lahko nastavimo atribut na objekte tipa `AE`, `Container` ali `ContentInstance`.

- *Name*: nastavimo ime vira.

- *Operation*: nastavimo operacijo zahtevka. Razred `Operation` določa naslednje izbire: `CREATE` za ustvarjanje vira, `RETRIEVE` za pridobivanje vira, `UPDATE` za urejanje vira, `DELETE` za brisanje vira in `NOTIFY` za obveščanje o viru (poleg je tudi `DISCOVERY` za odkrivanje virov, ki ni po specifikaciji ampak obstaja zaradi priročnosti).

2.3.1.4 Razred ObixUtil

`ObixUtil` je razred, ki nam pomaga pri pretvorbi objekta v format oBIX. Ustvarimo nov objekt tipa `Obj` (v paketu org.eclipse.om2m.commons.obix), ki predstavlja objekt oBIX. Temu objektu dodamo objekte različnih tipov (vse po specifikaciji oBIX [15]). V tem primeru dodamo operacijo, ki je predstavljena z razredom `Op` (v org.eclipse.om2m.commons.obix). Nastavimo ji naslednje atribute: ime operacije, URI (*Uniform Resource Identifier*) do operacije in instanco razreda `Contract`, ki predstavlja tip operacije (v tem primeru podamo niz *execute*).

2.3.1.5 Razred Controller

Razred `Controller` implementira vmesnik `InterworkingService` (paket org.-eclipse.om2m.interworking.service). Implementirani sta metodi *doExecute()* (ker so operacije, določene v `ObixUtil`, tipa *execute*) in *getAPOCPath()*. Metoda *doExecute()* se uporablja za prejemanje zahtevkov s strani CSE. Iz URI-ja zahtevka lahko pridobimo operacijo, ki smo jo nastavili prej v `ObixUtil` z metodo *setHref()*. Metoda *getAPOCPath()* mora vračati enako vrednost, kot jo vsebuje seznam dostopnih točk. Dostopno točko do aplikacije nastavimo ob kreiranju AE v razredu `Monitor`, kjer najprej pridobimo seznam z metodo *getPointOfAccess()* in nato jo z metodo *add()* dodamo.

2.3.1.6 Postopek izvajanja vtičnika

Na sliki 2.5 je prikazan splošen postopek izvajanja vtičnika, kateremu ustreza zgoraj opisani. Na sliki so prikazani tudi razredi, ki so potrebni za iskanje naprav in branje podatkov z njih.

Postopek izvajanja vtičnika po korakih:

1. Na začetku je treba zagnati vtičnik preko konzole OSGi (ki predstavlja vozlišče, na katerem je nameščen vtičnik). To storimo z ukazom `start <st>`, kjer *st* predstavlja številko vtičnika. Številko vtičnika dobimo tako, da poženemo ukaz `ss` v konzoli OSGi. Ta ukaz izpiše vse nameščene vtičnike.

2. Razred `Activator` prejme ukaz za zagon vtičnika.

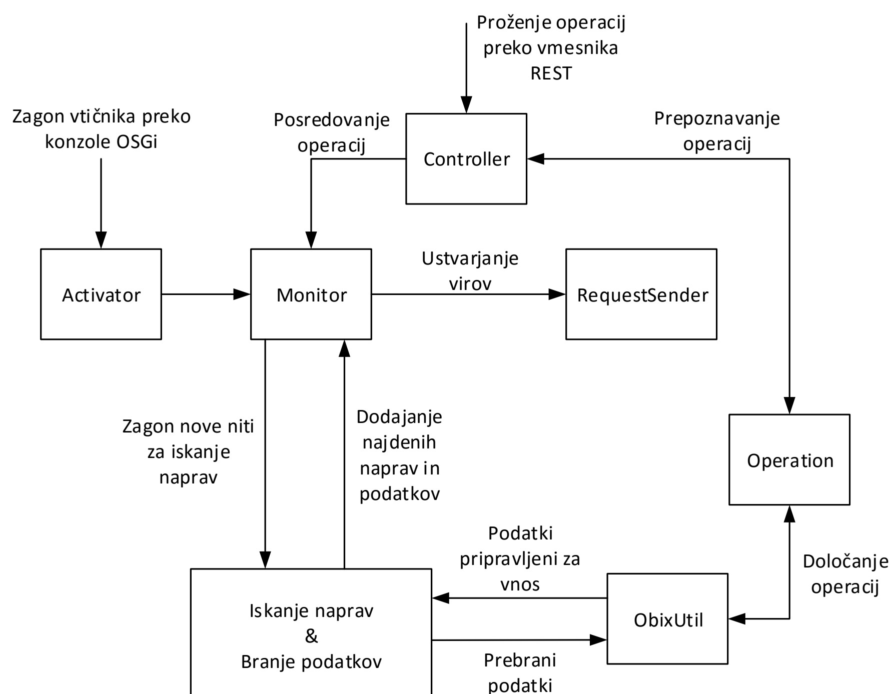
3. `Activator` zažene novo nit procesa, ki se začne v razredu `Monitor`.

4. Razred `Monitor` zažene novo nit procesa, ki skrbi za iskanje naprav in branje podatkov. Ustvarijo se viri, ki so potrebni za prihodnji vnos naprav.

5. Vsakič, ko je najdena nova naprava, se ustvarijo viri in vnesejo (opisni) podatki o napravi, ki jih razred `ObixUtil` preoblikuje v format oBIX. V opisu lahko nastopijo tudi operacije, ki so določene v razredu `Operation`. Vsi zahtevki se pošljejo v razredu `RequestSender`.

6. Vsakič, ko je nov podatek pripravljen za dodajanje v sistem, se preoblikuje v format oBIX (podatek kot par ključ-vrednost; sama vrednost ostane enaka kot jo preberemo).

7. Uporabnik oz. aplikacija, ki dostopa do podatkov preko storitve REST, lahko vnesene podatke prebere. Če so v opisnih podatkih navedene tudi operacije, jih lahko prožijo. Proženje operacije pošlje zahtevek, ki ga prestreže razred `Controller`. Ta na podlagi tipa operacije (ki so določene v razredu `Operation`) ustrezno izvrši določene ukaze (običajno se kličejo ustrezne metode v razredu `Monitor`).



Slika 2.5: Splošen postopek izvajanja vtičnika.

2.4 Bluetooth

Bluetooth je brezžična tehnologija povezovanja naprav in izmenjevanja podatkov med njimi na frekvencah od 2,4 GHz do 2,485 GHz. Specifikacijo so razvili pri Bluetooth Special Interest Group (Bluetooth SIG). IEEE (*Institute of Electrical and Electronics Engineers*) je Bluetooth standardiziral kot IEEE 802.15.1, ampak ga ne vzdržuje več [28]. Specifikacija Bluetooth določa protokole in aplikacijske profile, ampak ne definira programskih vmesnikov [13].

2.4.1 Profili Bluetooth

Bluetooth SIG je poleg protokolov določila tudi profile. Profil definira načine uporabe protokolov in njihovih lastnosti, ki so uporabni za posamezne skupine opravil. Naprava Bluetooth lahko podpira več profilov hkrati in s profili določa, kakšne podatke lahko pošilja. Aplikacija na napravi Bluetooth določa, katere profile naprava podpira. Dve napravi Bluetooth sta kompatibilni, če podpirata enaka profila.

Profili opisujejo enake primere uporabe naprave, ampak se za implementaciji BR/EDR (*Basic Rate/Enhanced Data Rate*, "klasični Bluetooth") in BLE (ang. *Bluetooth Low Energy*) upoorabljajo različni. Za doseganje kompatibilnosti med obema implementacijama se uporablja ti. krmilnik z dvojnim načinom delovanja (ang. *dual-mode controller*). Krmilnik mora biti prisoten na vsaj eni od obeh naprav.

Za BR/EDR obstaja velik nabor profilov, ki opisujejo različne tipe aplikacij oz. primere uporab naprav. Za LE (Low Energy) je razvijalcem na voljo veliko sprejetih profilov, lahko pa uporabijo GATT za ustvarjanje novih profilov. Z razširjanjem sprejetih profilov se vzdržuje interoperabilnost novih aplikacij z ostalimi napravami Bluetooth [23].

Primer navadnega profila Bluetooth je *Hands-Free Profile* (HFP), ki opredeljuje prostoročno uporabo naprave. Primer profila, ki je odvisen od profila GATT, je *Blood Pressure Profile* (BLP), ki določa, da je naprava zmožna merjenja krvnega tlaka in pošiljanja podatkov po specifikaciji BLP.

2.4.2 Bluetooth LE

Bluetooth LE oz. Bluetooth Low Energy (tudi Bluetooth Smart) je verzija Bluetooth, ki je bila razvita za IoT in se je začela kot del specifikacije Bluetooh 4.0 Core v letu 2010 [27]. Cilj razvijalcev je bil razviti standard z nizko porabo energije.

Vsi trenutni aplikacijski profili za BLE temeljijo na GATT (ang. *Generic Attribute Profile*), ki je splošna specifikacija za pošiljanje in prejemanje sporočil

preko BLE [21].

Bluetooth SIG je vnaprej definiral kopico profilov, ki temeljijo na GATT in zajemajo veliko različnih področij uporabe. Nekateri izmed takih profilov so:

- *Find Me Profile*: dovoli napravam, da locirajo druge naprave (npr. obesek, ki vsebuje oddajnik BLE, nam omogoči, da najdemo svoj mobilni telefon ali obratno),
- *Proximity Profile*: zaznava, ali je predmet v bližini ali ne,
- *HID over GATT Profile*: prenaša podatke HID preko BLE (npr. tipkovnice, miške, daljinski upravljalniki),
- *Glucose Profile*: prenaša podatke o nivoju glukoze preko BLE,
- *Health Thermometer Profile*: prenaša podatke o telesni temperaturi preko BLE,
- *Cycling Speed and Cadence Profile*: prenaša podatke o hitrosti in kadenci iz senzorjev na kolesu na pametni telefon oz. tablico.

Celoten seznam profilov, ki jih je odobril SIG, je na voljo v specifikaciji Bluetooth oz. na njihovi spletni strani [21].

Specifikacija Bluetooth dovoljuje definiranje novih profilov, ki pokrivajo druga področja uporabe kot standardni profili. Te definirajo proizvajalci, ki želijo zakriti način prenosa podatkov med njihovimi napravami in aplikacijami (npr. medicinska naprava in aplikacija za pametni telefon). Profile lahko tudi objavijo, da ostali prispevajo svoje implementacije profila, glede na specifikacijo proizvajalca [27].

2.4.2.1 Protokol *Attribute Protocol* - ATT

Protokol ATT je preprost odjemalec/strežnik protokol brez stanja, ki temelji na atributih naprave. V BLE je naprava lahko odjemalec, strežnik ali oboje. Odjemalec zahteva podatke od strežnika in ta mu pošlje podatke nazaj.

Protokol je strikten glede sekvence sporočil. Če zahtevek čaka (ni bilo odgovora z nasprotne strani), ni mogoče poslati novih zahtevkov, dokler se prejšnji ne obdela. To velja za obe smeri, če sta napravi odjemalec in strežnik hkrati.

Vsak strežnik ima podatke organizirane v atribute, kjer je vsak predstavljen s 16 bitnim oprimkom (ang. *attribute handle*), globalno unikatnim identifikatorjem (UUID), dovoljenji (ang. *permissions*) in vrednostjo (ang. *value*). Atributni oprimek je identifikator, ki se uporablja za dostop do vrednosti atributa. UUID določa tip in naravo vrednosti podatka.

Odjemalec lahko bere in piše vrednosti iz/v strežnik s pošiljanjem bralnih/pisalnih zahtevkov. Vsak zahtevek se naslovi na oprimek atributa, katerega se bere oz. piše. Strežnik odgovori z vrednostjo atributa oz. s potrditvijo prejetja zahtevka [27].

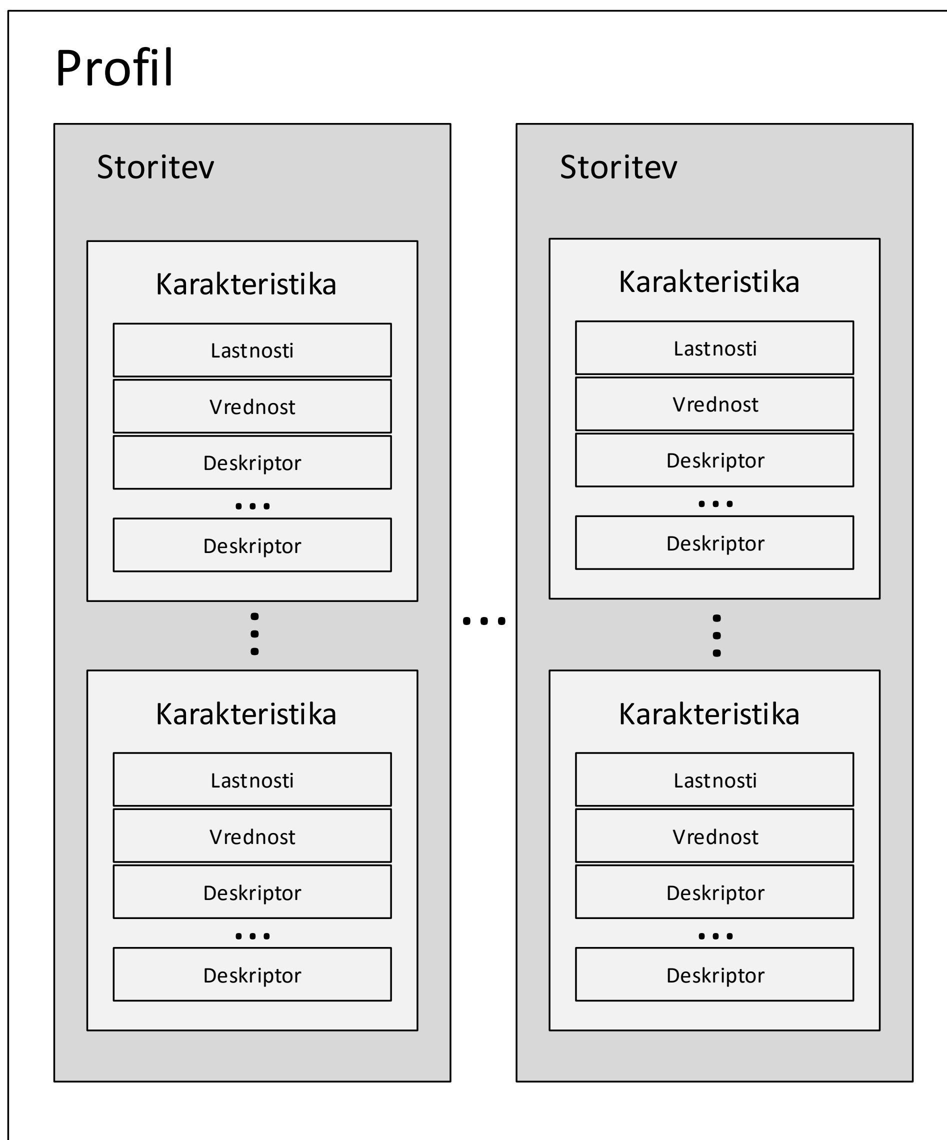
2.4.2.2 GATT

GATT (*The Generic Attributes*) definira, kako se prenašajo podakti profilov in uporabniški podatki preko povezave BLE. Definira postopke za izmenjavo podatkov in hierarhijo podatkovne strukture, ki je na voljo v napravah Bluetooth LE (prikazana na sliki 2.6). GATT uporablja protokol ATT za transportni protokol, ki definira potek izmenjave sporočil med dvema napravama Bluetooth LE.

Na vrhu hierarhije je profil Bluetooth, ki je sestavljen iz ene ali več storitev (ang. *service*), ki služijo podobnemu namenu (področju) uporabe. Vsi standardni profili BLE so zgrajeni na GATT in morajo biti z njim v skladu za pravilno delovanje. GATT je ključen za specifikacijo BLE, saj je vsak podatek na napravi BLE oblikovan, zapakiran in poslan po pravilih, ki jih določa GATT[27].

Storitev predstavlja skupino konceptualno podobnih delov podatkov, ki so predstavljeni s karakteristikami (ang. *characteristic*). Karakteristika je sestavljena iz tipa (predstavljenega z UUID), iz vrednosti, lastnosti, ki jih karakteristika podpira, in varnostnih dovoljenj. Vsebuje lahko tudi enega

ali več deskriptorjev (ang. *descriptor*), ki ponujajo metapodatke in/ali informacije o nastavljanju karakteristike, kateri pripadajo.



Slika 2.6: Hiearhija profilov, storitev, značilnosti, lastnosti, vrednosti, deskriptorjev v profilu GATT (povzeto po [22]).

Specifikacija GATT je podrobno opisana v dodatku B. Opisani so atributi, oprimki, tipi, dovoljenja, vrednosti, storitve, karakteristike in deskriptorji, ki jih opredeljuje GATT.

2.4.3 Bluetooth v Javi

Java APIs for Bluetooth Wireless Technology (JABWT) je specifikacija J2ME[3] za programske vmesnike, ki omogočajo izvajanje MIDletov[4] na vgrajenih napravah, kot so mobilne naprave, ki uporabljajo brezžično tehnologijo Bluetooth. JABWT je bil razvit kot JSR-82 s strani JCP (Java Community Process) [20]. JSR-82 implementacija obstaja tudi za Java 2 Platform Standard Edition (J2SE).

2.4.3.1 JSR-82

Java Specification Request 82 ali krajše JSR-82 definira način uporabe brezžične tehnologije Bluetooth za platformo Java 2, ki zagotavlja GCF (*Generic Connection Framework*), vključno s trenutnim profilom platforme Micro Edition (J2ME). Definira programski vmesnik API (*Application Programming Interface*), ki je v pomoč razvijalcem aplikacij Bluetooth v Javi [13]. Specifikacija Bluetooth se razširja z dodajanjem novih profilov, zato je specifikacija JSR-82 določena tako, da omogoča gradnjo novih profilov na podlagi tega programskega vmesnika. Novi profili so grajeni na protokolih OBEX (*Object Exchange Protocol*) in L2CAP (*Logical Link Control and Adaptation Protocol*), zato sta programska vmesnika za ta dva protokola tudi na voljo in omogočata prihodnje razvijanje profilov v Javi. Programski vmesnik, ki ga specificira JSR-82, omogoča:

- registriranje storitev,

[3]Java Platform, Micro Edition oz. Java 2 Platform, Micro Edition je javansko ogrodje za vgrajene sisteme (npr. mobilne naprave) [18].

[4]MIDlet je aplikacija, ki uporablja Mobile Information Device Profile (MIDP) in je del Connected Limited Device Configuration (CLDC) za okolje J2M [26].

- odkrivanje naprav in storitev,
- vzpostavljanje povezave preko RFCOMM, L2CAP ali OBEX,
- zagotavljanje varnosti.

Specifikacija JSR-82 naslavlja štiri izmed splošnejših profilov, kot so *Generic Access Profile* (GAP), *Serial Port Profile* (SPP), *Service Discovery Application Profile* (SDAP) in *Generic Object Exchange Profile* (GOEP).

Med profili obstajajo relacije. Na primer profil *File Transfer Profile* je narejen nad GOEP, ki bazira na SPP, ki je zgrajen na GAP.

Funkcionalnost programskega vmesnika, ki ga ponuja JSR-82, lahko razdelimo v tri kategorije:

- odkrivanje: naprav, storitev in registriranje storitev;
- komunikacija: vzpostavljanje povezav med napravami in uporaba teh povezav za komunikacijo med napravami;
- upravljanje: upravljanje in nadzorovanje povezav med napravami.

API definira 2 javanska paketa: javax.bluetooth in javax.obex. Oba paketa se nanašata na paket javax.microedition.io.

2.5 Z-Wave

Z-Wave je lastniška brezžična komunikacijska tehnologija, ki se osredotoča na področje hišne avtomatizacije. Z-Wave omogoča povezovanje različnih senzorje, aktuatorjev, stikal, ključavnic, luči in ostalih naprav v omrežje. V Evropi deluje na frekvenci 868,42 MHz in zagotavlja prenos podatkov do 100 kbps (kilobitov na sekundo) [29].

V omrežje Z-Wave lahko priključimo 232 vozlišč (ang. *nodes*), ki se delijo na krmilne naprave (ang. *controllers*) in podrejene naprave (ang. *slaves*). Omrežje je zankasto (ang. *mesh topology*), saj lahko vsako vozlišče deluje kot

obnavljalnik (ang. *repeater*) signala (vozlišče poleg lastnega prometa usmerja tudi promet ostalih vozlišč) za zagotavljanje povezljivosti po celi hiši.

Krmilne naprave lahko začnejo prenos podatkov in krmilijo promet po omrežju Z-Wave. Podrejene naprave pa so le končne naprave, ki imajo splošne vhodno-izhodne funkcionalnosti in izvršujejo ukaze krmilne naprave. Tudi pri obnavljanju signala, krmilna naprava pove podrejeni, ali naj sporočilo posreduje ali ne [10].

Krmilne naprave se ločijo glede na funkcionalnost v omrežju. Delimo jih na prenosne in statične naprave.

Prenosne krmilne naprave so lokacijsko neodvisne, saj imajo zmožnost ugotavljanja, kje se v prostoru nahajajo. To storijo s pinganjem ostalih naprav v bližini. Prenosne krmilne naprave ponavadi uporabljajo uporabniki omrežja, da lahko pošiljajo ukaze v omrežje. Druga funkcija prenosnih krmilnih naprav je vključevanje oz. izključevanje naprav (vozlišč) v oz. iz omrežja. Vsako omrežje Z-Wave ima primarno krilno napravo, ki vodi vključevanje/izključevanje in poseduje zadnjo konfiguracijo omrežja. Ostale krmilne naprave kopirajo to konfiguracijo [10]. Med prenosne krmilne naprave spada tudi naprava, uporabljena za potrebe tega magistrskega dela.

Poglavje 3

Knjižnice, orodja in naprave

Za razvoj rešitve smo uporabili odprtokodne knjižnice in orodja. Za razvoj vmesnika, ki deluje v klasičnem omrežju Bluetooth, smo uporabili knjižnico *Bluecove*, za Z-Wave knjižnico *zwave4j* in za BLE orodji *hcitool* in *gatttool*. Za testiranje rešitve smo uporabili naprave, ki so opisane v 3.4.

3.1 Bluecove

Bluecove je javanska knjižnica za Bluetooth (implementacija JSR-82), ki deluje z naslednjimi skladi Bluetooth :

- Microsoft Winsock,
- BlueSoleil (IVT Corporation),
- Broadcom WIDCOMM in
- Mac OS X.

Bluecove vsebuje dodaten modul BlueCove-GPL, ki je licenciran z GPL [1] in podpira izvajanje tudi na operacijskih sistemih Linux (podpora za Linuxov sklad Bluetooth *bluez*) [2].

[1]GNU General Public License, `https://www.gnu.org/licenses/gpl-3.0.html`

Bluecove vsebuje emulator JSR-82, ki nadomesti fizične naprave Bluetooth in nam tako omogoči lažje testiranje aplikacij. Več o sami implementaciji in uporabi knjižnice je opisano v razdelku 4.2.2.

3.2 Zwave4J

Knjižnica zwave4j[2] je ovojnica (ang. wrapper) za knjižnico OpenZWave[3] (OZW). Knjižnica nam omogoča razvoj aplikacij v programskem jeziku Java za nadzorovanje krmilnih enot, ki jih priključimo v računalnik, in s tem posledično nadzorovanje omrežja Z-Wave. Omogoča programsko vključevanje vozlišč v omrežje in nadzorovanje le-teh.

Knjižnica vsebuje mapo z nastavitvenimi datotekami. V mapi se nahajajo sheme XML in datoteke XML za raznorazne naprave Z-Wave, ki vsebujejo – glede na proizvajalca – specifične podatke o napravah. V mapi se nahaja glavna nastavitvena datoteka *options.xml*

Komunikacija med krmilno enoto, ki je priključena v računalnik, in ostalimi napravami v omrežju Z-Wave poteka asinhrono. Nekatere naprave v omrežju spijo, da varčujejo z energijo, in lahko prejmejo ukaze le, ko so zbujene. Zato v aplikaciji ne smemo pričakovati takojšnjega odziva naprave na ukaz. Zaradi tega razloga metode v OZW ne vrnejo vrednosti takoj. Po poslanem zahtevku v omrežje se odgovor vrne v aplikacijo preko sistema obvestil (povratnih klicev). Upravljanje z obvestili je tako ključno v vsaki aplikaciji, ki uporablja OZW.

Omrežje Z-Wave je dinamično. Krmilne enote in naprave se lahko dodajajo ali odvzemajo poljubno ob poljubnem času. Sistem obvestil se uporablja za informiranje aplikacije o vseh spremembah v strukturi omrežja.

Več o sami knjižnici in njeni uporabi v razdelku 4.1.3.

[2]`https://github.com/zgmnkv/zwave4j` (pridobljeno 6.6.2016).

[3]`https://github.com/OpenZWave/open-zwave`, `http://www.openzwave.net` (pridobljeno 6.6.2016).

3.3 Orodja in knjižnice za BLE

V programskem jeziku Java ne obstaja knjižnica za Bluetooth Low Energy, zato smo se poslužili orodij *gatttool* in *hcitool*, ki pa delujeta le v operacijskih sistemih Linux, kjer sta vključena v Bluetooth sklad *bluez*. Za izdelavo rešitve v operacijskih sistemih Windows, bi se morali poslužiti drugega programskega jezika.

3.3.1 Orodje *hcitool*

Hcitool je orodje, ki se uporablja za zaznavanje naprav Bluetooth. Z orodjem lahko prikažemo lokalne naprave Bluetooth, poiščemo naprave Bluetooth v dosegu, prikažemo različne informacije o napravi idr.

Glavni funkciji orodja, ki smo ju uporabili pri razvoju vmesnika za BLE, sta:

- prikaz lokalnih naprav Bluetooth z ukazom `hcitool dev` (izpis 3.1),

```
1 Devices:
2   hci1  00:1A:7D:DA:71:14
3   hci0  00:50:56:E6:7F:94
```

Izpis 3.1: Lokalni vmesniki HCI in pripadajoči naslovi MAC.

- iskanje naprav BLE v okolici z ukazom `hcitool lescan` (izpis 3.2).

```
LE Scan ...
08:7C:BE:8B:9E:BB Quintic BLE
08:7C:BE:8B:9E:BB (unknown)
08:7C:BE:8B:9E:BB Quintic BLE
08:7C:BE:8B:9E:BB (unknown)
08:7C:BE:8B:9E:BB Quintic BLE
08:7C:BE:8B:9E:BB (unknown)
08:7C:BE:8B:9E:BB Quintic BLE
08:7C:BE:8B:9E:BB Quintic BLE
08:7C:BE:8B:9E:BB (unknown)
08:7C:BE:8B:9E:BB Quintic BLE
08:7C:BE:8B:9E:BB (unknown)
08:7C:BE:8B:9E:BB Quintic BLE
08:7C:BE:8B:9E:BB (unknown)
```

Izpis 3.2: Zaznane naprave BLE v dosegu in pripadajoče ime naprave.

Ukaz `hcitool lescan` se izvaja v neskončnost, zato ga prekinemo s signalom SIGINT.

Več o orodju in njegovi uporabi je prikazano v razdelku 4.1.2.

3.3.2 Orodje *gatttool*

Gatttool je orodje za interakcijo z napravami Bluetooth Low Energy. Omogoča odkrivanje primarnih storitev, ki ji naprava ponuja. Omogoča tudi odkrivanje, pisanje in branje karakteristik preko povezovalnih točk (ang. *handle*).

Format ukaza je `gatttool <stikala>`, kjer so (za nas) najpomembnejša naslednja stikala:

- `--device=<naslov_MAC>`: naslov MAC naprave, ki ga pridobimo z orodjem *hcitool*;
- `-i <ime_lokalne_naprave>`: s tem stikalom določimo, katero lokalno napravo naj orodje uporabi za komunikacijo (uporaba stikala ni nujna);
- `--primary`: prikaz primarnih storitev, ki jih ponuja naprava (primer ukaza je prikazan v izpisu 3.3, primer rezultata pa v izpisu 3.4);

```
gatttool --device=00:11:22:33:44:55 --primary
```

Izpis 3.3: Primer ukaza s stikalom `--primary`.

```
attr handle = 0x0001, end grp handle = 0x000b uuid:
    00001800-0000-1000-8000-00805f9b34fb
attr handle = 0x000c, end grp handle = 0x000f uuid:
    00001801-0000-1000-8000-00805f9b34fb
attr handle = 0x0010, end grp handle = 0x0016 uuid: 0000fee8
    -0000-1000-8000-00805f9b34fb
attr handle = 0x0017, end grp handle = 0x001d uuid: 0000fee9
    -0000-1000-8000-00805f9b34fb
```

Izpis 3.4: Za vsako storitev na napravi se izpišejo začetni oprimek atributa (*attr handle*), končni oprimek atributa (*end grp handle*) in UUID storitve.

- `--characteristics`: prikaz vseh karakteristik na napravi (primer ukaza je prikazan v izpisu 3.5, rezultat tega ukaza pa v izpisu 3.6);

```
gatttool --device=00:11:22:33:44:55 --characteristics
```

Izpis 3.5: Primer ukaza s stikalom `--characteristics`.

```
1  handle = 0x0002, char properties = 0x0a, char value handle = 0x0003,
       uuid = 00002a00-0000-1000-8000-00805f9b34fb
2  handle = 0x0004, char properties = 0x02, char value handle = 0x0005,
       uuid = 00002a01-0000-1000-8000-00805f9b34fb
3  handle = 0x0006, char properties = 0x0a, char value handle = 0x0007,
       uuid = 00002a02-0000-1000-8000-00805f9b34fb
4  handle = 0x0008, char properties = 0x02, char value handle = 0x0009,
       uuid = 00002a04-0000-1000-8000-00805f9b34fb
5  handle = 0x000a, char properties = 0x0e, char value handle = 0x000b,
       uuid = 00002a03-0000-1000-8000-00805f9b34fb
6  handle = 0x000d, char properties = 0x22, char value handle = 0x000e,
       uuid = 00002a05-0000-1000-8000-00805f9b34fb
7  handle = 0x0011, char properties = 0x10, char value handle = 0x0012,
       uuid = 003784cf-f7e3-55b4-6c4c-9fd140100a16
8  handle = 0x0015, char properties = 0x04, char value handle = 0x0016,
       uuid = 013784cf-f7e3-55b4-6c4c-9fd140100a16
9  handle = 0x0018, char properties = 0x0c, char value handle = 0x0019,
       uuid = d44bc439-abfd-45a2-b575-925416129600
10 handle = 0x001b, char properties = 0x10, char value handle = 0x001c,
       uuid = d44bc439-abfd-45a2-b575-925416129601
```

Izpis 3.6: Za vsako karakteristiko na napravi se izpišejo deklaracije karakteristike, ki so opisane z oprimkom (*handle*), lastnostmi karakteristike (*char properties*), oprimkom vrednosti karakteristike (*char value handle*) in UUID karakteristike.

- `--char-desc`: prikaz vseh atributov na napravi (primer ukaza je prikazan v izpisu 3.7, rezultat pa v izpisu 3.8).

```
1 gatttool --device=00:11:22:33:44:55 --char-desc
```

Izpis 3.7: Primer ukaza s stikalom `--char-desc`.

```
handle = 0x0001, uuid = 00002800-0000-1000-8000-00805f9b34fb
handle = 0x0002, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0003, uuid = 00002a00-0000-1000-8000-00805f9b34fb
handle = 0x0004, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0005, uuid = 00002a01-0000-1000-8000-00805f9b34fb
handle = 0x0006, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0007, uuid = 00002a02-0000-1000-8000-00805f9b34fb
handle = 0x0008, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0009, uuid = 00002a04-0000-1000-8000-00805f9b34fb
handle = 0x000a, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x000b, uuid = 00002a03-0000-1000-8000-00805f9b34fb
handle = 0x000c, uuid = 00002800-0000-1000-8000-00805f9b34fb
handle = 0x000d, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x000e, uuid = 00002a05-0000-1000-8000-00805f9b34fb
handle = 0x000f, uuid = 00002902-0000-1000-8000-00805f9b34fb
handle = 0x0010, uuid = 00002800-0000-1000-8000-00805f9b34fb
handle = 0x0011, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0012, uuid = 003784cf-f7e3-55b4-6c4c-9fd140100a16
handle = 0x0013, uuid = 00002902-0000-1000-8000-00805f9b34fb
handle = 0x0014, uuid = 00002901-0000-1000-8000-00805f9b34fb
handle = 0x0015, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0016, uuid = 013784cf-f7e3-55b4-6c4c-9fd140100a16
handle = 0x0017, uuid = 00002800-0000-1000-8000-00805f9b34fb
handle = 0x0018, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0019, uuid = d44bc439-abfd-45a2-b575-925416129600
handle = 0x001a, uuid = 00002901-0000-1000-8000-00805f9b34fb
handle = 0x001b, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x001c, uuid = d44bc439-abfd-45a2-b575-925416129601
handle = 0x001d, uuid = 00002902-0000-1000-8000-00805f9b34fb
```

Izpis 3.8: Primer izpisa s stikalom `--char-desc`, kjer se za vsak atribut izpišeta oprimek atributa (*handle*) in UUID atributa.

Vsi prikazani primeri zapisov so dejanski podatki, pridobljeni iz uporabljene zapestnice BLE.

Več o orodju *gatttool* in njegovi uporabi je prikazano v razdelku 4.1.2.

3.4 Naprave

Za potrebe tega magistrskega dela smo izbrali naslednje naprave:

- Z-Wave pametna žarnica *domitech Z-Wave Smart LED Light Bulb*,

skupaj s krmilno napravo Z-Wave *AEOTEC Z-Stick GEN5*;

- pametna zapestnica BLE *Smart Bracelet TW64*;
- mobilni telefon z nameščenim operacijskim sistemom Android.

Za povezovanje in iskanje naprav BLE potrebujemo vmesnik Bluetooth, ki podpira verzijo Bluetooth 4.0 ali višje.

Za povezovanje z odjemalcem (aplikacijo oz. vtičnikom) smo uporabili aplikacijo *Bluetooth SPP Server Terminal*[4].

[4]Aplikacija je prosto dostopna v trgovini Play na spletnem naslovu `https://play.google.com/store/apps/details?id=mobi.minipedia.btserverandroid`

Poglavje 4

Razširitev OM2M

V okviru magistrskega dela smo razvili vtičnike za brezžične tehnologije Bluetooth, Bluetooth Low Energy in Z-Wave. Začeli smo z razvojem vmesnikov za vsako tehnologijo posebej. Potem smo jih vključili v platformo OM2M, ki je razširljiva z vtičniki.

Izziv pri implementaciji je bil razviti vmesnike za vsako tehnologijo posebej, ki znajo:

a) zaznati naprave v prostoru,

b) se z njimi povezati in

c) z njih brati podatke.

4.1 Implementacija vmesnikov

V prvem koraku smo razvili samostojno javansko aplikacijo, ki zna v prostoru zaznati naprave Bluetooth, BLE in Z-Wave, se z njimi povezati in od njih pridobivati podatke. Za vsako izmed tehnologij smo poiskali odprtokodne implementacije knjižnic.

Iskali smo javanske implementacije knjižnic, saj je platforma OM2M napisana v Javi. Na ta način bi poskrbeli za lažjo integracijo med našo rešitvijo in platformo OM2M.

4.1.1 Vmesnik za Bluetooth

Za implementacijo vmesnika za Bluetooth smo uporabili knjižnico Bluecove. Implementacija Bluetooth se začne z iskanjem naprav v okolici. Za vsako zaznano napravo poiščemo storitve, ki jih naprava ponuja. Na storitve lahko nato odpiramo povezave, preko katerih pridobivamo podatke iz naprave.

4.1.1.1 Iskanje naprav in storitev

Implementacija se začne z javanskim vmesnikom (ang. *interface*) `DiscoveryListener`. Z implementiranjem tega razreda moramo napisati (povoziti) naslednje metode:

- *deviceDiscovered()*: proces pokliče to metodo, ko je zaznana nova naprava Bluetooth. Metoda sprejme dva parametra: napravo, ki je predstavljena z razredom `RemoteDevice`, in razred naprave, ki je predstavljen z javanskim razredom `DeviceClass`.

- *servicesDiscovered()*: proces pokliče to metodo, ko je na napravi Bluetooth zaznana nova storitev (ang. *service*). Metoda sprejme celoštevilski identifikator transakcije iskanja storitev in polje odkritih storitev na napravi, ki so predstavljene z razredom `ServiceRecord`.

- *inquiryCompleted()*: proces pokliče to metodo, ko je zaznavanje naprav končano. Sprejme celoštevilski argument, na podlagi katerega ugotovimo, kako se je iskanje naprav končalo. Možne so tri vrednosti (te so statično določene v razredu `DiscoveryListener`, zato jih naslavljamo po imenu in ne direktno po številki):

  – `INQUIRY_COMPLETED` oz. 0x00: iskanje naprav ni bilo prekinjeno.

  – `INQUIRY_TERMINATED` oz. 0x05: iskanje naprav je bilo prekinjeno.

  – `INQUIRY_ERROR` oz. 0x07: med iskanjem je prišlo do napake. Iskanje se ni končalo brez prekinitev.

- *serviceSearchCompleted()*: ta metoda se pokliče, ko je zaznavanje storitev na napravah končano ali prekinjeno. Tudi ta metoda sprejme celoštevilski transakcijski identifikator in celo število, ki nam pove, kako se je iskanje storitev na napravi končalo. Možne so naslednje vrednosti:
  - `SERVICE_SEARCH_COMPLETED`: iskanje storitev se je brez prekinitev.
  - `SERVICE_SEARCH_TERMINATED`: iskanje storitev je bilo prekinjeno (s klicem metode *cancelServiceSearch()* v razredu `DiscoveryAgent`).
  - `SERVICE_SEARCH_ERROR`: med iskanjem storitev je prišlo do napake.
  - `SERVICE_SEARCH_NO_RECORDS`: ni bilo najdenih storitev.
  - `SERVICE_SEARCH_DEVICE_NOT_REACHABLE`: ni bilo mogoče vzpostaviti povezave med lokalno napravo in napravo, na kateri je potekalo iskanje storitev, oz. je ta povezava bila prekinjena.

Naslednji korak je deklariranje lokalne naprave Bluetooth. Lokalna naprava je predstavljena z razredom `LocalDevice` v paketu javax.bluetooth in omogoča dostop do sklada Bluetooth in dostop do lokalne naprave Bluetooth v splošnem. V razredu `LocalDevice` nam je na voljo metoda *getLocalDevice*, s katero pridobimo instanco tega razreda. Ko imamo instanco razreda `LocalDevice`, lahko pridobimo instanco do razreda `DiscoveryAgent`. Ta nam omogoča iskanje naprav in storitev. V kontekstu iskanja naprav ponuja metodi:

- *startInquiry()*: začne postopek iskanja naprav v dosegu. Metodi podamo argumenta:
  - *AccessCode*: celo število, ki pomeni tip poizvedbe. To je lahko `GIAC` (*General/Unlimited Inquiry Access Code*) oz. 0x9E8B33, `LIAC` (*Limited Dedicated Inquiry Access Code*) oz. 0x9E8B00 ali število med 0x9E8B00 in 0x9E8B3F.
  - *Listener*: podamo instanco našega razreda, ki implementira vmesnik `DiscoveryListener`. S tem omogočimo klice metod, ki smo jih

implementirali (*deviceDiscovered()*, *servicesDiscovered()*, *inquiryCompleted()* in *serviceSearchCompleted()*).

- *cancelInquiry()*: metoda se uporablja za prekinitev iskanja naprav. Kot argument sprejme instanco našega razreda, ki implementira vmesnik `DiscoveryListener`.

Razred `DiscoveryAgent` ponuja poleg metod za iskanje naprav tudi metodi za iskanje storitev:

- *searchServices()*: uporablja se za iskanje vseh storitev na napravi Bluetooth, ki so določene v podanem argumentu *uuidSet*. Polje *uuidSet* je tipa `UUID[]` (razred se nahaja v paketu javax.bluetooth) in v njem so naštete storitve, predstavljene z UUID (enolični identifikator storitve). Ko je iskanje končano in so storitve najdene, se pridobijo privzeti atributi[1] (storitve) in tisti, ki so določeni v polju podanem kot argument *attrSet*. Polje je tipa `int[]`. Metoda sprejme še naslednja argumenta:
  - *btDev*: naprava na kateri iščemo storitve,
  - *discListener*: instanca razreda, ki implementira razred `DiscoveryListener` (da se lahko kličeta metodi *servicesDiscovered()* in *serviceSearchCompleted()*).
- *cancelServiceSearch()*: s klicem te metode prekličemo iskanje storitev na napravi. Kot argument ji podamo celoštevilski identifikator transakcije, ki ustreza iskanju, ki ga želimo preklicati.

Metodi, ki sta del razreda `DiscoveryAgent` in ju še nismo omenili, sta:

- *retrieveDevices()*: metoda vrne seznam naprav Bluetooth (predstavljene z razredom `RemoteDevice`), ki so bile najdene v prejšnjih iskanjih ali so določene kot vnaprej znane. Izbiro vnesemo kot celoštevilski argument. Za naprave, najdene v prejšnjih iskanjih, uporabimo atribut

[1]Privzeti atributi so: *ServiceRecordHandle* (0x0000), *ServiceClassIDList* (0x0001), *ServiceRecordState* (0x0002), *ServiceID* (0x0003) in *ProtocolDescriptorList* (0x0004)

CACHED (v razredu DiscoveryAgent), ki ima vrednost 0x00, za vnaprej znane naprave pa atribut PREKNOWN (ravno tako iz razreda DiscoveryAgent), ki ima vrednost 0x01.

- *selectService()*: metoda se uporablja za izbiro storitve, ki vsebuje podan argument UUID. Metoda vrne povezovalni niz, ki se lahko kasneje uporabi za odpiranje povezave v metodi open razreda Connector.

4.1.1.2 Povezovanje z napravo in prenos podatkov

Povezovanje z napravo in prenos podatkov z naprave sledi, ko je iskanje naprav končano in smo zaznali vse storitve, ki jih naprave ponujajo.

Za povezovanje z napravo in branje podatkov z nje smo ustvarili razred ReadData. V tem razredu je metoda *readSerialData()*, ki za argument sprejme URL. V prejetem URL je določena shema, ki opredeljuje tip povezave, naslov naprave (skupaj z vrati), na katero se povezujemo, in še trije opcijski parametri. Ti parametri so *authenticate*, ki definira, ali je za vzpostavitev povezave potrebno biti avtenticiran, *encryption*, ki definira, ali mora biti povezava šifrirana, in *master*, ki definira, ali mora biti naprava nadrejena ali je lahko tudi podrejena. URL pridobimo iz objekta tipa ServiceRecord z metodo *getConnectionUrl()*.

Za odpiranje povezave uporabimo metodo *open()* iz razreda Connector, ki se nahaja v paketu javax.microedition.io. Metoda vrača objekt tipa Connection, ki je abstraktni tip, zato je v primeru, ko je shema povezave enaka *btspp* (profil serijske povezave oz. RFCOMM), potrebno dobljeno povezavo pretvoriti (ang. *cast*) v povezavo tipa StreamConnection. Objekt tipa StreamConnection ima metodo *openInputStream()*, s katero pridobimo vhodni tok InputStream. Slednjega podamo razredu BufferedReader, ki nam omogoča branje podatkov po vrsticah z metodo *readLine()*. Pri implementaciji odpiranja povezav na storitve smo si pomagali z dokumentom [8].

Ob odprtju povezave na strežnik je potrebno odjemalec in strežnik spariti. Na operacijskih sistemih Windows se prikaže obvestilo s številko, katero vnesemo v mobilni telefon oz. drugo napravo Bluetooth. Na operacijskem

sistemu Linux tega ni, je pa na strežniku (mobilnem telefonu) potrebno dovoliti, da se računalnik upari z napravo Bluetooth.

Za testiranje vmesnika smo na mobilnem telefonu zagnali aplikacijo *Bluetooth SPP Server Terminal*, ki na njem izpostavi storitev SPP, ki vzpostavi serijsko povezavo RFCOMM. Z aplikacijo odpremo povezavo te storitve in tako lahko s telefona prejemamo poslane podatke.

4.1.2 Vmesnik za BLE

Javanska knjižnica, ki bi podprla delovanje BLE, ne obstaja. Za implementacijo vmesnika za Bluetooth LE smo zato uporabili orodji *hcitool* in *gatttool*. Pregledali smo knjižnice za BLE v drugih programskih jezikih, vendar niso omogočale takšnega dostopa (tako podrobnega) do podatkov naprav, kot jih izbrani orodji. Zaradi izbire teh orodij omejimo izvajanje aplikacije na operacijske sisteme Linux. Za razvoj vmesnika, ki bi deloval na operacijskih sistemih Windows, bi morali uporabiti knjižnice drugih programskih jezikov ali druga orodja, ki se izvajajo v Windowsu.

4.1.2.1 Iskanje naprav

Implementacija se začne z iskanjem naprav BLE v okolici. Orodje *hcitool* omogoča veliko funkcij, ampak nam najbolj uporaben ukaz je `hcitool lescan` (glej 3.3.1), ki poišče vse naprave v okolici.

Najprej smo definirali metodo *process()*. V njej smo uporabili javanska razreda `Runtime` in `Process`, ki omogočata izvršbo ukaza v lupini.

Za iskanje naprav BLE uporabimo ukaz *hcitool lescan*, ki ga podamo metodi *process()*. Težava pri branju standardnega izhoda tega ukaza je, da orodje *hcitool* izpisuje zaznane naprave, vendar med izpisovanjem ne prazni medpomnilnika. Zato v Javi ne moremo prebrati vrednosti takoj, ampak moramo počakati, da *hcitool* izprazni medpomnilnik. Druga možnost je prekinitev izvajanja ukaza, saj se ob prekinitvi izprazni medpomnilnik in lahko preberemo izpis programa. Odločili smo se za slednjo pot in uporabili orodje *timeout*.

Orodje *timeout* omogoča izvajanje ukaza za določen čas, po pretečenem času pa podanemu ukazu (procesu) pošlje določen signal. Ukazu *timeout* je potrebno določiti, kakšen signal pošlje ukazu *hcitool lescan*, ker privzeti signal `SIGTERM` povzroči blokiranje lokalne naprave Bluetooth (*hcitool* ob naslednjem iskanju vrača napako *Set scan parameters failed: Input/output error*[2]). Ukazu *timeout* zato podamo signal `SIGINT`, ki pravilno zaustavi iskanje naprav in ne povzroča napak pri naslednjih iskanjih. Končni ukaz *timeout* je:

```
1 timeout --signal SIGINT 5s hcitool lescan
```

Čas, po katerem se pošlje signal, poljubno nastavimo (v tem primeru je nastavljen na 5 sekund). Z metodo *extractAddresses()* preberemo rezultat ukaza (za format izpisa glej 3.3.1) in izluščimo naslove MAC in pripadajoča imena naprav. Vrstica v izpisu se lahko za posamezno napravo ponovi večkrat, zato je v seznam naprav (predstavljen z razredom `HashMap`, kjer se za ključ uporabi naslov MAC in za vrednost ime naprave) potrebno eno napravo zapisati le enkrat. V izpisu se za en naslov MAC lahko pojavi dejansko ime naprave ali pa se izpiše *unknown* (prikazano v primeru izpisa v 3.3.1). V primeru, da metoda *extractAddresses()* za napravo najde pravo ime, tega tudi shrani v seznam, v nasprotnem primeru se shrani "unknown".

V rešitvi smo si zamislili dva seznama naprav. Prvi seznam *oldDevices* vsebuje vse zaznane naprave, drugi, *newDevices*, pa le na novo zaznane (zaznane v novem ciklu iskanja). Če rezultat iskanja vrne naprave, ki so že na seznamu vseh naprav *oldDevices*, je ne dodamo na seznam *newDevices* in tako preprečimo ponovno branje storitev in ostalih opisnih podatkov naprave. Za vse naprave, ki so na seznamu *newDevices*, opravimo iskanje storitev, karakteristik in deskriptorjev.

Iskanje storitev, karakteristik in deskriptorjev je napisano v metodi *searchServices()* (v razredu `ReadGatt`). Pri iskanju storitev gremo čez seznam *newDevices* in za vsako napravo izvedemo iskanje storitev. To storimo z orodjem

[2]Najlažja rešitev tega problema je odklop in ponovni priklop lokalne naprave Bluetooth.

gatttool. Ukaz `gatttool -i <hci_vmesnik> -b <naslov_MAC> --primary` nam izpiše vse storitve, ki jih ponuja naprava BLE (za format izpisa glej 3.3.2). Iz odgovora pridobimo začetni in končni oprimek storitve ter UUID storitve z regularnimi izrazi.

Za vsako storitev pridobimo karakteristike, ki ji pripadajo. To storimo z orodjem *gatttool*, pri čemer uporabimo stikalo `--characteristics` v kombinaciji s stikaloma `-s` in `-e`, ki označujeta začetni in končni oprimek (dobimo ju iz prejšnjega izpisa storitev). Na ta način dobimo le tiste karakteristike, ki spadajo pod isto storitev. Iz tega izpisa (za format in primer izpisa glej 3.3.2) dobimo oprimek, lastnosti, oprimek vrednosti in UUID karakteristike.

Za vsako karakteristiko je treba pridobiti še deskriptorje (če jih ima). Najprej pridobimo izhod programa *gatttool* s stikalom `–char-desc`, ki nam izpiše vse atribute na napravi (vključno s storitvami, karakteristikami, vrednostmi karakteristik in deskriptorji). Deskriptorje dobimo tako, da izločimo vse, kar ni deskriptor. To storimo na podlagi identifikatorjev UUID in, kot je navedeno v podrazdelku 2.4.2.2, imajo primarne storitve UUID 0x2800, sekundarne 0x2801, vključene storitve 0x2802 in karakteristike 0x2803. Ostanejo še vrednosti karakteristik, ki so vedno prvi atribut po deklaraciji karakteristike. Ukazu dodamo stikalo `-s`, ki določa, od kje naprej naj nam izpiše atribut. Prejšnje karakteristike nas ne zanimajo, zato stikalu podamo vrednost UUID oz. oprimek deklaracije karakteristike, katere deskriptorje iščemo. Prvi atribut izpisa je tako deklaracija karakteristike, drugi atribut, takoj za deklaracijo, je vrednost karakteristike, nato nastopijo desktiptorji, pod pogojem, da je UUID različen od zgoraj naštetih. Če pridemo do vrednosti UUID, ki ustreza storitvam ali karakteristikam, deksriptorjev za to karakteristiko ni več.

4.1.2.2 Branje podatkov z naprave

Drugi korak pri implementaciji vmesnika za BLE je branje podatkov iz naprave. Tu si ravno tako pomagamo z orodjem *gatttool*, ki omogoča izpis podatkov o storitvah, karakteristikah in deskriptorjih. Ko so vsi podatki z naprave znani, pričnemo z branjem podatkov.

Branje podatkov poteka v ločeni niti od niti iskanja naprav in poteka periodično v metodi *refreshingValues()* v razredu `ReadGatt`.

Dejanske vrednosti oz. podatki so na napravi shranjeni v vrednostih karakteristike. Ob iskanju storitev in karakteristik se v seznam *charValues* beležijo oprimki, na katerih se nahajajo vrednosti karakteristik. Zraven se zabeležijo še naslov MAC naprave, UUID storitve in UUID karakteristike, da se točno ve, kam spada vrednost. Vrednost karakteristike pridobimo tako, da z orodjem *gatttool* uporabimo stikalo `--char-read`. Dodati moramo še stikalo `-a`, če želimo prebrati vrednost na podlagi oprimka, oz. `-u`, če želimo prebrati vrednost na podlagi identifikatorja UUID.

4.1.3 Vmesnik za Z-Wave

Vmesnik za Z-Wave smo razvili s pomočjo knjižnice *zwave4j*. Implementacija vmesnika za Z-Wave se prične z definiranjem poti do mape nastavitvenih datotek, ki jih potrebuje knjižnica *zwave4j* (glej 3.2). Nastavitve se preberejo v objekt `Options` (iz paketa org.zwave4j) z metodo *create()*. Objektu `Options` lahko programsko dodajamo nastavitve tipa *boolean*, *integer* in *string*, da nam ni treba začasnih nastavitev pisati v datoteko XML.

Naslednji korak je instanciranje glavnega razreda v *zwave4j* - razreda `Manager`. Instanco razreda `Manager` pridobimo s klicem metode *create()*, ki se nahaja v razredu `Manager`. Vse funkcionalnosti potekajo preko tega objekta.

Glavna metoda *startListening()* se začne z definicijo objekta, ki skrbi za obvestila. Za to uporabimo vmesnik `NotificationWatcher`. Naredimo anonimno instanco tega vmesnika in v njem povozimo metodo *onNotification()*. Metoda sprejme dva argumenta: obvestilo, predstavljeno z razredom `Notification`, in kontekst, ki je predstavljen z razredom `Object`. Iz obvestila lahko z metodo *getType()* dobimo tip obvestila, na podlagi katerega ugotovimo za kakšno vrsto obvestila gre. S stavkom *switch* za vse tipe iz razreda `NotificaionType` opredelimo, kaj se zgodi, če se metoda pokliče s tem tipom obvestila. Nekaj glavnih tipov obvestil:

- DRIVER_READY: gonilnik krmilne enote je pripravljen za uporabo. Obvestilo vsebuje identifikator *HomeId*, ki se uporablja pri večini klicev metod v razred Manager.
- ALL_NODES_QUERIED: vsa vozlišča so bila poizvedena, zato lahko aplikacija pričakuje popolne podatke.
- ALL_NODES_QUERIED_SOME_DEAD: vsa vozlišča so bila poizvedena, ampak nekaj jih je bilo neodzivnih.
- NODE_ADDED: novo vozlišče je bilo dodano na seznam naprav v omrežju Z-Wave (dodana so tudi ob inicializaciji aplikacije).
- NODE_REMOVED: vozlišče je bilo odstranjeno iz seznama naprav v omrežju (odstranjena so tudi ob zapiranju aplikacije).
- NODE_PROTOCOL_INFO: obvestilo vsebuje splošne informacije o vozlišču.
- VALUE_ADDED: vrednost iz vozlišča je bila dodana. Obvestilo se pojavi ob dodajanju vozlišča v omrežje. Za vsak tip podatka (za vsak CommandClass) se pojavi vsaj eno obvestilo.
- VALUE_CHANGED: vrednost vozlišča se je posodobila in je drugačna od prejšnje vrednosti.
- VALUE_REFRESHED: vrednost vozlišča se je posodobila.

Instanco vmesnika NotificationWatcher dodamo instanci Manager z metodo *addWatcher()*. Metoda poleg instance vmesnika NotificationWatcher sprejme še kontekst, v katerem določimo dodatne lastnosti, ki se pošljejo k vsakemu obvestilu. V našem primeru te funkcije nismo uporabili in smo metodi za drugi argument podali kar *null*.

Določiti je treba, na katerih serijskih vratih je priklopljena naša krmilna enota. Uporabili smo metodo *addDriver()* razreda Manager, ki sprejme niz znakov s potjo do serijskih vrat. Primer za operacijske sisteme Windows je //./COM7, za operacijske sisteme Linux pa /dev/tty7.

Nato čakamo na obvestilo DRIVER_READY, ki nam pove, da je krmilna enota pripravljena. Ko je pripravljena, lahko dodajamo naprave v omrežje Z-Wave, jih odstranjujemo ali pustimo, da opravljajo delo. V obvestilih se pojavi vse, kar se zgodi v omrežju.

Ob zapiranju aplikacije odstranimo gonilnik krmilne enote z metodo *removeDriver()*. S klicem metode *destroy()* uničimo instanco razreda Manager.

Poleg glavne metode sta implementirani tudi metodi za dodajanje (*addingDevices()*) in odstranjevanje (*removingDevices()*) vozlišč v/iz omrežja. V obeh primerih ustvarimo instanco lokalnega razreda CmdCallback, ki implementira vmesnik ControllerCallback. Povoziti je potrebno metodo *onCallback()*, ki sprejme tri argumente: stanje krmilne enote ControllerState in napake krmilne enote ControllerError ter kontekst. Na podlagi vrednosti iz razreda ControllerState lahko ugotovimo, v kakšnem stanju je krmilna enota. Stanja krmilne enote so: NORMAL, STARTING, CANCEL, ERROR, WAITING, SLEEPING, IN_PROGRESS, COMPLETED, FAILED, NODE_OK, NODE_FAILED, NOT_SUPPORTED.

Za dodajanje vozlišč v omrežje uporabimo metodo *beginControllerCommand()* na objektu tipa Manager, ki sprejme tri argumente: *homeId*, konstanto ADD_DEVICE iz razreda ControllerCommand in instanco prej ustvarjenega razreda CmdCallback.

Za odstranjevanje naprav iz omrežja je postopek enak, le da namesto konstante za dodajanje naprav metodi *beginControllerCommand()* podamo konstanto REMOVE_DEVICE.

Za prekinitev dodajanja oz. odstranjevanja vozlišč uporabimo metodo *cancelControllerCommand()* razreda Manager, kateri podamo *homeId*.

Implementirali smo še metodi *switchAllOn()* in *switchAllOff()*, ki prižgeta oz. ugasneta vse naprave v omrežju Z-Wave. V vsaki od metod se pokliče istoimenska metoda, le da jih pokličemo na instanci razreda Manager in jim podamo *homeId*.

4.2 Integracija s platformo OM2M

Narejene vmesnike smo integrirali s platformo OM2M. To smo storili preko vtičnikov, saj je platforma OM2M zgrajena na razširljivi platformi OSGi, ki podpira razširitve preko vtičnikov.

Razvili smo po en vtičnik za vsako brezžično tehnologijo. Poimenovanje je v skladu z ostalimi vtičniki:

- Bluetooth: org.eclipse.om2m.binding.bt,
- Bluetooth LE: org.eclipse.om2m.binding.ble in
- Z-Wave: org.eclipse.om2m.binding.zwave.

Z razvojem vtičnikov za OM2M moramo slediti specifikaciji oneM2M, na kateri je zgrajen OM2M. Z vsakim vtičnikom ustvarimo posebno aplikacijsko entiteto, imenovano IPE (glej 2.2.2.1).

4.2.1 Integracija vmesnikov in vtičnikov

Pri integraciji vmesnikov v platformo OM2M smo sledili primeru dobre prakse, ki je opisana v razdelku 2.3.1. Vtičnik vsake tehnologije je narejen na enak način kot zgoraj opisani primer. Vsi vsebujejo razrede `Activator`, `Monitor`, `Controller`, `RequestSender`, `Constants`, `ObixUtil`, `Operations` in ostale razrede, potrebne za delovanje vmesnika.

Activator

Razred je enak v vseh treh vtičnikih, le da se v vsakem pokliče ustrezna metoda *start()* v ustreznem razredu `Monitor`. Implementiran je na enak način kot v primeru dobre prakse.

Constants

V tem razredu so shranjene konstantne vrednosti, kot so npr. ime AE, imena vsebnikov, ime dostopne točke. Vsak vtičnik ima svoj razred `Constants`, ki

vsebuje ustrezne vrednosti glede na tehnologijo, ki jo vtičnik predstavlja.

RequestSender

Razred je enak kot v primeru dobre prakse, saj lahko z njim ustvarjamo vse vire, ki jih potrebujemo v vseh vtičnikih.

Ostali razredi so različni za vsak vtičnik posebej, zato jih bomo podrobneje opisali. Na koncu vsakega razdelka vtičnika je shema, ki poenostavi razumevanje delovanja posameznega vtičnika.

4.2.2 Vtičnik za Bluetooth

Vtičnik za Bluetooth vsebuje še naslednje razrede:

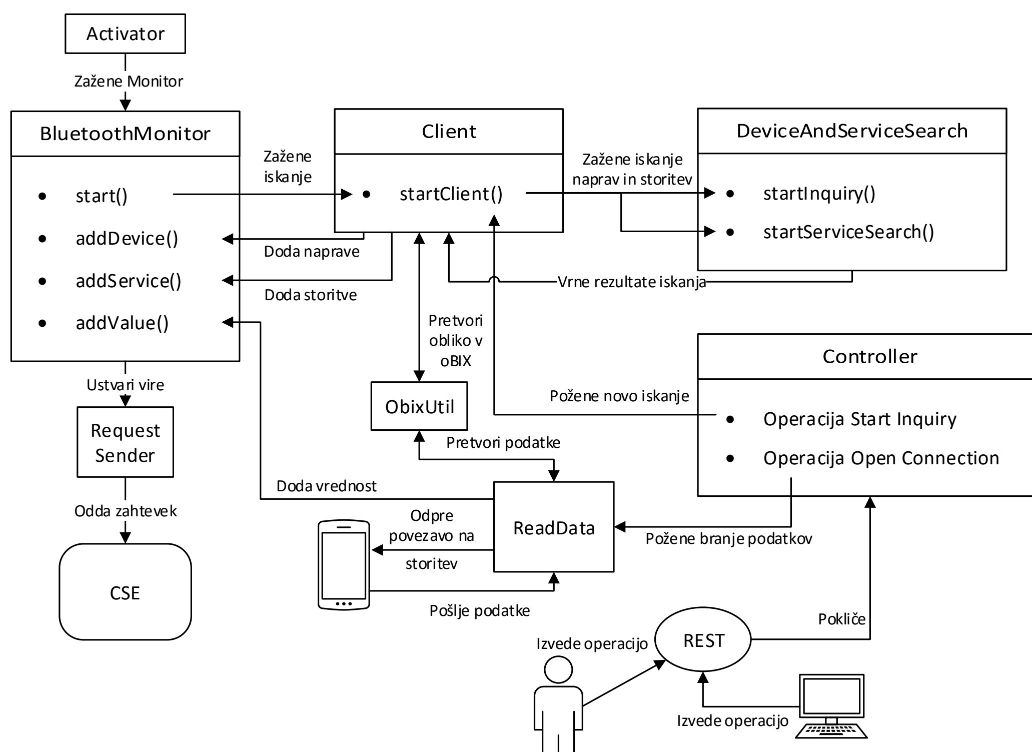
- `Client`: glavni razred, kjer se začne iskanje naprav in storitev,
- `DeviceAndServiceSearch`: razred, ki implementira vmesnik DiscoveryListener,
- `ReadData`: razred, ki se uporablja za odpiranje povezav na napravo in branje podatkov z naprave.

Za lažjo predstavo je postopek izvajanja vtičnika za Bluetooth prikazan na sliki 4.1.

BluetoothMonitor

Razred vsebuje metodo *start()*, v kateri se ustvarijo potrebni viri. To so AE, vsebnik za opis tehnologije in glavne operacije ter vsebnik za naprave. V vsebnik, ki vsebuje glavne operacije, smo dodali operacijo *Start Inquiry*, ki ob prejetem zahtevku ponovno sproži iskanje naprav.

Razred vsebuje še metodo za dodajanje naprav *addDevice()* in metodo za dodajanje storitev *addService()*. Na koncu izvajanja tega razreda zaženemo novo nit, v kateri poženemo iskanje naprav Bluetooth.



Slika 4.1: Postopek izvajanja vtičnika za Bluetooth.

Client in DeviceAndServiceSearch

`Client` je glavni razred za iskanje naprav in iskanje storitev na napravah. Vsebuje metodo *startClient()*, v kateri se ustvari nov objekt razreda `DeviceAndServiceSearch`. `DeviceAndServiceSearch` implementira vmesnik `DiscoveryListener` (implementacija je opisana v 4.1.1.1). Iz razreda `DeviceAndServiceSearch` dobimo seznam naprav, seznam storitev, ki jih ponujajo naprave, in seznam razredov naprav. Naprave so predstavljene z razredom `RemoteDevice`, iz katerega lahko pridobimo naslov MAC (metoda *getBluetoothAddress()*) in ime naprave (z metodo *getFriendlyName()*).

V vsaki iteraciji zanke, ko gremo čez seznam naprav, dodamo napravo v sistem OM2M. Za to uporabimo metodo *addDevice()* v razredu `BluetoothMonitor`. Metoda sprejme dva argumenta; naslov MAC naprave in reprezentacijo oBIX opisa naprave. Naslov MAC naprave smo uporabili za ime vsebnika. V opis smo zapisali ime naprave, glavni razred in podrazred naprave ter število najdenih storitev na napravi. Ob dodajanju naprave v OM2M se obenem ustvarijo novi vsebniki v vsebniku, ki predstavlja napravo. To so vsebniki za predstavitev naprave, za podatke z naprave in za storitve. Vsebnik za storitve se ustvari v vsebniku za opis, saj storitve opisujejo napravo.

V vsaki iteraciji zanke čez naprave pridobimo storitve, shranjene v seznamu objektov tipa `ServiceRecord`. Za dodajanje storitev v OM2M uporabimo metodo *addService()* v razredu `BluetoothMonitor`. Metoda sprejme dva argumenta: naslov MAC naprave (da vemo kam uvrstiti storitve) in predstavitev storitve v formatu oBIX. V predstavitev storitve vključimo ime storitve, oprimek zapisa in povezavo do storitve. Dodamo tudi operacijo *openConnection*, ki odpre povezavo do storitve.

Controller

Razred implementira vmesnik `InterworkingService`, zato sta v njem metodi *doExecute()* in *getAPOCPath()*. V metodi *doExecute()* pričakujemo dve operaciji.

Prva je ponovno iskanje naprav. Najprej preverimo, če iskanje naprav

že poteka. To storimo tako, da preverimo spremenljivko *inquiryActive* tipa *boolean*, ki smo jo nastavili na *true*, ko smo v razredu `BluetoothMonitor` zagnali novo nit. Ko se je nit zaključila, smo spremenljivko nastavili na false. V primeru, da je iskanje zaključeno, ga ponovno zaženemo, kot smo ga v `BluetoothMonitor`. V primeru, da je v teku, samo vrnemo odgovor.

Druga operacija je odpiranje povezave do storitve na napravi. S strani OM2M dobimo poleg vrste operacije tudi povezavo, ki jo odpremo v razredu `ReadData` in preberemo podatke, kot je opisano v 4.1.1.2. Vsako prebrano vrstico iz naprave vnesemo v OM2M kar iz razreda *ReadData*. Za predstavitev podatka uporabimo objekt `Str`, ki ponazarja znakovno predstavitev podatka. Vsak prejet podatek je nova instanca vsebine v vsebniku s podatki naprave.

4.2.3 Vtičnik za BLE

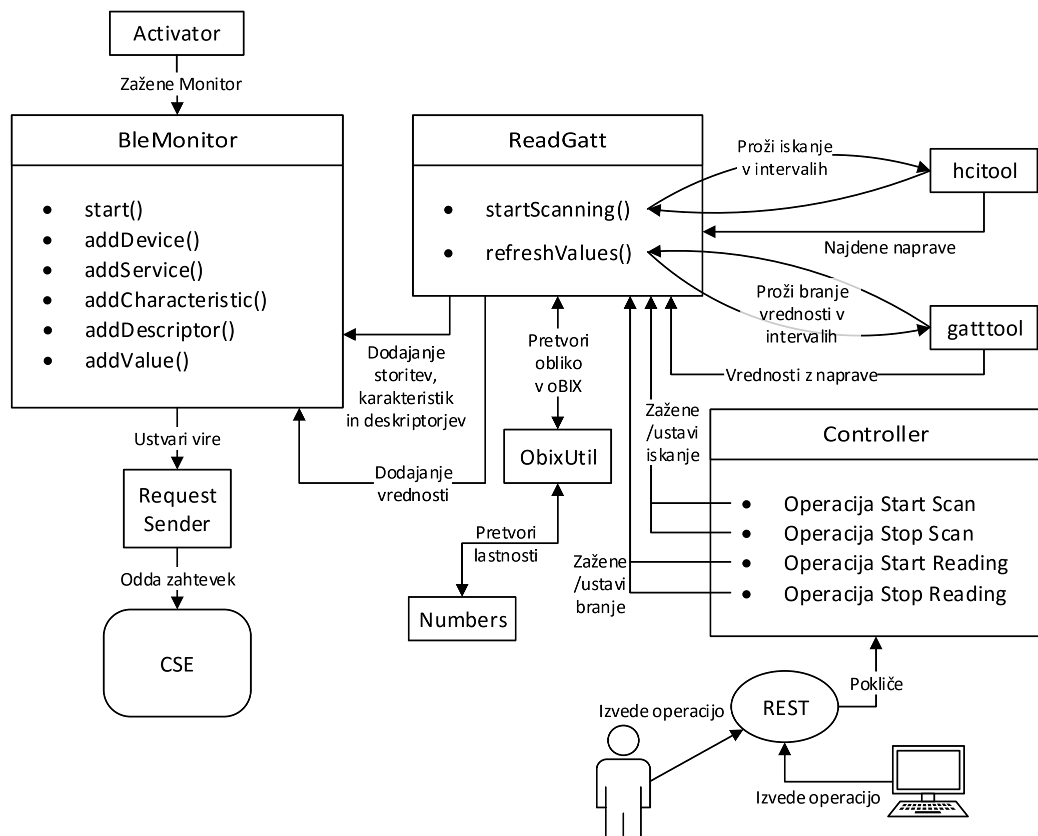
Vtičnik za BLE poleg standardnih razredov vsebuje še razreda `ReadGatt` in `Numbers`.

Za lažjo predstavo je postopek izvajanja vtičnika za BLE prikazan na sliki 4.2.

BleMonitor

Razred poleg metod za kreiranje virov vsebuje še metode za dodajanje naprav *addDevice()*, za dodajanje storitev *addService()*, za dodajanje karakteristik *addCharacteristic()*, za dodajanje deskriptorjev *addDescriptor()*, za dodajanje vrednosti *addValue()* in za popravljanje naslovov MAC *correctAddress()*. Slednja se uporablja za popravljanje naslovov, saj ti vsebujejo dvopičja, ki niso dovoljena za imena virov. Zato dvopičja zamenjamo z minusi.

Na začetku se ustvarijo viri: AE, v AE se ustvarita vsebnik za opis in vsebnik za naprave. Po ustvaritvi potrebnih virov se zažene nova nit z iskanjem naprav.



Slika 4.2: Postopek izvajanja vtičnika za BLE.

ReadGatt

Je glavni razred za iskanje naprav in storitev na napravah BLE. Implementacija razreda je opisana v podrazdelku 4.1.2.1. Vse novo zaznane naprave, ki so v seznamu *newDevices*, dodamo v OM2M z metodo *addDevice()* v `BleMonitor`. Metoda *addDevice()* prejme dva argumenta: naslov in ime naprave (točno to, kar je shranjeno v seznamu *newDevices*). Naprava se vnese v vsebnik vseh naprav z imenom, ki je enako naslovu naprave. Tu pride do prej omenjenega problema, ker v imenu virov ne smejo nastopati dvopičja. Naslov oz. ime vsebnika ustrezno popravimo z metodo *correctAddress()*. V vsebnik naprave se dodata še vsebnik z opisom naprave in vsebnik s storitvami. V opis naprave dodamo naslov naprave in ime naprave.

Po vnosu naprave v OM2M, jo dodamo na seznam *oldDevices*. Po vnosu vseh naprav v OM2M, pričnemo z iskanjem storitev na napravah. Iskanje storitev je podrobno opisano v podrazdelku 4.1.2.1. Vsako najdeno storitev vnesemo v OM2M z metodo *addService()* v `BleMonitor`. Metoda sprejme tri argumente: naslov naprave, UUID primarne storitve in opis naprave v formatu oBIX , v katerem je shranjen primarni UUID, oprimek začetka storitve in oprimek konca storitve. V opis storitve dodamo tudi ime storitve, ki ga poiščemo s pomočjo knjižnice *SmartGattLib*, ki ima zmožnost preslikave UUID v ime storitve. Za to uporabimo razred `Service` (v paketu com.movisens.smartgattlib), kjer nam metoda *lookup()* vrne ime storitve, če le to obstaja.

Vsaka storitev ima eno ali več karakteristik, ki jih vnesemo v OM2M metodo *addCharacteristic()* v razredu `BleMonitor`. Metoda sprejme štiri argumente: naslov naprave, UUID primarne storitve, UUID karakteristike in opis naprave oBIX, v katerem je oprimek deklaracije karakteristike, oprimek vrednosti karakteristike, lastnosti karakteristike in UUID karakteristike. V opis naprave dodamo tudi ime karakteristike, ki ga poiščemo na podlagi UUID, z metodo *lookup()* iz razreda `Characteristic` iz knjižnice SmartGattLib. Po vnosu karakteristike dodamo vsebnik za podatke in za deskriptorje.

Vse vrednosti karakteristike so shranjene v seznamu *charValues*, kot je to

opisano v 4.1.2.2. Ob osveževanju vrednosti primerjamo vrednost iz seznama in dejansko vrednost z naprave. Če se je vrednost spremenila, jo vnesemo v OM2M z metodo *addValue()* v razredu `BleMonitor` in staro vrednost v *charValues* nadomestimo z novo. Ob iskanju karakteristik vnesemo v seznam prazen niz, da se ob prvem branju podatkov z naprave, podatki vnesejo v OM2M. V OM2M vnesemo vrednost, ki nam jo izpiše orodje *gatttool*. Vrednosti so vedno v šestnajstiški obliki, razen če karakteristika nima vrednosti. V slednjem primeru vnesemo *null*. Metoda *addValue()* prejme štiri argumente: naslov naprave, UUID storitve, UUID karakteristike in vrednost, predstavljeno v formatu oBIX. Vrednost se vnese v vsebnik s podatki posamezne karakteristike.

Vsaka karakteristika ima nič ali več deskriptorjev. Način, kako dobiti deskriptorje, je opisan v podrazdelku 4.1.2.1. Za dodajanje posameznih deskriptorjev uporabimo metodo *addDescriptor()*, ki prejme naslov naprave, UUID primarne storitve, UUID karakteristike in opis deskriptorja, predstavljenega v formatu oBIX. V opisu deskriptorja so shranjeni UUID, vrednost in ime deskriptorja, ki ga poiščemo v razredu `Descriptor` v knjižnici SmartGattLib.

Controller

Razred je narejen na enak način, kot tisti pri Bluetooth, le da podpira druge operacije. Podpira operaciji za začetek in zaustavitev iskanja ter operaciji za začetek in zaustavitev osveževanja vrednosti. Iskanje naprav se izvaja v zanki, dokler je spremenljivka *scanning* enaka *true*. Enako velja za osveževanje vrednosti, ki se izvaja, dokler je spremenljivka *refreshing* enaka *true*. V primeru, da želimo začeti oz. zaustaviti iskanje naprav, je potrebno preveriti vrednost spremenljivke *threadAlive*, ki nam pove, če iskanje že poteka. Če tega ne bi storili, bi sprožili dve niti za iskanje naprav, kar bi lahko privedlo do nekonsistentnega obnašanja aplikacije. Enako velja za začenjanje oz. zaustavljanje osveževanja vrednosti, le da je v tem primeru potrebno preveriti vrednost spremenljivke *readingThreadAlive*. Te štiri operacije se lahko prožijo iz vsebnika, ki vsebuje opis BLE.

Numbers

Razred se uporablja za pretvorbo šestnajstiške vrednosti lastnosti karakteristike v besedno obliko. Iz poizvedovanja po karakteristikah dobimo lastnosti karakteristike zapisane v obliki `0xNN` (glej 2.4.2.2). Šestnajstiško vrednost NN najprej pretvorimo v binarno. Na podlagi binarnega zapisa vemo, kateri bit je postavljen. V seznam *properties* smo zapisali besedne vrednosti lastnosti v obratnem vrstnem redu, da nam ni treba obračati binarnega zapisa. Nato gremo čez binarni zapis in kjer je bit nastavljen, pogledamo v seznam za besedno obliko lastnosti. Vse skupaj združimo v en niz lastnosti, ki ga vnesemo v opis karakteristike.

4.2.4 Vtičnik za Z-Wave

Vtičnik za Z-Wave vsebuje le en dodaten razred `ZWaveMain`.

Za lažjo predstavo je postopek izvajanja vtičnika za Z-Wave prikazan na sliki 4.3.

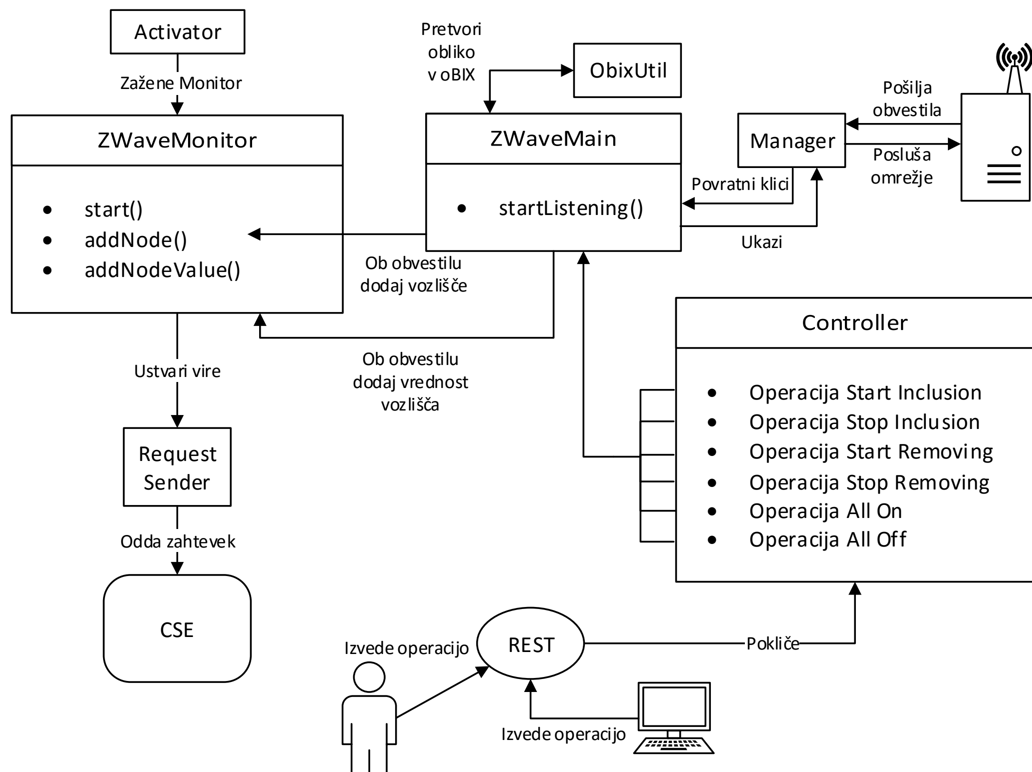
ZWaveMonitor

V tem razredu se ustvari AE, v kateri kreiramo vsebnika za opis tehnologije in za naprave v omrežju. Poleg metod za ustvarjanje virov sta v razredu še metodi za dodajanje vozlišča *addNode()* in vrednosti *addNodeValue()*. Po ustvarjenih virih se v novi niti zažene metoda *startListening()*, ki upravlja z omrežjem Z-Wave.

ZWaveMain

Prej omenjena metoda *startListening()* se nahaja v razredu `ZWaveMain`, ki je edini razred, ki upravlja z omrežjem Z-Wave. Implementacija razreda je opisana v razdelku 4.1.3. Glede na povratni klic metode *onNotification()*, lahko ob ustreznem obvestilu dodajamo vozlišča in njihove vrednosti v OM2M.

Vozlišča dodajamo ob obvestilu tipa `NODE_PROTOCOL_INFO`. Za dodajanje vozlišč uporabimo metodo *addNode()* iz razreda `ZWaveMonitor`. Ta sprejme



Slika 4.3: Postopek izvajanja vtičnika za Z-Wave.

niz znakov, ki se uporabi za ime vsebnika vozlišča. Ime je sestavljeno iz tipa vozlišča in njegovega identifikatorja, ki sta ločena s "_".

Vrednosti vozlišč dodajamo ob obvestilih tipa VALUE_ADDED, VALUE_REMOVED, VALUE_CHANGED ali VALUE_REFRESHED. V vseh primerih se kliče metoda *addNodeValue()* iz razreda ZWaveMonitor, ki sprejme ime vozlišča (v takšni obliki kot smo prej napisali) in vrednost, predstavljeno v formatu oBIX. V vrednost vozlišča zapišemo razred ukaza, instanco, indeks, vrsto, tip, oznako in vrednost vozlišča.

Controller

Razred je enak kot pri prejšnjih dveh vtičnikih, le da podpira druge operacije. Operacije, ki jih podpira, so: pričetek in zaustavitev dodajanja vozlišč (ang. *inclusion*), pričetek in zaustavitev odstranjevanja vozlišč in vklop/izklop vseh vozlišč v omrežju Z-Wave. Niti dodajanja in odstranjevanja vozlišč tečejo v zankah, dokler so ustrezne spremenljivke *boolean* enake *true*. Za dodajanje se uporablja spremenljivka *adding*, za odstranjevanje *removing*. Operacija za pričetek vključevanja vozlišč postavi *adding* na *true* in začne novo nit. Pri zaustavitvi vključevanja vozlišč se *adding* postavi na *false* in vključevanje se zaustavi ob naslednji iteraciji. Ekvivalentno je pri odstranjevanju vozlišč, le da upravljamo s spremenljivko *removing*.

Poglavje 5

Testiranje vtičnikov

Namen tega poglavja je prikazati delovanje razvitih vtičnikov. Implementirane vtičnike smo preizkusili z uporabo naprav, ki so naštete v sekciji 3.4.

5.1 Priprava okolja

Vsi preizkusi so opravljeni na osebnem računalniku na operacijskem sistemu Linux (distribucija Debian). V osebni računalnik smo priklopili krmilno napravo za omrežje Z-Wave in vmesnik Bluetooth.

Vtičnike smo dodali vozlišču MN, ki ga ponuja platforma OM2M. Platforma OM2M je projekt zgrajen z orodjem Maven, zato jo zaženemo tako, da poženemo vozlišči IN in MN, ki sta rezultat gradnje projekta. Najprej zaženemo IN in nato MN, ki se registrira v IN. Vozlišči sta predstavljeni s platformo OSGi, preko katere poženemo vtičnike.

5.2 Preizkus vtičnika za Bluetooth

Vtičnik pripada vozlišču MN, ki ga ponuja platforma OM2M. Ob zagonu vtičnika se najprej ustvarijo viri:

- entiteta AE z imenom BLUETOOTH (prikaz zahtevka, ki se pošlje na CSE, je prikazan v izpisu 5.1),

- kjer se ustvari vsebnik DESCRIPTOR (zahtevek je prikazan v izpisu 5.2), znotraj katerega se ustvari instanca vsebine, ki vsebuje operacijo *Start Inquiry* (zahtevek je prikazan v izpisu 5.3) in
- vsebnik DEVICES za zaznane naprave.

```
RequestPrimitive [operation=1,
 to=/mn-cse,
 from=admin:admin,
 resourceType=2,
 name=BLUETOOTH,
 content=org.eclipse.om2m.commons.resource.AE@1180fdb,
 returnContentType=application/obj,
 requestContentType=application/obj,
 ]
```

Izpis 5.1: Zahtevek za kreiranje AE, ki se pošlje na CSE.

```
RequestPrimitive [operation=1,
 to=/mn-cse/mn-name/BLUETOOTH,
 from=admin:admin,
 resourceType=3,
 name=DESCRIPTOR,
 content=org.eclipse.om2m.commons.resource.Container@16d9b56,
 returnContentType=application/obj,
 requestContentType=application/obj,
 ]
```

Izpis 5.2: Zahtevek za kreiranje vsebnika DESCRIPTOR, ki se pošlje na CSE.

```
RequestPrimitive [operation=1,
 to=/mn-cse/mn-name/BLUETOOTH/DESCRIPTOR,
 from=admin:admin,
 resourceType=4,
 content=org.eclipse.om2m.commons.resource.ContentInstance@e02cf4,
 returnContentType=application/obj,
 requestContentType=application/obj,
 ]
```

Izpis 5.3: Zahtevek za kreiranje instance vsebine v vsebniku DESCRIPTOR, ki se pošlje na CSE.

Po kreiranju virov se zažene iskanje naprav in storitev.

Iskanje najde mobilno napravo, ki smo jo uporabili za testiranje. Mobilna naprava ima naslov MAC `2C8A72563AA3` in ime `SS90`. Najdena naprava se vnese v OM2M tako, da se ustvari vsebnik z imenom `2C8A72563AA3`. Znotraj tega vsebnika se ustvarita vsebnik za opis naprave DESCRIPTOR in vsebnik za prebrane podatke DATA. Znotraj vsebnika DESCRIPTOR se ustvari vsebnik SERVICES, kjer se hranijo vse storitve, ki jih ponuja naprava. V vsebniku DESCRIPTOR se ustvari tudi instanca vsebine, ki je prikazana v izpisu 5.4.

```
1 <obj>
2     <str val="SS90" name="Device name"/>
3     <int val="512" name="Major class"/>
4     <int val="12" name="Minor class"/>
5     <int val="16" name="Number of services"/>
6 </obj>
```

Izpis 5.4: Vsebina zahtevka za dodajanje opisa naprave.

Sledi iskanje storitev na napravi. Tiste, ki so bile najdene, so prikazane v izpisu 5.5.

```
1  btl2cap://2C8A72563AA3:001f;authenticate=false;encrypt=false;master=false
2  10000 No value
3  btl2cap://2C8A72563AA3:001f;authenticate=false;encrypt=false;master=false
4  10001 No value
5  btspp://2C8A72563AA3:2;authenticate=false;encrypt=false;master=false
6  10002 Headset Gateway
7  btspp://2C8A72563AA3:3;authenticate=false;encrypt=false;master=false
8  10003 Handsfree Gateway
9  btl2cap://2C8A72563AA3:0017;authenticate=false;encrypt=false;master=false
10 10004 AV Remote Control Target
11 btl2cap://2C8A72563AA3:0019;authenticate=false;encrypt=false;master=false
12 10005 Advanced Audio
13 btl2cap://2C8A72563AA3:0017;authenticate=false;encrypt=false;master=false
14 10006 No value
15 btl2cap://2C8A72563AA3:000f;authenticate=false;encrypt=false;master=false
16 10007 Android Network Access Point
17 btl2cap://2C8A72563AA3:000f;authenticate=false;encrypt=false;master=false
18 10008 Android Network User
19 btgoep://2C8A72563AA3:19;authenticate=false;encrypt=false;master=false
```

```
20 1000a OBEX Phonebook Access Server
21 btgoep://2C8A72563AA3:16;authenticate=false;encrypt=false;master=false
22 1000b MAP SMS/MMS
23 btgoep://2C8A72563AA3:17;authenticate=false;encrypt=false;master=false
24 1000c MAP EMAIL
25 btspp://2C8A72563AA3:15;authenticate=false;encrypt=false;master=false
26 1000d SIM Access Server
27 btgoep://2C8A72563AA3:12;authenticate=false;encrypt=false;master=false
28 1000e OBEX Object Push
29 btgoep://2C8A72563AA3:20;authenticate=false;encrypt=false;master=false
30 1000f OBEX File Transfer
31 btspp://2C8A72563AA3:4;authenticate=false;encrypt=false;master=false
32 10010 BTSPPServer
33 btspp://2C8A72563AA3:5;authenticate=false;encrypt=false;master=false
34 10011 FuturedialDMI
```

Izpis 5.5: Storitve na uporabljeni mobilni napravi. Vsaka storitev obsega dve vrstici. V prvi je povezava do storitve, v drugi sta oprimek in ime storitve.

Vse najdene storitve se vnesejo v OM2M. Za vsako storitev se ustvari instanca vsebine v vsebniku SERVICES. Instanca vsebine storitve je predstavljena z oprimkom, imenom, povezavo in operacijo, ki odpre povezavo na storitev. Vsebina zahtevka, ki se pošlje na CSE, je predstavljena s formatom oBIX. V izpisu 5.6 je prikazan primer zapisa storitve *BTSPPServer*.

```
1 <obj>
2     <str val="BTSPPServer" name="Service name"/>
3     <str val="10010" name="Record handle"/>
4     <str val="btspp://2C8A72563AA3:4;authenticate=false;encrypt=false;master
        =false" name="Connection URL"/>
5     <op href="/in-cse/in-name/AUTODETECTION?op=open_connection&amp;href=
        btspp%3A%2F%2F2C8A72563AA3%3A4%3Bauthenticate%3Dfalse%3Bencrypt%3
        Dfalse%3Bmaster%3Dfalse" name="Open connection" is="execute"/>
6 </obj>
```

Izpis 5.6: Vsebina zahtevka za dodajanje opisa storitve *BTSPPServer* v formatu oBIX.

Ko so naprave in storitve dodane v OM2M, se lahko prične branje vrednosti. Za pošiljanje podatkov z naprave uporabimo aplikacijo Android, ki je nameščena na mobilni napravi. Aplikacija izpostavi (poleg ostalih storitev) storitev *BTSPPServer* (glej 3.4). Povezavo do storitve odpremo tako,

Attribute	Value
ty	4
ri	/mn-cse/cin-890200848
pi	/mn-cse/cnt-222095193
ct	20160924T111354
lt	20160924T111354
st	0
cnf	text/plain:0
cs	466
con	

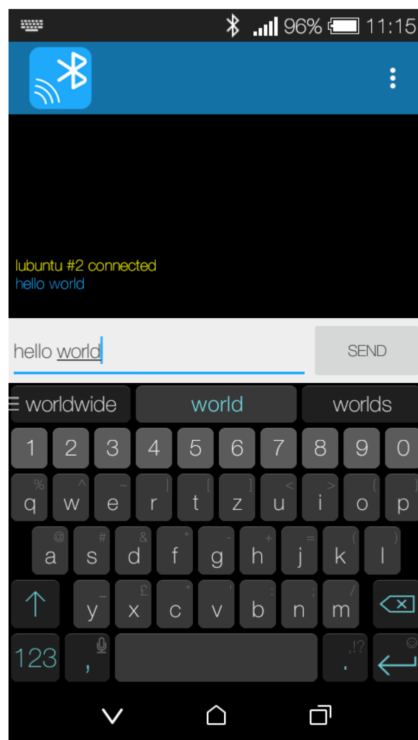
Attribute	Value
Service name	BTSPPServer
Record handle	1000a
Connection URL	btspp://2C8A72563AA3:4;authenticate=false;encrypt=false;master=false
Open connection	/mn-cse/mn-name/BLUETOOTH?op=open_connection&href=btspp%3A%2F %2F2C8A72563AA3%3A4%3Bauthenticate%3Dfalse%3Bencrypt%3Dfalse%3Bmaster%3Dfalse

Successful POST request.

Name	Value
Status	Connection opened.

Slika 5.1: Odpiranje povezave na storitev *BTSPPServer* preko spletnega vmesnika.

da (preko vmesnika REST) sprožimo operacijo *Open Connection* storitve *BTSPPServer* (prikazano na sliki 5.1). V tem času mora biti aplikacija na mobilni napravi zagnana. Na napravi se ob odprtju povezave pojavi obvestilo za dovoljenje za dostop do naprave (zahtevek za sparitev naprave in lokalnega vmesnika). Ko na napravi potrdimo uparjanje, lahko začnemo s pošiljanjem podatkov. V konkretnem primeru gre za navaden tekst, ki ga uporabnik vnese v aplikacijo. Na sliki 5.2 je prikazan vnos niza *hello world*, ki se vnese v sistem OM2M. Na sliki 5.3 je prikazan vnesen podatek v sistemu OM2M.

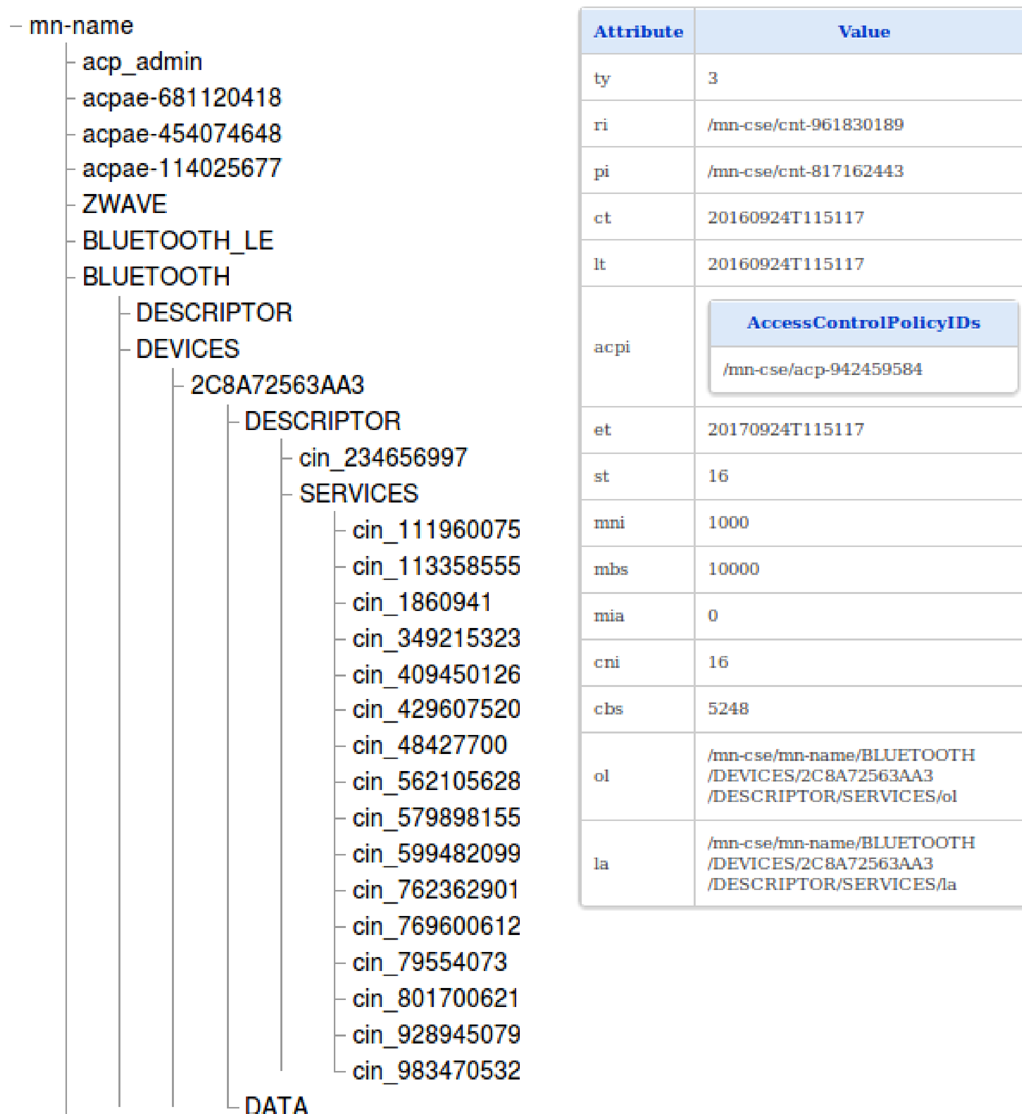


Slika 5.2: Pošiljanje niza *hello world* iz aplikacije *Bluetooth SPP Server Terminal* na mobilni napravi.

<table>
<tr><th>Attribute</th><th>Value</th></tr>
<tr><td>ty</td><td>4</td></tr>
<tr><td>ri</td><td>/mn-cse/cin-249649093</td></tr>
<tr><td>pi</td><td>/mn-cse/cnt-578183924</td></tr>
<tr><td>ct</td><td>20160924T111459</td></tr>
<tr><td>lt</td><td>20160924T111459</td></tr>
<tr><td>st</td><td>0</td></tr>
<tr><td>cnf</td><td>text/plain:0</td></tr>
<tr><td>cs</td><td>111</td></tr>
<tr><td>con</td><td><table><tr><th>Attribute</th><th>Value</th></tr><tr><td>Value</td><td>hello world</td></tr></table></td></tr>
</table>

Slika 5.3: Prebrana vrednost z mobilne naprave, prikazana v spletni aplikaciji OM2M.

Na sliki 5.4 je prikazana struktura virov po dodajanju vseh podatkov z mobilne naprave.



Slika 5.4: Struktura virov aplikacije BLUETOOTH po vnosu vseh podatkov z mobilne naprave.

5.3 Preizkus vtičnika za BLE

Vtičnik za BLE najprej ustvari AE z imenom BLUETOOTH_LE. V AE se ustvarita vsebnik z imenom DESCRIPTOR in vsebnik z imenom DEVICES.

V prvega se shrani instanca vsebine, ki vsebuje štiri operacije za upravljanje z vmesnikom. Drugi se uporablja za hranjenje vsebnikov naprav.

Zažene se iskanje naprav. Vmesnik najde pametno zapestnico BLE z naslovom MAC 08:7C:BE:8B:9E:BB. Naprava se kot vsebnik doda v vsebnik DEVICES. Vsebnik naprave se imenuje po naslovu MAC naprave, vendar se pred ustvarjanjem vira popravi naslov, da ne vsebuje dvopičij. Tako se ustvari vsebnik z imenom 08-7C-BE-8B-9E-BB. Vsebniku naprave se dodata vsebnik DESCRIPTOR za opis naprave in vsebnik SERVICES za storitve. V vsebniku DESCRIPTOR se ustvari instanca vsebine z opisom naprave, prikazanem v izpisu 5.7.

```
1 <obj>
2     <str val="08:7C:BE:8B:9E:BB" name="Device MAC address"/>
3     <str val=" Quintic BLE" name="Device name"/>
4 </obj>
```

Izpis 5.7: Opis pametne zapestnice v formatu oBIX.

Sledi iskanje storitev, karakteristik in deskriptorjev na napravi. Na napravi najdemo storitve, prikazane v izpisu 5.8, karakteristike, prikazane v izpisu 5.9, in deskriptorje, prikazane v izpisu 5.10.

```
1 attr handle = 0x0001, end grp handle = 0x000b uuid:
      00001800-0000-1000-8000-00805f9b34fb
2 attr handle = 0x000c, end grp handle = 0x000f uuid:
      00001801-0000-1000-8000-00805f9b34fb
3 attr handle = 0x0010, end grp handle = 0x0016 uuid: 0000fee8
      -0000-1000-8000-00805f9b34fb
4 attr handle = 0x0017, end grp handle = 0x001d uuid: 0000fee9
      -0000-1000-8000-00805f9b34fb
```

Izpis 5.8: Storitve, najdene na pametni zapestnici BLE.

```
1 handle = 0x0002, char properties = 0x0a, char value handle = 0x0003, uuid =
      00002a00-0000-1000-8000-00805f9b34fb
2 handle = 0x0004, char properties = 0x02, char value handle = 0x0005, uuid =
      00002a01-0000-1000-8000-00805f9b34fb
```

```
3 handle = 0x0006, char properties = 0x0a, char value handle = 0x0007, uuid =
      00002a02-0000-1000-8000-00805f9b34fb
4 handle = 0x0008, char properties = 0x02, char value handle = 0x0009, uuid =
      00002a04-0000-1000-8000-00805f9b34fb
5 handle = 0x000a, char properties = 0x0e, char value handle = 0x000b, uuid =
      00002a03-0000-1000-8000-00805f9b34fb
6 handle = 0x000d, char properties = 0x22, char value handle = 0x000e, uuid =
      00002a05-0000-1000-8000-00805f9b34fb
7 handle = 0x0011, char properties = 0x10, char value handle = 0x0012, uuid =
      003784cf-f7e3-55b4-6c4c-9fd140100a16
8 handle = 0x0015, char properties = 0x04, char value handle = 0x0016, uuid =
      013784cf-f7e3-55b4-6c4c-9fd140100a16
9 handle = 0x0018, char properties = 0x0c, char value handle = 0x0019, uuid =
      d44bc439-abfd-45a2-b575-925416129600
10 handle = 0x001b, char properties = 0x10, char value handle = 0x001c, uuid =
      d44bc439-abfd-45a2-b575-925416129601
```

Izpis 5.9: Karakteristike, najdene na pametni zapestnici BLE.

```
1 handle = 0x000f, uuid = 00002902-0000-1000-8000-00805f9b34fb
2 handle = 0x0013, uuid = 00002902-0000-1000-8000-00805f9b34fb
3 handle = 0x0014, uuid = 00002901-0000-1000-8000-00805f9b34fb
4 handle = 0x001a, uuid = 00002901-0000-1000-8000-00805f9b34fb
5 handle = 0x001d, uuid = 00002902-0000-1000-8000-00805f9b34fb
```

Izpis 5.10: Vsi deskriptorji na pametni zapestnici BLE.

Vsi prebrani podatki se vnesejo v OM2M. Najprej se v vsebnik SERVICES vnesejo storitve. Vsebnik posamezne storitve je poimenovan z UUID storitve. V vsebniku storitve se ustvarita vsebnik DESCRIPTOR za opis storitve in vsebnik CHARACTERISTICS za karakteristike storitve. V vsebniku DESCRIPTOR se ustvari instanca vsebine z opisom storitve. Izpis 5.11 prikazuje primer vsebine zahtevka za dodajanje opisa storitve.

```
1 <obj>
2     <str val="00001800-0000-1000-8000-00805f9b34fb" name="Service UUID"/>
3     <str val="0x0001" name="Attr handle"/>
4     <str val="0x000b" name="End grp handle"/>
5     <str val="Generic Access" name="Service name"/>
6 </obj>
```

Izpis 5.11: Vsebina zahtevka za kreiranje opisa storitve *Generic Access* v formatu oBIX.

Nato se v vsebnik CHARACTERISTICS ustrezne storitve vnesejo karakteristike. Ime vsebnika karakteristike je predstavljeno z UUID karakteristike. V vsebniku karakteristike se ustvarijo vsebnik DESCRIPTOR za opis, vsebnik DATA za vrednosti in vsebnik BLE_DESCRIPTORS za deskriptorje. V vsebniku DESCRIPTOR se ustvari instanca vsebine, ki opisuje karakteristiko. Izpis 5.12 prikazuje primer vsebine zahtevka za dodajanje opisa karakteristike.

```
1 <obj>
2     <str val="00002a00-0000-1000-8000-00805f9b34fb" name="Characteristic
          UUID"/>
3     <str val="0x0002" name="Handle"/>
4     <str val="0x0003" name="Value handle"/>
5     <str val="Device Name" name="Characteristic name"/>
6     <str val="Write, Read, " name="Properties"/>
7 </obj>
```

Izpis 5.12: Vsebina zahtevka za kreiranje opisa karakteristike *Device Name* v formatu oBIX.

Po karakteristikah se v vsebnik BLE_DESCRIPTORS posamezne karakteristike vnesejo njeni deskriptorji. Vsak deskriptor je predstavljen z instanco vsebine. Izpis 5.13 prikazuje primer vsebine zahtevka, ki se pošlje za dodajanje deskriptorja.

```
<obj>
    <str val="00002902-0000-1000-8000-00805f9b34fb" name="Descriptor UUID
        "/>
    <str val="00 00 " name="Descriptor value"/>
    <str val="Client Characteristic Configuration" name="Descriptor name
        "/>
</obj>
```

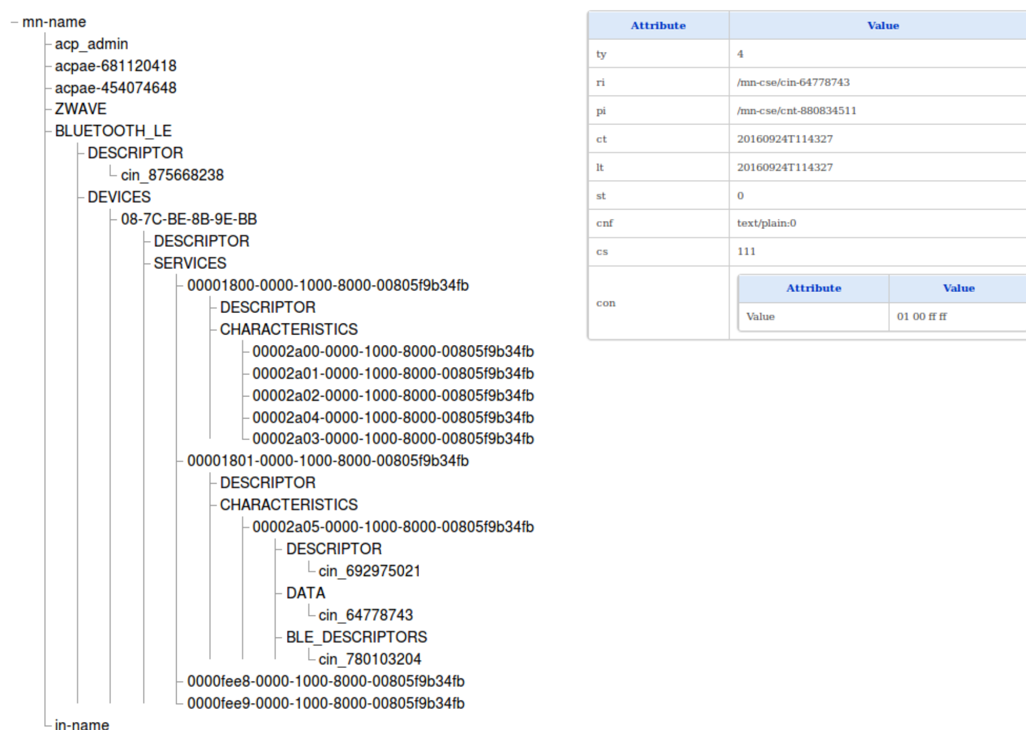
Izpis 5.13: Vsebina zahtevka za dodajanje deskriptorja karakteristike v formatu oBIX.

Ko so vneseni vsi opisni podatki naprave, pomeni, da so se ustvarili vsi potrebni vsebniki za vnos podatkov oz. vrednosti karakteristik. Prične se osveževanje vrednosti na napravah, ki so že bile vnesene v sistem. Podatki se vnašajo kot instance vsebine v vsebnik DATA, ki pripada ustreznima vsebnikoma karakteristike in storitve. Izpis 5.14 prikazuje primer vsebine zahtevka za dodajanje vrednosti karakteristike.

```
<obj>
    <str val="64 00 c8 00 00 00 d0 07" name="Value"/>
</obj>
```

Izpis 5.14: Vsebina zahtevka za vnos vrednosti karakteristike v formatu oBIX.

Na sliki 5.5 je prikazana struktura virov na koncu dodajanja vseh podatkov s pametne zapestnice BLE.



Slika 5.5: Struktura virov aplikacije BLUETOOTH_LE po vnosu vseh podatkov s pametne zapestnice BLE.

5.4 Preizkus vtičnika za Z-Wave

Na začetku izvajanja vtičnika se ustvari AE z imenom ZWAVE. V AE se ustvarita vsebnik DESC za operacije (in opisne stvari) in vsebnik DEVICES za hranjenje vozlišč. V izpisu 5.15 je prikazana vsebina zahtevka za kreiranje instance vsebine v vsebniku DESC.

```
<obj>
    <op href="/in-cse/in-name/zwave?op=start_inclusion" name="Start
        inclusion of nodes" is="execute"/>
    <op href="/in-cse/in-name/zwave?op=stop_inclusion" name="Stop inclusion
        of nodes" is="execute"/>
    <op href="/in-cse/in-name/zwave?op=on" name="Switch all on" is="execute
        "/>
    <op href="/in-cse/in-name/zwave?op=off" name="Switch all off" is="
        execute"/>
</obj>
```

Izpis 5.15: Vsebina zahtevka za kreiranje instance vsebine v vsebniku DESC, ki vsebuje operacije za nadzor vmesnika Z-Wave.

Sledi zagon vmesnika in poslušanje omrežja Z-Wave. Vozlišče mora najprej biti del omrežja Z-Wave, zato moramo žarnico dodati v omrežje. To storimo tako, da sprožimo operacijo dodajanja vozlišč *start_inclusion* preko spletne aplikacije OM2M. Ob tem se vklopi vključevanje vozlišč v omrežje in takrat prižgemo žarnico, ki jo krmilnik zazna in samodejno doda v omrežje. Po vključitvi žarnice v omrežje lahko zaključimo vključevanje vozlišč s proženjem operacije *stop_inclusion* preko spletne aplikacije OM2M.

Vozlišča Z-Wave se v OM2M dodajajo ob prejetju obvestila tipa NODE_PROTOCOL_INFO. Ustvari se vsebnik z imenom, ki je sestavljeno iz identifikatorja naprave in tipa naprave. Ustvarita se dva vsebnika, prvi z imenom Static_PC_Controller_1, ki predstavlja krmilno napravo, in drugi z imenom Multilevel_Power_Switch_2, ki predstavlja pametno žarnico LED. V vsakem vsebniku naprave se ustvarita vsebnik DESC za opis vozlišča in vsebnik DATA za podatke vozlišča. Ob prejetju obvestila tipov VALUE_ADDED, VALUE_REMOVED, VALUE_CHANGED in VALUE_REFRESHED se vrednosti vnesejo v sistem v vsebnik DATA kot instance vsebine. V izpisu 5.16 je prikazan primer vsebine zahtevka za dodajanje teh vrednosti.

```
1 <obj>
2     <int val="32" name="commandClass"/>
3     <int val="1" name="instance"/>
4     <int val="0" name="index"/>
5     <str val="BASIC" name="genre"/>
6     <str val="BYTE" name="type"/>
7     <str val="" name="label"/>
8     <str val="21" name="value"/>
9 </obj>
```

Izpis 5.16: Vsebina zahtevka za kreiranje instance vsebine v vsebniku DATA.

Na sliki 5.6 je prikazana struktura virov na koncu dodajanja podatkov iz omrežja Z-Wave.

- mn-name
  - acp_admin
  - acpae-681120418
  - acpae-454074648
  - acpae-114025677
  - ZWAVE
    - DEVICES
      - Static_PC_Controller_1
      - Multilevel_Power_Switch_2
        - DESCRIPTOR
        - DATA
          - cin_320753809
          - cin_511032305
          - cin_549189798
          - cin_725449874
          - cin_976441793
          - cin_600164676
          - cin_692496736
          - cin_721144770
          - cin_757460271
          - cin_819215232
          - cin_937427593
          - cin_940069546
          - cin_173977795
          - cin_386985820
          - cin_578010171
          - cin_319471148
          - cin_323636551
          - cin_550906440
          - cin_59123415
          - cin_712287634
    - DESCRIPTOR
      - cin_342187476

Attribute	Value
ty	3
ri	/mn-cse/cnt-19425064
pi	/mn-cse/cnt-416461942
ct	20160924T113425
lt	20160924T113425
acpi	AccessControlPolicyIDs: /mn-cse/acp-838798030
et	20170924T113425
st	0
mni	1000
mbs	10000
mia	0
cni	0
cbs	0
ol	/mn-cse/mn-name/ZWAVE/DEVICES /Multilevel_Power_Switch_2 /DESCRIPTOR/ol
la	/mn-cse/mn-name/ZWAVE/DEVICES /Multilevel_Power_Switch_2 /DESCRIPTOR/la

Slika 5.6: Struktura virov aplikacije ZWAVE po vnosu podatkov s pametne žarnice LED.

Poglavje 6

Semantika

Semantika je v IoT ključnega pomena za interoperabilnost naprav in popolno avtonomnost sistemov. Za popolno avtonomnost sistema mora, v primeru samodejnega zaznavanja naprav, aplikacija vedeti pomen podatka, ki ga prebere z naprave. Na ta način lahko aplikacija samodejno dodaja naprave v sistem in prebrane podatke pravilno razvršča v podatkovni model. S tem imajo tudi ostale naprave v sistemu dostop do podatkov, za katere vedo točen pomen.

To poglavje vključujemo v delo, ker želimo poudariti, da je za popolno avtonomnost sistema s samodejno zaznavo naprav potrebna semantika. Izvedbo samodejne zaznave naprav smo prikazali v poglavju 4. Trenutna različica platforme OM2M ne podpira semantike, zato v tem poglavju opišemo, kako naj bi potekal prevod podatkovnega modela posamezne tehnologije v podatkovni model sistema oneM2M. Opisana je ontologija sistema oneM2M Base Ontology, razredi v njej in kako jo uporabljamo.

6.1 Ontologija

Ontologija je formalno poimenovanje in definiranje tipov, lastnosti in odnosov med entitetami, ki obstajajo v določeni domeni diskurza [12]. Z drugimi besedami, ontologija je slovar s strukturo, kjer se slovar nanaša na določeno

domeno oz. področje in vsebuje koncepte, ki se uporabljajo znotraj te domene oz. področja (npr. koncept naprave (ang. *Device*) ima dogovorjen, dobro definiran pomen znotraj področja v ontologiji *Smart Appliances REFerence* (SAREF[1])).

Ontologija je sestavljena iz lastnosti (ang. *property*) in razredov (ang. *class*). Lastnosti predstavljajo razmerja in povezujejo posameznike iz določene domene (ang. *domain*; razred) s posamezniki iz določenega obsega (ang. *range*; tudi razred). Obstajata dva tipa lastnosti: lastnosti objekta (ang. *object properties*) in lastnosti podatkov (ang. *data properties*). Lastnost objekta opisuje razmerje med dvema posameznima objektoma. Lastnost podatka opisuje razmerje med posameznim objektom in konkretnimi podatkovnimi vrednostmi, ki so lahko tipizirane ali netipizirane.

Razredi so interpretirani kot množice posameznikov in tudi kot konkretne predstavitve konceptov.

Koncepti ne predstavljajo posameznikov, ampak razrede posameznikov, zato se koncepti v OWL[2] imenujejo razredi.

Struktura ontologije je predstavljena z dogovorjenimi, dobro definiranimi razmerji med koncepti.

Lastnost objekta povezuje razred domene (ang. *domain class*) z razredom obsega (ang. *range class*):

$$Domain\ Class \rightarrow Object\ Property \rightarrow Range\ Class$$

Primer 1: V ontologiji SAREF lastnost objekta *dovrši* (ang. *accomplishes*) povezuje *napravo* (*device*) z *opravilom* (*task*).

$$device \rightarrow accomplishes \rightarrow task$$

Lastnost podatka povezuje objekt s podatkom.

Primer 2: V ontologiji SAREF podatkovna lastnost *imaProizvajalca* (*has-*

[1]Ontologija SAREF je skupni model konsenza, ki omogoča ujemanje obstoječih sredstev (standardov, protokolov, podatkovnih modelov itd.) v domeni pametnih naprav.

[2]W3C OWL oz. Web Ontology Language je standardni jezik za ontologije.

Manufacturer) povezuje napravo s podatkom podatkovnega tipa *literal* (podatek je v taki obliki kot je zapisan).

$$device \rightarrow hasManufacturer \rightarrow literal$$

Tretji tip lastnosti v OWL je lastnost oznake. Lastnost oznake se uporablja za zagotavljanje dodatnih informacij o elementih v ontologiji (npr. instance, razredi). Označimo lahko avtorja, verzijo ali zapišemo komentar. Objekt lastnosti oznake je lahko dobeseden podatek, referenca URI ali posameznik.

6.2 Ontologija v oneM2M

Ontologije in njihove predstavitve z OWL so uporabljene v oneM2M zaradi zagotavljanja sintaktične in semantične interoperabilnosti sistema oneM2M z zunanjimi sistemi. Pričakuje se, da so zunanji sistemi prav tako opisani z ontologijami.

OneM2M je določil eno ontologijo in ta se imenuje *oneM2M Base Ontology*. Od zunanjih organizacij in podjetij se pričakuje, da bodo prispevali svoje ontologije, katere bo možno preslikati (preko podrazredov, ekvivalenc ...) v oneM2M Base Ontology.

Ontologija Base Ontology je zasnovana tako, da zagotavlja minimalno število konceptov, razmerij in omejitev, ki so potrebne za semantično odkrivanje entitet v sistemu oneM2M. Da lahko entitete v sistemu oneM2M odkrijemo, morajo biti semantično opisane z razredi (koncepti) v ontologiji nekega standarda/proizvajalca/tehnologije in ti razredi (koncepti) morajo biti povezani z nekaterimi razredi v ontologiji Base Ontology kot podrazredi.

Base Ontology omogoča tehnologijam non-oneM2M, da naredijo izpeljane ontologije za opisovanje svojih podatkovnih modelov z namenom vzajemnega delovanja s sistemom oneM2M.

Base Ontology vsebuje le razrede in lastnosti, ne pa tudi instanc, ker se Base Ontology (in izpeljane ontologije) uporabljajo le za semantični opis entitet v sistemu oneM2M. Instanciranje (podatki posameznih entitet v sistemu oneM2M - npr. naprave, stvari itd.) se izraža preko virov oneM2M.

6.2.1 Sintaktična interoperabilnost

Sintaktična interoperabilnost se v glavnem uporablja za vzajemno delovanje z napravami, ki so v območnem omrežju (ang. *Area Network*) in niso del oneM2M. Ta omrežja so v našem primeru omrežja izbranih brezžičnih tehnologij. V tem primeru se ontologija, ki je predstavljena z datoteko OWL in vsebuje komunikacijske parametre (npr. imena operacij, imena vhodnih/izhodnih parametrov, njihovi tipi in strukture ...) specifične za omrežje, uporabi za nastavljanje IPE.

S pomočjo datoteke OWL je IPE zmožna poiskati vire oneM2M (AE, vsebnike), ki so strukturirani po specifikacijah omrežja. To omogoča entitetam oneM2M branje/pisanje v/iz virov tako, da IPE serializira podatke in jih pošlje napravi v omrežju (ali jih od naprave prejme).

Pomen virov je implicitno opredeljen s strani tehnologije omrežja. Vsaka ontologija, ki opisuje določen tip vzajemno-delujočega omrežja, mora izhajati iz ontologije oneM2M Base Ontology. Tipi naprav ontologije vzajemno-delujočega omrežja se morajo preslikati v koncept *Interworked Device* ontologije oneM2M Base Ontology.

6.2.2 Semantična interoperabilnost

Semantična interoperabilnost se uporablja predvsem za opisovanje funkcionalnosti storitev, ki jih ponujajo naprave skladne z oneM2M (naprave M2M).

Primer: imamo različne pralne stroje, ki so skladni z oneM2M in vsi imajo funkcijo pranja, sušenja, izbire temperature itd., ampak viri oneM2M (vsebniki), preko katerih dostopamo do teh funkcionalnosti, imajo lahko različna imena virov, strukturo in vsebino. V takem primeru ontologija – predstavljena kot datoteka OWL – vsebuje določene tipe aplikacijskih storitev M2M in/ali skupnih storitev naprave M2M, skupaj s funkcionalnostjo te storitve (npr. funkcija pranja).

Vsaka ontologija, ki opisuje določen tip naprave M2M, mora biti izpeljana iz ontologije oneM2M Base Ontology. Naprava se mora preslikati v koncept

Device oz. *Naprava* ontologije oneM2M Base Ontology.

6.2.3 Prevedba podatkovnega modela non-oneM2M v podatkovni model oneM2M

Obstajajo trije načini, kako podpreti vzajemno delovanje IPE preko referenčne točke Mca:

1. **Vzajemno delovanje s popolno preslikavo semantike podatkovnega modela non-oneM2M na Mca.**
   Ta način podpira popolno semantično vzajemno delovanje podatkovnega modela, ki ga uporablja rešitev non-oneM2M, in splošnega podatkovnega modela, ki ga uporablja oneM2M za izmenjavo podatkov med aplikacijami.

   IPE vsebuje vzajemno-delujočo logiko protokola. V primeru kompleksnega podatkovnega modela non-oneM2M, se lahko to odraža v zapletenosti strukture osnovnih virov oneM2M, ki jo zgradi IPE na CSE. Ti viri predstavljajo podatkovni model non-oneM2M na način oneM2M in so dostopni preko Mca. Omogočajo dostop entitetam CSE in AE do entitet v podatkovnem modelu non-oneM2M preko IPE.

2. **Vzajemno delovanje z uporabo vsebnikov za transparenten prenos kodiranih podatkov non-oneM2M in ukazov preko Mca.**
   Pri tem načinu IPA transparentno zapakira podatke in ukaze v vsebnike, da jih entitete CSE in AE lahko uporabijo. V tem primeru morajo CSE in AE poznati pravila kodiranja, da lahko dekodirajo vsebino vsebnikov.

3. **Vzajemno delovanje z uporabo preusmeritvenega mehanizma.**
   Ta način podpira prehodni sistem, ki zna preslikati operacije iz sveta oneM2M v svet non-oneM2M. CSE ali AE zagotovita preslikan vmesnik kot strukturo virov oneM2M in ko pride do proženja operacije v strukturi, se operacije preusmerijo v IPE.

6.2.4 Uporaba ontologije oneM2M Base Ontology

Vsaka instanca razreda ontologije oneM2M Base Ontology je instancirana v atributu *descriptor* v viru tipa *semanticDescriptor*. Vir *semanticDescriptor* lahko instancira več razredov in vsebuje:

- Instance razredov v obliki RDF. Vsaka instanca razreda je globalno določena znotraj rešitve oneM2M z uporabo atributa *rdf:about*. Ta vsebuje URI (npr. naslov MAC naprave), ki je unikaten znotraj rešitve oneM2M.
- Atribut *ontologyRef*, ki identificira razred, katerega instanca je opisana v atributu *descriptor*. Glede na instanciranje je to lahko razred v ontologiji Base Ontology ali razred v drugi ontologiji, ki je podrazred razreda v Base Ontology.
- Instancirane lastnosti objekta, kjer je instanciran razred razred domene.
- Instancirane lastnosti podatkov, kjer je instanciran razred razred domene.

Če je razred obsega od lastnosti objekta instanciran v viru *semanticDescriptor*, ki je različen od vira *semanticDescriptor*, v katerem je instanciran razred domene, potem mora vir *semanticDescriptor* razreda domene vsebovati instanco lastnosti pripisa *resourceDescriptorLink*, ki vsebuje URI vira *semanticDescriptor* instance razreda obsega.

Vsak razred, ki je izpeljan iz zunanje ontologije, mora biti podrazred ontologije iz Base Ontology ali razred obsega neke lastnosti objekta, katere razred domene je razred (npr. *class:Thing*) ali podrazred ontologije Base Ontology.

Če je razred izpeljan iz zunanje ontologije podrazred razreda iz Base Ontology, potem je instanciran na enak način kot razred iz Base Ontology.

Če razred, izpeljan iz zunanje ontologije, ni podrazred ontologije Base Ontology, ampak je razred dosega neke lastnosti objekta, katerega razred domene je razred ali podrazred ontologije Base Ontology, potem je instanciran

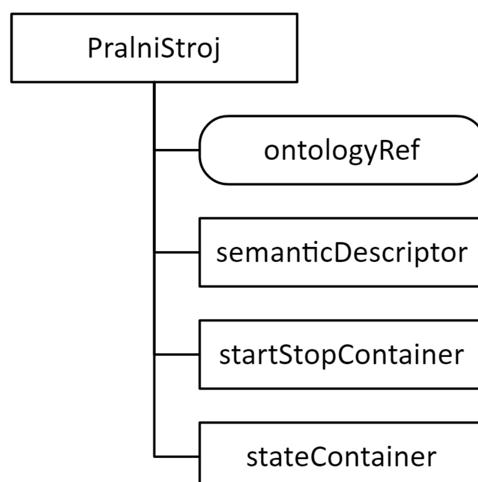
v podatkih atributa *descriptor* vira *semanticDescriptor*, ki je otrok vira oneM2M, ki instancira podrazred ontologije Base Ontology.

Razredi in instanciranje posameznih razredov v ontologiji Base Ontology so opisani v dodatku A.10.

6.2.5 Primer preslikave ontologije SAREF v oneM2M

Preslikava ontologije v oneM2M opiše, kako se instanca te ontologije predstavi v strukturi virov sistema oneM2M.

Preslikava poteka po pravilih, ki so opisana v razdelku 6.2.4. Na sliki 6.1 je prikazana struktura vsebnika, ki predstavlja pametni pralni stroj. Sestavljen je iz atributa *ontologyRef*, ki vsebuje URI koncepta ontologije pametnega pralnega stroja, npr. "http://www.tno.com/saref#WashingMachine". Vsebnika *startStopContainer* in *stateContainer* predstavljata funkcijski vmesnik pralnega stroja. Stroj lahko zaženemo, ustavimo, ali pridobimo njegovo trenutno stanje.



Slika 6.1: Struktura vsebnika *PralniStroj*, ki predstavlja pametni pralni stroj v sistemu oneM2M.

V izpisu 6.1 je prikazana vsebina vira *semanticDescriptor*. Vsebina vira predstavlja povezavo med operacijami, ki jih je mogoče izvesti na pralnem stroju, in vsebniki, ki jih vsebuje vsebnik *PralniStroj*. Ti vsebniki opisujejo metode REST, ki jih lahko izvajamo nad njimi. Operacijo pranja lahko zaženemo z zahtevkom *Create* na vsebniku *startStopContainer*, čigar URI je naveden. Enako velja za operacijo pridobivanja stanja stroja, kjer se izvrši zahtevek *Retrieve* na zadnjo vsebino instance vsebnika *stateContainer*.

```
1  <rdf:RDF>
2    <rdf:Description rdf:about="http://www.tno.com/saref#WASH_LG_123">
3      <rdf:type rdf:resource="http://www.tno.com/saref#WashingMachine"/>
4      <saref:hasManufacturer>LG</saref:hasManufacturer>
5      <saref:hasDescription>Very cool Washing Machine</saref:hasDescription>
6      <saref:hasLocation rdf:resource="http://www.tno.com/saref#Bathroom"/>
7      <msm:hasService rdf:resource="http://www.tno.com/saref#
          WashingService_123"/>
8      <msm:hasService rdf:resource="http://www.tno.com/saref#StateService_123
          "/>
9    </rdf:Description>
10
11   <rdf:Description rdf:about="http://www.tno.com/saref#WashingService_123">
12     <rdf:type rdf:resource="http://www.tno.com/saref#WashingService"/>
13     <msm:hasOperation rdf:resource="http://www.tno.com/saref#
          WashingOperation_123"/>
14   </rdf:Description>
15
16   <rdf:Description rdf:about="http://www.tno.com/saref#WashingOperation_123
        ">
17     <rdf:type rdf:resource="http://www.tno.com/saref#WashingOperation"/>
18     <hr:hasMethod>Create</hr:hasMethod>
19     <hr:hasURITemplate>/CSE1/WASH_LG_123/startStopContainer </hr:
          hasURITemplate>
20     <msm:hasInput rdf:resource="http://www.tno.com/saref#Action"/>
21   </rdf:Description>
22
23   <rdf:Description rdf:about="http://www.tno.com/saref#StateService123">
24     <rdf:type rdf:resource="http://www.tno.com/saref#StateService"/>
25     <msm:hasOperation rdf:resource="http://www.tno.com/saref#
          StateOperation123"/>
26   </rdf:Description>
27
28   <rdf:Description rdf:about="http://www.tno.com/saref#StateOperation123">
29     <rdf:type rdf:resource="http://www.tno.com/saref#StateOperation"/>
```

```
30      <hr:hasMethod>Retrieve</hr:hasMethod>
31      <hr:hasURITemplate>/CSE1/WASH_LG_123/state/stateContainer/latest</hr:
            hasURITemplate>
32      <msm:hasOutput rdf:resource="http://www.tno.com/saref#State"/>
33    </rdf:Description>
34  </rdf:RDF>
```

Izpis 6.1: Vsebina vira *semanticDescriptor*, ki pripada vsebniku *PralniStroj* (povzeto iz [16]).

Semantika v oneM2M je povzeta iz [16].

6.3 Semantika v OM2M

Prosto dostopna različica platforme OM2M trenutno ne podpira vira tipa *semanticDescriptor*. Podatkovne modele izbranih brezžičnih tehnologij smo prevedli z uporabo preusmeritvenega mehanizma, ki je opisan v 6.2.3. Za to se uporablja razred `Controller`, ki ga najdemo v vseh vtičnikih.

Poglavje 7

Sklepni del

V magistrskem delu smo preučili koncepta IoT in M2M, podrobno smo pregledali specifikacijo globalnega standarda oneM2M, na katerem je zgrajena platforma OM2M. Preučili smo specifikacijo semantike in ontologijo standarda oneM2M ter kako to aplicira platforma OM2M.

Pri pregledu področja nismo našli konkretnejših implementacij vtičnikov, ki podpirajo povezljivost z zunanjimi omrežji, ki niso zasnovani na protokolu IP, saj je novejša verzija OM2M izšla v juniju 2016.

Za razvoj vtičnikov smo izbrali pogosto uporabljene brezžične tehnologije in preučili njihove odprtokodne knjižnice. Platformo OM2M smo razširili s tremi vtičniki brezžičnih tehnologij Bluetooth, Bluetooh Low Energy in Z-Wave, ki omogočajo preiskovanje prostora in s tem odkrivanje naprav. Omogočajo povezovanje z zaznanimi napravami in branje podatkov, ki jih naprave ponujajo oz. pošiljajo. Vtičniki so zgrajeni po principih dobrih praks. Omogočajo izgradnjo podatkovne strukture glede na vrsto tehnologije in vnos prebranih podatkov v to strukturo.

Nadaljnje delo

Najpomembnejša stvar, ki bi jo bilo v prihodnosti dobro dodati k implementaciji vtičnikov, je apliciranje ontologije standarda oneM2M. Trenutna verzija

OM2M ne podpira semantike, kot je določeno v specifikaciji oneM2M, zato smo v delu opisali le način, kako prevesti ontologije zunanjih omrežij (sistemov) v ontologijo oneM2M Base Ontology. Ko bo platforma OM2M podpirala semantiko, bi vtičnikom spremenili način vnosa podatkov v sistem OM2M.

Implementirali smo vtičnike za tri popularne brezžične tehnologije oz. protokole. Naslednji korak za razvijalce bi bil implementacija vtičnikov za ostale tehnologije, tako žične kot brezžične.

Naša implementacija je polno podprta na operacijskih sistemih Linux. Za podprtje delovanja vtičnikov tudi na operacijskih sistemih Windows bi bilo v tem primeru potrebno spremeniti implementacijo za Bluetooth Low Energy, saj trenutno uporablja orodja, ki so del paketa Bluetooth na sistemih Linux *bluez*. Smiselna bi bila uporaba javanske knjižnice za BLE, če bo v prihodnosti na voljo.

V primeru implementacije za tehnologijo Z-Wave bi lahko dodali interakcijo z vozlišči, ki sama po sebi ne pošiljajo podatkov (aktuatorji in ne senzor). Smiselno bi bilo, da je interakcija z vozliščem omogočena na način, ki ga vozlišče podpira.

Literatura

[1] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira. OM2M: Extensible ETSI-compliant M2M service platform with self-configuration capability. *Procedia Computer Science*, 32:1079 – 1086, 2014.

[2] Bluecove. Bluecove. `http://bluecove.org`. Dostopano 6.6.2016.

[3] A. Aguiar C. Pereira. Towards efficient mobile M2M communications: Survey and open challenges. *Sensors*, 14(10):19582–19608, 2014.

[4] Amaxilatis Dimitrios, Georgitzikis Vasileios, Giannakopoulos Dimitrios, and Chatzigiannakis Ioannis. Employing internet of things technologies for building automation. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pages 1–8. IEEE, 2012.

[5] Omar Elloumi. oneM2M – Presentation To W3C. `https://www.w3.org/WoT/IG/wiki/images/6/66/OneM2M-for-W3C_jan_26th_2016_OELLOUMI_MBAUER.pdf`, January 2016. Pridobljeno 6.6.2016.

[6] Asma Elmangoush, Hakan Coskun, Sebastian Wahle, and Thomas Magedanz. Design aspects for a reference M2M communication platform for Smart Cities. In *Innovations in Information Technology (IIT), 2013 9th International Conference on*, pages 204–209. IEEE, 2013.

[7] Bob Emerson. M2M: The Internet of 50 Billion Devices. *WinWin Magazine*, pages 19–22, Jan. 2010.

[8] Sony Ericsson. *Developing Applications with the Java APIs for Bluetooth^{TM} (JSR-82)*, January 2004.

[9] The Eclipse Foundation. Eclipse OM2M. http://www.eclipse.org/om2m/. Dostopano 6.6.2016.

[10] Mikhail T. Galeev. Catching the Z-Wave. http://www.embedded.com/design/connectivity/4025721/Catching-the-Z-Wave, October 2006. Pridobljeno 6.6.2016.

[11] Gartner. Press release: Gartner says 6.4 billion connected "Things" will be in use in 2016, up 30 percent from 2015. http://www.gartner.com/newsroom/id/3165317. Dostopano 6.6.2016.

[12] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199 – 220, 1993.

[13] Java Community Process. *Java^{TM} APIs for Bluetooth^{TM} Wireless Technology (JSR-82)*, July 2008. Specification Version 1.1.1.

[14] Eui-Jik Kim and Sungkwan Youm. Machine-to-machine platform architecture for horizontal service integration. *EURASIP Journal on Wireless Communications and Networking*, 2013(1):1, 2013.

[15] OASIS. oBIX Version 1.1, Committee Specification Draft 01 / Public Review Draft 01. http://docs.oasis-open.org/obix/obix/v1.1/csprd01/obix-v1.1-csprd01.html. Dostopano 6.9.2016.

[16] oneM2M Partners. *oneM2M Technical Specification: Base Ontology*. oneM2M, 0.10.0 edition, June 2016.

[17] oneM2M Partners. *oneM2M Technical Specification: Functional Architecture*. oneM2M, 2.9.1 edition, July 2016.

[18] Oracle. Java Platform, Micro Edition (Java ME). http://www.oracle.com/technetwork/java/embedded/javame/overview/index.html. Dostopano 6.6.2016.

[19] Bodhi Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao. Tiny web services for sensor device interoperability. In *Information Processing in Sensor Networks, 2008. IPSN'08. International Conference on*, pages 567–568. IEEE, 2008.

[20] Java Community Process. JSR 82: Java^{TM} APIs for Bluetooth. `https://jcp.org/en/jsr/detail?id=82`. Dostopano 6.6.2016.

[21] Bluetooth SIG. Adopted specifications. `https://www.bluetooth.com/specifications/adopted-specifications`. Dostopano 6.6.2016.

[22] Bluetooth SIG. GATT Specifications. `https://www.bluetooth.com/specifications/generic-attributes-overview`. Dostopano 6.6.2016.

[23] Bluetooth SIG. Profiles overview. `https://www.bluetooth.com/specifications/profiles-overview`. Dostopano 6.6.2016.

[24] JaeSeung Song, Andreas Kunz, Mischa Schmidt, and Piotr Szczytowski. Connecting and Managing M2M Devices in the Future Internet, 2013.

[25] Jorg Swetina, Guang Lu, Philip Jacobs, Francois Ennesser, and Jaeseung Song. Toward a standardized common M2M service layer platform: Introduction to oneM2M. *IEEE Wireless Communications*, 21(3):20–26, 2014.

[26] K. Topley. *J2ME in a Nutshell: A Desktop Quick Reference.* In a Nutshell (o'Reilly) Series. O'Reilly, 2002.

[27] K. Townsend, R. Davidson, and C. Cufí. *Getting Started with Bluetooth Low Energy.* O'Reilly, 2014.

[28] Wikipedia. Bluetooth. `https://en.wikipedia.org/wiki/Bluetooth`. Dostopano 6.6.2016.

[29] Wikipedia. Z-Wave.
`http://en.wikipedia.org/wiki/Z-Wave`. Dostopano 6.6.2016.

[30] Sheng-Kai Zhang, Jian-Wei Zhang, and Wen Li. Design of M2M Platform Based on J2EE and SOA. In *E-Business and E-Government (ICEE), 2010 International Conference on*, pages 2029–2032. IEEE, 2010.

Apendiks

Dodatek A

Specifikacija oneM2M

A.1 Referenčne točke

Referenčna točka (ang. *reference point*) je sestavljena iz enega ali več kakršnih-koli vmesnikov med dvema ponudnikoma storitev. Naslednje referenčne točke so podprte s strani CSE (uporablja se nomenklatura Mc(-), ki izhaja iz besedne zveze *M2M communications*):

- *Mca* je referenčna točka, čez katero poteka komunikacija med AE in CSE. To omogoča AE uporabo storitev, ki jih ponuja CSE in omogoča CSE, da komunicira z AE (AE in CSE lahko nastopata v isti entiteti ali pa tudi ne).

- *Mcc* je referenčna točka, čez katero poteka komunikacija med dvema CSE. To omogoča CSE uporabo storitev druge CSE.

- *Mcn* je referenčna točka, čez katero poteka komunikacija med CSE in NSE. To omogoča CSE uporabo storitev NSE (poleg storitev prenosa in povezovanja).

- *Mcc'* je referenčna točka, čez katero poteka komunikacija med dvema CSE v infrastrukturnih vozliščih (glej A.2), ki so skladna z oneM2M in se nahajajo v različnih domenah ponudnika storitev M2M. Ta povezava

omogoča CSE v vozlišču domenske infrastrukture ponudnika storitev M2M, da komunicira s CSE v drugem vozlišču domenske infrastrukture drugega ponudnika storitev M2M in uporabi njegove storitve (in obratno). Mcc' razširja dosegljivost storitev preko referenčne točke Mcc ali njenega podomrežja.

A.2 Vozlišča

Vozlišča (ang. *nodes*) so logične entitete, ki so posamezno določljive v sistemu oneM2M. So logični ekvivalent fizične (ali virtualne) naprave [5]. Lahko so CSE-Capable (vsebujejo CSE) ali Non-CSE-Capable (ne vsebujejo CSE).

Vozlišče, ki je CSE-Capable, je logična entiteta, ki vsebuje vsaj eno oneM2M CSE in vsebuje nič ali več AE. Primeri vozišč CSE-Capable so ASN, IN in MN.

Vozlišče, ki je Non-CSE-Capable, je logična entiteta, ki ne vsebuje CSE in vsebuje nič ali več AE. Primera vozlišč Non-CSE-Capable sta ADN in Non-oneM2M vozlišče.

CSE, ki se nahajajo v različnih vozliščih, so lahko različne. Odvisne so od storitev, ki jih podpirajo, in od značilnosti (npr. različna količina spomina, druga strojna programska oprema) fizične entitete, ki jih vsebuje.

Arhitektura oneM2M podpira naslednje tipe vozlišč:

- *Application Service Node* – ASN je vozlišče, ki vsebuje eno CSE in vsaj eno AE. V domeni področja (glej A.3) se lahko nahaja nič ali več vozlišč ASN. CSE v ASN komunicira z drugo CSE preko referenčne točke Mcc, ki se nahaja v MN ali v IN. AE v ASN komunicira s CSE v istem ASN preko Mca. ASN komunicira z (več) NSE preko Mcn. Primer fizične preslikave: ASN bi se lahko nahajal znotraj naprave M2M.

- *Application Dedicated Node* – ADN je vozlišče, ki vsebuje vsaj eno AE in ne vsebuje CSE. V domeni področja (glej A.3) se lahko pojavi nič ali več vozlišč ADN. AE v ADN komunicira s CSE preko Mca, ki se nahaja

v MN ali v IN. Primer fizične preslikave: ADN se lahko nahaja znotraj omejene naprave M2M.

- *Middle Node* – MN je vozlišče, ki vsebuje eno CSE in nič ali več AE. V domeni področja se lahko pojavi nič ali več vozlišč MN. CSE v MN komunicira z eno CSE v MN ali IN preko Mcc in z eno ali več CSE v MN ali ASN. CSE v MN lahko komunicira tudi z več AE, ki so v istem MN, ali v ADN preko Mca. CSE v MN komunicira z (več) NSE preko Mcn . Primer fizične preslikave: MN se lahko nahaja v usmerjevalniku M2M (ang. *M2M Gateway*).

- *Infrastructure Node* – IN je vozlišče, ki vsebuje eno CSE in nič ali več AE. IN se pojavi samo enkrat v infrastrukturni domeni (glej A.3) na enega ponudnika storitev M2M. CSE v IN lahko vsebuje funkcije CSE, ki niso skladne z ostalimi tipi vozlišč. CSE v IN komunicira z eno ali več CSE preko Mcc, ki so v (enem ali več) MN in/ali v (enem ali več) ASN. CSE v IN komuicira z enim ali več AE v istem IN ali ADN preko Mca. CSE v IN komunicira z (več) NSE preko Mcn in s (več) CSE preko Mcc', ki se nahajajo v IN drugih ponudnikov storitev M2M. Primer fizične preslikave: IN se lahko nahaja v infrastrukturi M2M storitev.

- *Non-oneM2M* – NoDN je vozlišče, ki ne vsebuje entitet oneM2M (niti AE ali CSE). S temi vozlišči ponazorimo naprave, ki so priključene v sistem oneM2M, z namenom prikaza vzajemnega delovanja, vključno z upravljanjem.

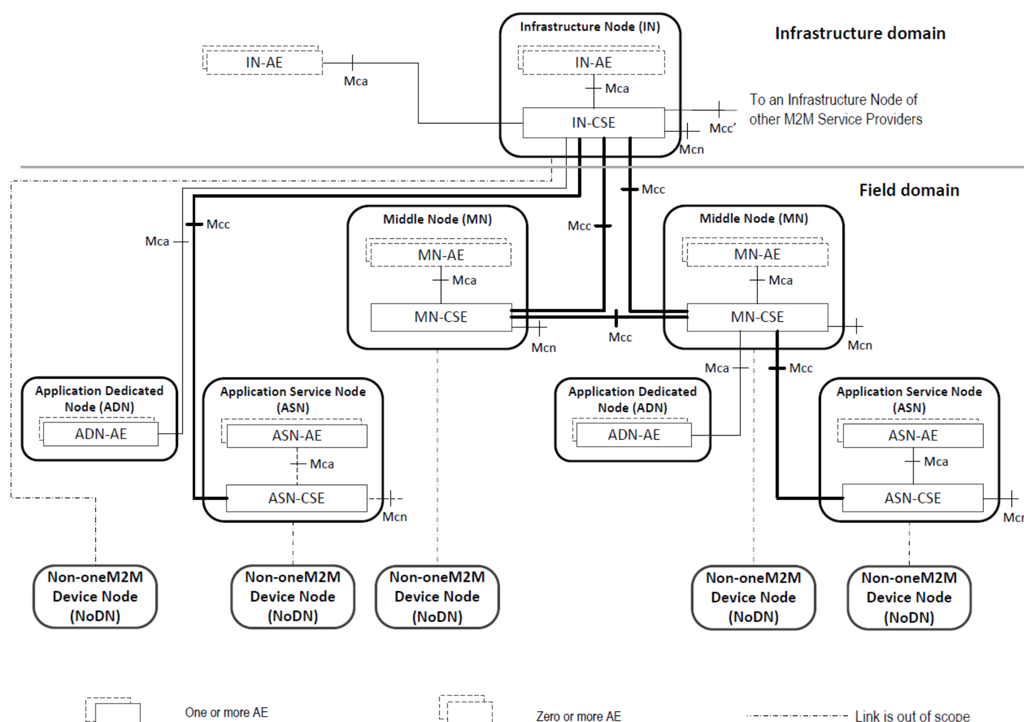
A.3 Domene

V oneM2M sta definirana dva tipa domen:

- I*nfrastructure Domain* oz. *infrastrukturna domena* kateregakoli ponudnika storitev M2M vsebuje točno eno infrastrukturno vozlišče in predstavlja zunanje omrežje.

- *Field Domain* oz. *domena področja* kateregakoli ponudnika storitev M2M lahko vsebuje vozlišča ASN, ADN, MN ali NoDN.

Možne konfiguracije povezovanja različnih entitet, ki so podprte v sistemu oneM2M, so prikazane na sliki A.1. Slika ne omejuje množičnosti entitet in ne zahteva, da so vsa razmerja prisotna.



Slika A.1: Podprte konfiguracije v oneM2M (pridobljeno iz [17]).

A.4 Identifikatorji M2M

Identifikator M2M je zaporedje znakov in se uporablja za sklicevanje na entitete (npr. CSE ali AE), vire (glej A.6) ali objekte (npr. ponudnik storitev M2M ali vozlišča M2M), ki so določeni v oneM2M. Identifikator M2M ima konsistenten pomen - konsistentno se nanaša na isti vir, entiteto ali objekt v času, ko obstajajo v nekem kontekstu.

Določeni so naslednji identifikatorji:

- *M2M Service Provider Identifier* (M2M-SP-ID): ponudnik storitev je enolično določen s tem identifikatorjem. Gre za statično vrednost, ki se pripiše ponudniku storitev.

- *Application Entity Identifier* (AE-ID) enolično določa AE v vozlišču M2M ali AE, ki zahteva komunikacijo z vozliščem M2M. AE-ID identificira AE za vse vrste interakcij znotraj sistema M2M.

- *Application Identifier* (App-ID) enolično določa aplikacijo M2M znotraj danega konteksta. Obstajata dva tipa App-ID: registriran App-ID in neregistriran App-ID. Registriran je garantirano globalno unikaten, medtem ko neregistriran ni.

- *CSE Identifier* (CSE-ID) je globalno unikaten in enolično določa CSE po tem, ko se CSE instancira znotraj vozlišča M2M v sistemu M2M. CSE-ID je globalno unikaten, ampak ni potrebno da je, če ga uporabljamo znotraj specifične domene ponudnika storitev M2M.

- *M2M Node Identifier* (M2M-Node-ID) je globalno unikaten in enolično določa vozlišče M2M, ki gosti CSE in/ali AE. Sistem M2M dovoli ponudniku storitev M2M, da nastavi CSE-ID in M2M-Node-ID na enako vrednost. M2M-Node-ID omogoča ponudniku storitev M2M, da poveže CSE-ID z določenim vozliščem M2M.

- *M2M Service Subscription Identifier* (M2M-Sub-ID) omogoča ponudniku storitev M2M, da poveže aplikacije, vozlišča M2M, CSE in storitve z določeno naročniško storitvijo M2M, med naročnikom M2M in ponudnikom storitev M2M. M2M-Sub-ID je unikaten za vsakega naročnika M2M.

- *M2M Request Identifier* (M2M-Request-ID) sledi zahtevku, ki ga je sprožil AE preko Mca in preko Mcc, če ga je sprožil CSE. Vključuje se tudi v odgovor na zahtevek preko Mca ali Mcc.

- *M2M External Identifier* (M2M-Ext-ID) je uporabljen s strani ponudnika storitev M2M, ko so storitve, usmerjene na CSE (identificirano s CSE-ID), zahtevane iz omrežja.
- *Underlying Network Identifier* (UNetwork-ID) se uporablja za identifikacijo omrežja. UNetwork-ID je statična vrednost in unikatna znotraj ponudnika storitev M2M.
- *Trigger Recipient Identifier* (Trigger-Recipient-ID) se uporablja za identifikacijo instance ASN/MN-CSE, ko se zahtevajo storitve na napravi iz omrežja.
- *M2M Service Profile Identifier* (M2M-Service-Profile-ID): definira pravila, ki veljajo za AE, katere se registrirajo na vozliščih M2M. Vsakemu profilu *M2M Service Profile* se določi ta identifikator, da ga lahko pridobimo za preverjanje pristnosti.

A.5 Formati identifikatorjev

V splošnem velja, da so identifikatorji AE, CSE in ostalih virov globalno unikatni. Zaradi optimizacije se v primeru, če lahko iz njihovega konteksta pridobimo področje uporabe, identifikator skrajša. Skrajšani identifikatorji so definirani kot relativni formati identifikatorjev. Sistem M2M bo uporabil identifikatorje M2M-SP-ID, CSE-ID, App-ID, AE-ID in identifikatorje virov, glede na formate in pravila, določena v naslednji strukturi. Na prvem nivoju je podano ime identifikatorja, na drugem vrsta identifikatorja (relativni ali absolutni), oznaka formata in, če gre za relativni tip, še kontekst. Na tretjem nivoju sta podana format in pravila uporabe.

- M2M-SP-ID:
  - absolutni; oznaka: M2M-SP-ID
    - M2M-SP-ID je določen v skladu s formatom FQDN definiranem v IETF RFC 1035 [i.7]. Uporablja se prefiks '//'.

Struktura: //{FQDN}
FQDN (*Fully Qualified Domain Name*) predstavlja polno ime kvalificirane domene ponudnika storitev M2M.
Primera:

· //www.m2mprovider.com
· //globalm2m.org

Sledeča identifikatorja M2M-SP-ID bi se lahko uporabila za ločitev dveh različnih segmentov storitev:

· //automotive.m2m.telematics-servicecompany.com
· //building-management.m2m.telematicsservice-company.-com

∗ Kadarkoli se uporabi M2M-SP-ID, velja le tukaj definirani absolutni format

- CSE-ID:
  - relativni; oznaka: SP-relative-CSE-ID, kontekst: domena ponudnika storitev M2M, ki gostuje CSE
    ∗ SP-relative-CSE-ID se začne s poševnico ("/"), ki ji sledijo nerezervirani znaki, definirani v določbi 2.3 dokumenta IETF RFC 3986 [i.10].
    SP-relative-CSE-ID je unikaten znotraj konteksta domene ponudnika storitev M2M, ki gostuje CSE.
    Ponudnik storitev M2M predpisuje SP-Relative-CSE-ID in garantira, da je unikaten znotraj konteksta domene.
    Primeri:
      · /123A38ZZY
      · /CSE090112
      · /3ace4fd3
    ∗ na referenčnih točkah Mca in Mcc: nanašanje na CSE, ki jih gosti domena istega ponudnika storitev M2M.

- absolutni; oznaka: Absolute-CSE-ID
  * konkatenacija po formatu {M2M-SP-ID}{SP-relative-CSE-ID}, kjer se identifikatorja ustrezno nadomestita.
    Absolute-CSE-ID je skladen s določbo 3 v RFC3986 [10].
    Primera:
    · //www.m2mprovider.com/C3219
    · //m2m.thingscompany.com/ab3f124a
  * na referenčnih točkah Mca in Mcc: nanašanje na CSE, ki jih gostijo domene različnih ponudnikov storitev M2M.

- AE-ID:
  - relativni; oznaka: AE-ID-Stem; kontekst: registrirni CSE, kjer se je registrirala AE ali domena ponudnika storitev M2M, ki gostuje AE.
    * AE-ID-Stem je zaporedje znakov, ki lahko vključujejo kateregakoli izmed rezerviranih znakov, definiranih v določbi 2.3 dokumenta IETF RFC 3986 [i.10]. Prvi znak AE-ID-Stem ima določen pomen in njegova vrednost je opredeljena na naslednji način:
      1. Prvi znak je 'C': AE-ID-Stem je bil določen s strani registrskega CSE od AE. V tem primeru je AE-ID-Stem unikaten znotraj konteksta registrskega CSE. Gostujoč CSE mora garantirati, da je AE-ID-Stem unikaten znotraj konteksta gostujočega CSE.
         Primera:
         · C190XX7T
         · Ca3e3f3ab
      2. Prvi znak je 'S': AE-ID-Stem je bil določen s strani registrskega ponudnika storitev M2M. V tem primeru je AE-ID-Stem unikaten znotraj konteksta domene ponudnika

storitev M2M. Ponudnik storitev M2M mora garantirati, da je AE-ID-Stem unikaten znotraj konteksta domene ponudnika storitev M2M.
Primera:
· S190XX7T
· Sa3e3f3ab

Uporaba drugih znakov na prvem mestu AE-ID-Stem je rezervirana. Katera od naštetih dveh metod se bo uporabila, je odvisno od postopka registracije AE.

* na referenčni točki Mca: nanašanje na AE, ki so se registrirali na CSE, kjer je registriran izvir zahtevka - originator[1]

– relativni; oznaka: SP-relative-AE-ID, kontekst: domena ponudnika storitev M2M, ki gosti AE.

  * 1. V primeru, da se AE-ID-Stem začne s črko 'C', je SP-relative-AE-ID sestavljen po formatu {SP-relative-CSE-ID}/{AE-ID-Stem}, kjer se identifikatorja ustrezno zamenjata.
    Primera:
    · /CSE090112/C190XX7T
    · /3ace4fd3/Ca3e3f3ab
    2. V primeru, da se AE-ID-Stem začne s črko 'S', je AE-ID-Stem unikaten znotraj domene ponudnika storitev M2M. V tem primeru je SP-relative-AE-ID sestavljen po formatu /{AE-ID-Stem}, kjer se identifikator ustrezno zamenja.
    Primera:
    · /S190XX7T
    · /Sa3e3f3ab

  SP-relative-AE-ID se začne s poševnico ('/') in je skladen z določbo 4.2 v dokumentu IETF RFC 3986 [i.10].

[1] Originator je AE ali CSE, ki prvi pošlje zahtevek.

        * na referenčnih točkah Mca in Mcc: nanašanje na AE, registrirane s CSE, ki niso originator zahtevka ampak jih gosti domena ponudnika storitev M2M, na katerega je originator priklopljen.

    - absolutni; oznaka: Absolute-AE-ID
        * Format Absolute-AE-ID identifikatorja AE-ID je sestavljen po formatu {M2M-SP-ID}{SP-relative-AE-ID}, kjer ustrezno zamenjamo identifikatorja. Absolute-AE-ID je skladen z določbo 3 dokumenta IETF RFC 3986 [i.10].
          Primeri:
            · //m2m.prov.com/CSE3219/C9886
            · //m2m.things.com/ab3f124a/Ca2efb3f4
            · //m2m.things.com/S98821
        * na referenčnih točkah Mca in Mcc: sklicevanje na AE, ki jih gosti različna domena ponudnika storitev M2M, kot tista na katero je originator zahtevka priklopljen; na referenčni točki Mcc' za vse AE.

- Resource identifier:
    - relativni; oznaka: Unstructured-CSE-relative-Resource-ID; kontekst: CSE, ki gosti vir
        * Unstructured-CSE-relative-Resource-ID je zaporedje znakov, ki lahko vsebuje znake definireane v določbi 2.3 dokumenta IETF RFC 3986 [i.10]. Identifikator vira, ki je relativen na CSE, je unikaten znotraj konteksta gostujočega CSE. CSE, ki gostuje vir, je zadolžen za garantiranje unikatnosti identifikatorja znotraj konteksta gostujočega CSE.
          Primeri:
            · container123
            · a1b2c3d4b0b00f0fa66a123456789abc

        · xxyz1234

    * na referenčni točki Mca: sklicevanje na vire, ki jih gosti CSE, kateri dobi zahtevek na vir.

- relativni; oznaka: Structured-CSE-relative-Resource-ID; kontekst: CSE, ki gosti vir

    * Structured-CSE-relative-Resource-ID je zaporedje znakov, ki lahko vsebuje nerezervirane znake, določene v določbi 2.3 dokumenta IETF RFC 3986 [i.10], kot tudi poševnico ('/') (se ne začne s poševnico). Structured-CSE-relative Resource-ID je unikaten v kontekstu CSE, ki gosti vir. Struktura predstavlja razmerje starš-otrok, kjer so deli imena virov staršev ločena od imena virov otrok s poševnico. Prvi del imena vira je vir CSEBase. CSE, ki gosti vir, garantira unikatnost identifikatorja v kontekstu gostujočega CSE.
    Primera:

        · CSEBase_Name/Container_Name/CI_N

        · CSEBase_Name/AE_Name

    * na referenčni točki Mca: sklicevanje na vire, ki jih gosti CSE, kateri dobi zahtevek na vir.

- relativni; oznaka:SP-relative-Resource-ID; kontekst: CSE, ki gosti vir

    * Sestavi se po formatu {SP-relative-CSE-ID}/{Unstructured-CSE-relative-Resource-ID}{SP-relative-CSE-ID}/{Structured-CSE-relative-Resource-ID}, kjer se identifikatorji ustrezno zamenjajo. SP-relative-Resource-ID se začne s poševnico in je skladen z določbo 4.2 v dokumentu IETF RFC 3986 [i.10]. SP-relative-Resource-ID je unikaten v kontekstu ponudnika storitev.
    Primera:

        · CSE_ID/CI_ResId

      · CSE_ID/CSEBase_Name/AE_Name/Cont_Name
    * na referenčnih točkah Mca in Mcc: sklicevanje na vire, ki jih gosti ista domena ponudnika storitev M2M, kot domena, ki gosti CSE, katera prejme zahtevek za dostop do vira.
  – absolutni; oznaka: Absolute Resource ID
    * Sestavi se po formatu {M2M-SP-ID}{SP-relative Resource ID}, kjer se identifikatorja ustrezno zamenjata. Absolute-CSE-ID je skladen z določbo 3 v dokumentu IETF RFC 3986 [i.10]. Primera:
      · //www.m2mprovider.com/CSE_ID/CI_ResID
      · //www.m2mprovider.com/CSE_ID/CSEBase_Name/Cont_Name/CI_Name
    * na referenčnih točkah Mca in Mcc: sklicevanje na vire, ki jih gosti druga domena ponudnika storitev M2M, kot domena, ki gosti CSE, katera prejme zahtevek za dostop do vira; na referenčni točki Mcc' za vse vire.

- APP-ID:
  – App-ID
    * Format "R[authority-ID]/[registered-App-ID]" ali "N[non-registered-App-ID]". Če je prva črka 'R', authority-ID in registered-App-ID določi registrirni organ. registered-App-ID je upravljan s strani lastnika authority-ID. Če je prva črka 'N', potem non-registered-App-ID ni registriran s strani registrirnega organa.
    * Uporaba je skladna s postopkom registracije AE.

A.6 Viri

Vse entitete v sistemu oneM2M (npr. AE, CSE ...) so predstavljene kot viri. Vsak vir ima določeno strukturo, s katero je predstavljen. Ti viri so enolično naslovljivi. Prav tako so določeni postopki za dostop do virov.

Poznamo tri kategorije virov:

- Običajni viri: vključujejo celoten nabor reprezentacij podatkov, ki predstavljajo bazo informacij za upravljanje.
- Virtualni viri oz. atributi: uporabljajo se za proženje procesov in/ali pridobivanje rezultatov, ampak nimajo stalne reprezentacije v CSE.
- Objavljeni viri: objavljen vir je vir na oddaljenem CSE, ki je povezan s prvotnim virom (ki je bil objavljen), in ima nekatere značilnosti prvotnega vira.

A.7 Tipi virov

Spodaj so našteti običajni in virtualni tipi virov. Naveden je kratek opis tipa vira in tipi virov, ki se lahko pojavijo kot otrok oz. starš v strukturi:

- accessControlPolicy:
  - Opis: vsebuje predstavitev privilegijev. Je povezan z vsemi viri, ki so dostopni zunanjim entitetam - zunaj gostujočega CSE. Nadzoruje kdo ima privilegije za izvedbo nečesa v nekem kontekstu, ko dostopa do vira.
  - Otroci: subscription.
  - Starši: AE, AEAnnc, remoteCSE, remoteCSEAnnc, CSEBase.
- AE:
  - Opis: vsebuje informacije o AE. Ustvari se po uspešni registraciji AE na registrskem CSE.
  - Otroci: subscription, container, group, accessControlPolicy, pollingChannel, schedule.
  - Starši: remoteCSE, remoteCSEAnnc, CSEBase.

- container (*vsebnik*):
  - Opis: razpolaga s podatki med entitetami. Uporablja se kot mediator, ki medpomni izmenjane podatke med AE in/ali CSE. Izmenjava podatkov med AE (npr. AE v vozlišču v domeni področja in istoležni AE v infrastrukturni domeni) ne zahteva direktne povezave in dovoljuje scenarije, kjer sta obe entiteti različno časovno dosegljivi.
  - Otroci: container, contentInstance, subscription, latest, oldest.
  - Starši: AE, AEAnnc, container, containerAnnc, remoteCSE, remoteCSEAnnc, CSEBase.
- contentInstance (*primer vsebine* oz. *instanca vsebine*):
  - Opis: Predstavlja podatek v viru container.
  - Otroci: brez.
  - Starši: container, containerAnnc.
- CSEBase
  - Opis: strukturni koren vseh virov, ki se nahajajo v CSE. Vsebuje informacije o CSE.
  - Otroci: remoteCSE, remoteCSEAnnc, node, AE, container, group, accessControlPolicy, subscription, mgmtCmd, locationPolicy, statsConfig, statsCollect, request, delivery, schedule.
  - Starši: brez.
- latest (V):
  - Opis: virtualni vir, ki kaže na zadnjo ustvarjeno instanco vsebine (*contentInstance*), znotraj vira *container*.
  - Otroci: brez.
  - Starši: container.

- mgmtCmd:
  - Opis: vir *Management Command* predstavlja metodo za izvrševanje procedur za upravljanje, ki jih zahtevajo obstoječi upravljavski protokoli.
  - Otroci: execInstance, subscription.
  - Starši: CSEBase.
- mgmtObj:
  - Opis: vir *Management Object* predstavlja upravljavske funkcije, ki zagotavljajo abstrakcijo za preslikavo v zunanje tehnologije za upravljanje. Predstavlja vozlišče in programsko opremo, ki je nameščena na njem.
  - Otroci: subscription, mgmtObj, schedule.
  - Starši: node, mgmtObj, mgmtObjAnnc.
- node:
  - Opis: predstavlja informacije določenega vozlišča.
  - Otroci: mgmtObj, subscription.
  - Starši: CSEBase, remoteCSE.
- oldest (V):
  - Opis: enako kot *latest*, le da ta virtualni vir hrani najstarejšo ustvarjeno instanco.
  - Otroci: brez.
  - Starši: container.
- remoteCSE:
  - Opis: predstavlja oddaljeni CSE, za katerega je bila identificirana registracija z registrarnim CSE, s strani vira *CSEBase*.

  - Otroci: AE, container, group, accessControlPolicy, subscription, pollingChannel, schedule, node.
  - Starši: CSEBase

- request:
  - Opis: izraža kontekst dostopa izdanega zahtevka.
  - Otroci: subscription.
  - Starši: CSEBase.

- semanticDescriptor:
  - Opis: uporablja se za shranjevanje semantičnega opisa, ki se nanaša na vir (in podvire). Semantične informacije uporabi semantična funkcionalnost sistema oneM2M in so na voljo aplikacijam ali CSE.
  - Otroci: subscription.
  - Starši: AE, container, contentInstance, group, node, flexContainer, timeSeries.

- subscription:
  - Opis: vir *subscription* predstavlja naročniške informacije vira. Tak vir je otrok vira, na katerega smo naročeni.
  - Otroci: schedule.
  - Starši: accessControlPolicy, accessControlPolicyAnnc, AE, AEAnnc, container, CSEBase, delivery, eventConfig, execInstance, group, groupAcce, locationPolicy, mgmtCmd, mgmtObj, mgmtObjAnnc, m2mServiceSubscriptionProfile, node, nodeAnnc, serviceSubscribedNode, remoteCSE, remoteCSEAnnc, request, schedule, statsCollect, statsConfig.

A.7.1 Univerzalni atributi

Univerzalni atributi so atributi, ki so prisotni v vseh tipih virov. Spodaj so našteti univerzalni atributi vseh tipov virov, ki so običajni (niso virtualni ali objavljeni):

- *resourceType*: določen je ob kreiranju in je samo za branje. Določa tip vira. Vsak vir ima atribut *resourceType*.
- *resourceID*: je identifikator vira, ki se uporablja v nehierarhični metodi naslavljanja (glej A.8). Vsebuje identifikator vira v formatu Unstructured-CSE-relative-Resource-ID (glej odstavek *Formati identifikatorjev* pod A.4). Atribut se določi s strani gostujočega CSE, ko se izvede procedura ustvarjanja vira. Gostujoči CSE pripiše unikaten *resourceID* v tem CSE.
- *resourceName*: predstavlja ime vira, ki se uporablja v hierarhični metodi naslavljanja (glej A.8) in ponazarja razmerja virov starš-otrok. Atribut poda tisti, ki je vir ustvaril. Gostujoči CSE uporabi predlagan *resourceName*, če ta še ne obstaja med otroki ciljnega starševskega vira. Če ta obstaja, gostujoči CSE zahtevek zavrne in vrne napako izvoru zahtevka. Gostujoči CSE dodeli vrednost atributa *resourceName*, če ga ne dodeli tisti, ki je vir ustvaril.
- *parentID*: je *resourceID* starša tega atributa. *ParentID* je prisoten v vseh tipih virov razen v *CSEBase*.
- *creationTime* oz. čas kreiranja vira je atribut, ki je obvezen za vse vire. Vrednost atributa določi sistem, ko se vir lokalno ustvari. Vrednost tega atributa se ne spreminja.
- *lastModifiedTime*: čas zadnje posodobitve vsebine vira je atribut, ki je obvezen. Vrednost atributa določi gostujoči CSE, ko se vir ustvari, in vrednost se posodobi, ko se vir posodobi oz. se ustvari/izbriše eden od otrokov vira.

- *expirationTime*: določa čas, po katerem gostujoča CSE izbriše vir. Atribut lahko določi izvor zahtevka in v tem primeru je čas obstoja vira informacija gostujočemu CSE. Ne glede na to, se gostujoči CSE odloči o pravi vrednosti atributa *expirationTime*. Če se odloči o zamenjavi vrednosti atributa, se o tem obvesti izvor zahtevka.

A.8 Naslavljanje v oneM2M

A.8.1 Naslavljanje virov

Za dostop do vira se uporablja naslov, ki je sestavljen iz niza znakov in enolično identificira vir znotraj področja zahtevka. Področje zahtevka je lahko:

- Relativno na CSE: zahtevek naslavlja vir, ki se nahaja v istem CSE kot prejemnik zahtevka CSE. V tem primeru se za naslavljanje vira lahko uporabi format identifikatorja vira, ki je relativen na CSE.
- Relativno na SP (*Service Provider*): zahtevek naslavlja vir, ki se nahaja v CSE znotraj iste domene ponudnika storitev M2M kot izvor zahteve. V tem primeru se za naslavljanje vira lahko uporabi format identifikatorja vira, ki je relativen na ponudnika storitev (SP).
- Absolutno: zahtevek naslavlja vir, ki se nahaja v CSE znotraj druge domene ponudnika storitev M2M, kot je domena izvora zahtevka. V tem primeru se za naslavljanje vira uporabi absolutni format. Ta format se lahko uporabi tudi v ostalih primerih.

Vsakemu viru se lahko na več načinov ustvari identifikator, preko katerega se lahko pridobi dostop do vira. Obstajata dve metodi za naslavljanje virov znotraj oneM2M in trije načini za vsako metodo, glede na področje zahtevka za dostop do vira (formati so opisani v A.5):

- Nehierarhična metoda:

  - relativno na CSE: uporabi se format Unstructured-CSE-relative-Resource-ID;
  - relativno na SP: uporabi se format SP-relative Resource-ID, skupaj s formatom Unstructured-CSE-relative-Resource-ID;
  - absolutno: uporabi se format Absolute-Resource-ID, skupaj z Unstructured-CSE-relative-Resource-ID.

- Hierarhična metoda:

  - relativno na CSE: uporabi se format Structured-CSErelative-Resource-ID;
  - relativno na SP: uporabi se format SP-relative Resource-ID, skupaj s Structured-CSErelative-Resource-ID;
  - absolutno: uporabi se format Absolute-Resource-ID, skupaj s Structured-CSE-relative-Resource-ID.

A.8.2 Naslavljanje aplikacijskih entitet

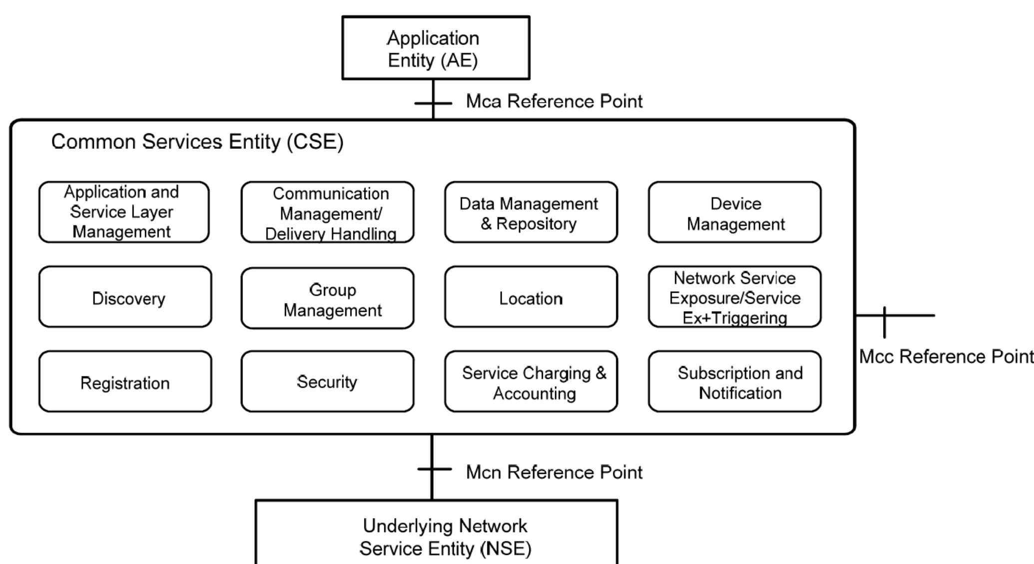
Cilj naslavljanja v M2M je doseganje CSE, na katerem je registrirana ciljna AE, in doseganje vozlišča M2M, na katerem se nahaja ciljna AE. Ta princip velja za vse AE.

Dosegljivost in usmerjanje iz/v AE v vozliščih M2M je povezano s CSE, na katerih so AE registrirane, in s povezljivostjo CSE z osnovnim omrežjem. Doseganje AE se izvaja z doseganjem CSE, na katerem je AE registrirana. Za doseganje CSE iz osnovnega omrežja se uporabi CSE-PoA (*CSE Point of Access*), ki vsebuje vse potrebne informacije. CSE-PoA običajno vsebuje informacije, ki se prevedejo v spletni naslov.

CSE-PoA se uporablja v sistemu M2M za komuniciranje s CSE v vozlišču M2M. Ko se povezava s CSE vzpostavi, se do AE lahko dostopa, dokler je AE enolično določena. Informacije, ki se nahajajo v CSE-PoA, so odvisne od značilnosti osnovnega omrežja in od prenosnih zmogljivosti vozlišča M2M.

A.9 Funkcije skupnih storitev (CSF)

Storitve, ki se nahajajo znotraj CSE, imenujemo *funkcije skupnih storitev* (ang. *Common Services Functions*). CSF nudijo storitve aplikacijskim entitetam preko referenčne točke Mca in ostalim CSE preko referenčne točke Mcc, kot je prikazano na sliki A.2. Komunikacija CSE - NSE poteka preko referenčne točke Mcn. Funkcije znotraj CSE lahko komunicirajo med sabo.



Slika A.2: Funkcije skupnih storitev (pridobljeno iz [17]).

Med funkcije skupnih storitev spadajo:

- *Application and Service Layer Management* - upravljanje z aplikacijskim in storitvenim nivojem zagotavlja upravljanje AE in CSE v ADN, ASN, MN in IN. To omogoča nastavljanje, odpravljanje napak in nadgradnjo funkcij CSE in nadgradnjo AE.

- *Communication Management and Delivery Handling* - upravljanje komunikacij in upravljanje z dostavami omogočata komunikacijo z ostalimi CSE, AE in NSE.

- *Data Management and Repository*: funkcija upravljanja s podatki in repozitoriji zagotavlja shranjevanje podatkov in mediacijske funkcije.
- *Device Management*: funkcija upravljanja z napravami zagotavlja upravljanje naprav v MN (na usmerjevalnikih M2M), na ASN in ADN (npr. naprave M2M) ter naprav, ki so znotraj območnega omrežja M2M.
- *Discovery*: funkcija odkrivanja aplikacij in storitev išče informacije o aplikacijah in storitvah po ustreznih atributih in virih.
- *Group Management*: funkcija upravljanja s skupinami je zadolžena za zahtevke, ki so naslovljeni na skupino.
- *Location*: funkcija dovoli AE pridobivanje informacij o geografski lokaciji vozlišč (npr. ASN, MN) za uporabljanje storitev, ki so lokacijsko odvisne.
- *Network Service Exposure, Service Execution and Triggering*: funkcija upravlja s komunikacijami osnovnega omrežja za dostop do omrežnih funkcij storitev preko referenče točke Mcn.
- *Registration*: funkcija registracije obdela zahteve iz AE ali CSE z namenom registriranja v registrirnem CSE (ang. *Registrar CSE*), da lahko registrirane entitete uporabljajo storitve, ki jih ponuja registrirni CSE.
- *Security*: funkcija varnosti obsega naslednje funkcionalnosti: ravnanje z občutljivimi podatki, varnostno upravljanje, vzpostavitev varnostnih zvez, nadzorovanje dostopa (vključno z identifikacijo, preverjanjem pristnosti in avtorizacijo), upravljanje z identitetami.
- *Service Charging and Accounting*: zagotavlja funkcije zaračunavanja za storitveni nivo. Podpira več različnih modelov zaračunavanja, vključno s spletnim kreditnim nadzorom v realnem času.
- *Subscription and Notification*: zagotavlja obvestila, ki se nanašajo na naročnino, ki sledi spremembam vira (npr. brisanje vira).

A.10 Ontologija oneM2M

A.10.1 Razredi in lastnosti v Base Ontology

Na sliki A.3 so prikazani vsi razredi in vse lastnosti v ontologiji oneM2M Base Ontology. Podrobneje bomo opisali nekaj pomembnejših razredov.

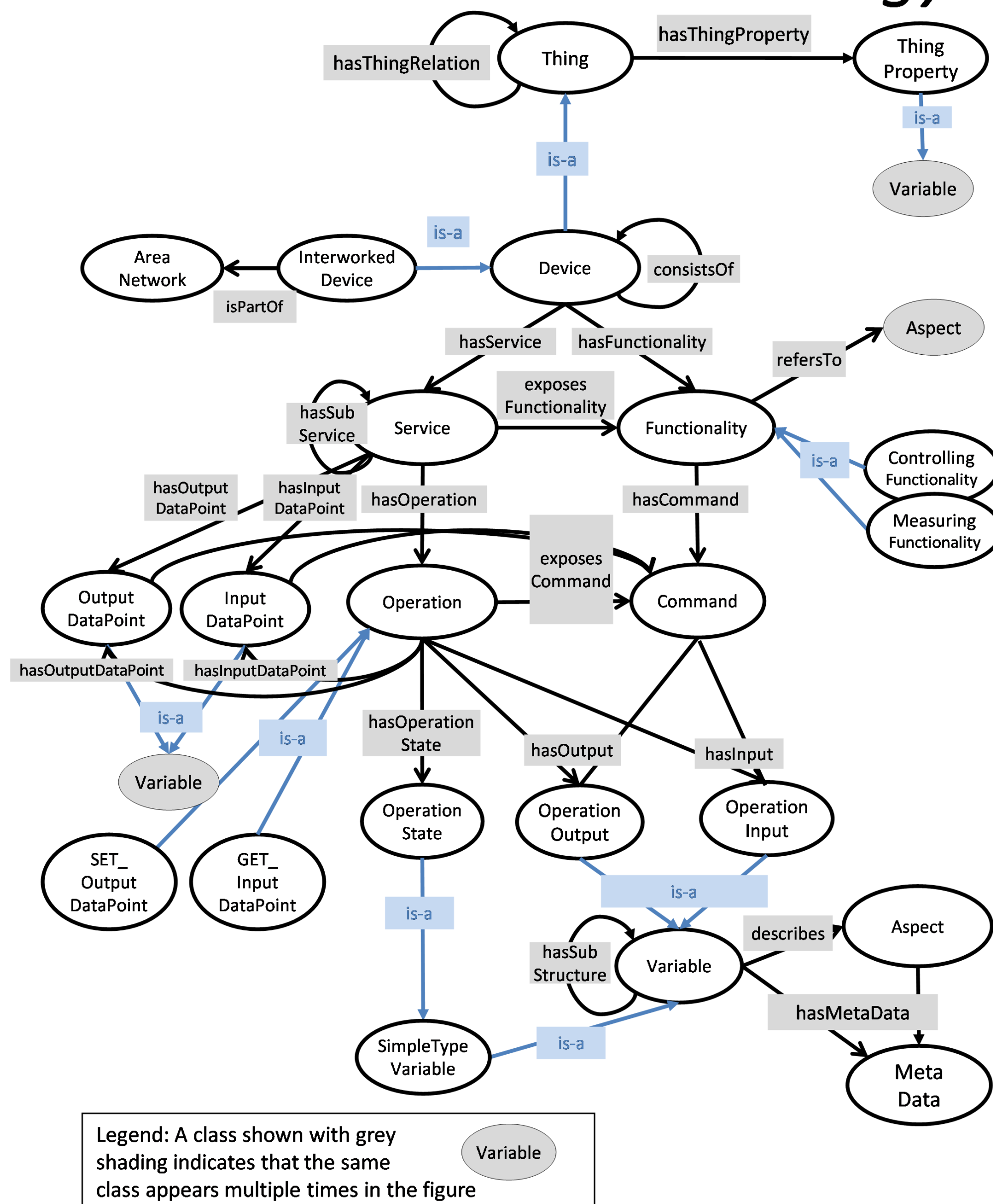
- Razred *Device* oz. naprava: je podrazred razreda *Thing* in ima zmožnost interakcije s svojim okoljem preko omrežja. Naprava je narejena za izvrševanje opravil. Vsebuje logiko in je zmožna proizvajanja in/ali sprejemanja podatkov, ki jih deli z ostalimi entitetami (napravami oz. stvarmi) v omrežju. Lahko je fizična ali nefizična entiteta. Za izvršitev opravila mora naprava izvesti eno ali več funkcionalnosti, ki so izpostavljene omrežju preko storitev naprave. Naprava je lahko sestavljena iz več naprav. Vsaka naprava (vključno s podnapravami) mora biti posamezno naslovljiva v omrežju.

- Razred *Interworked Device* oz. naprava, ki deluje vzajemno: je podrazred razreda *Device* in predstavlja napravo, ki je del območnega omrežja, ki ne podpira vmesnikov oneM2M in lahko dostopa do sistema oneM2M s komuniciranjem preko posredne (virtualne) naprave, ki jo ustvari IPE.

- Razred *Area Network*: predstavlja omrežje, ki zagotavlja storitve prenosa med vzajemno-delujočimi napravami in sistemom oneM2M. Različna omrežja lahko uporabljajo heterogene omrežne tehnologije, ki ne podpirajo protokola IP.

- Razred *Service* oz. storitev: je elektronska predstavitev funkcionalnosti (ang. *Functionality*) v omrežju. Storitev izpostavi funkcionalnost omrežju tako, da jo je mogoče najti, registrirati in z njo oddaljeno upravljati.

- Razred *Functionality*: predstavlja funkcionalnost, ki je potrebna za izvršitev opravila, kateremu je naprava namenjena.



Slika A.3: Razredi in lastnosti ontologije Base Ontology. Elipse predstavljajo razrede, puščice predstavljajo lastnosti. Razred, označen s sivo barvo, se na sliki pojavi večkrat. Povezava *is-a* nakazuje dedovanje (podrazred, pod-lastnost), ostale povezave so lastnosti objektov (pridobljeno iz [16]).

  - Podrazred *ControllingFunctionality* predstavlja funkcionalnost, ki vpliva na resnični svet, ampak ne zbira podatkov. Ima ukaze (ang. *Commands*) in/ali operacije (ang. *Operations*), ki prejemajo podatke. Termostat bi za *ControllingFunctionality* imel *"spreminjanje temperature"*.
  - Podrazred *MeasuringFunctionality* predstavlja funkcionalnost, ki ne vpliva na resnični svet, ampak podatke le zbira. Ima ukaze in/ali operacije, ki pošiljajo podatke. Temperaturni senzor bi imel *MeasuringFunctionality* za merjenje temperature.

- Razred *Operation* oz. operacija: predstavlja sredstvo storitvi za komunikacijo med napravami preko omrežja. Operacije so predstavljene na način, ki ga razumejo naprave in predstavljajo ukaze, ki so razumljivi ljudem. Operacija je prehodna, kar pomeni, da se lahko pokliče, proizvede rezultat in se dokonča. Na primer: AE pokliče operacijo naprave, ki sproži neko akcijo na napravi. Če ima operacija vhod, lahko vhodne argumente prejme od:

  - vhodnih podatkovnih točk (ang. *InputDataPoints*), ki so obstojne entitete, in/ali
  - razreda *OperationInput* (prehodna entiteta, katere podatki se pobrišejo, ko se operacija zaključi).

  Če ima operacija izhod, lahko podatke pošlje na izhodne podatkovne točke (ang. *OutputDataPoints*) ali na *OperationOutput*.

- Razred *Command* oz. ukaz: predstavlja akcijo, ki se lahko izvede in tako podpre funkcionalnost. Ukaz je človeku razumljiva akcija, ki gre vse do naprave. Operacija izpostavi ukaz omrežju. Razreda *OperationInput* in *OperatinOutput* predstavljata argumente ukazu. Funkcionalnost *"zatemnitev"* luči, ki lahko oddaljeno upravlja z lučjo, ima ukaz *"nastavi intenzivnost"*, kjer parameter zavzema vrednosti od 0 do 100%.

A.10.2 Instanciranje posameznih razredov v oneM2M

Instanciranje posameznih razredov v oneM2M je določeno na naslednje načine:

- Razred *Device* (ali podrazred) se instancira v vrednosti atributa *descriptor* vira tipa *semanticDescriptor*, ki je otrok vira AE. Instanca je identificirana z atributom *rdf:about*, ki vsebuje URI (npr. naslov MAC naprave), ki je unikaten v sistemu oneM2M.

- Razred *InterworkedDevice* (ali podrazred) se instancira na enak način kot *Device*, le da je vir *semanticDescriptor* otrok:

  - vira AE v IPE ali
  - vira *container* oz. *flexContainer*, ki je otrok vira AE v IPE.

  Instanca je identificirana z atributom *rdf:about*, ki vsebuje URI (npr. identifikator naprave v zunanjem sistemu), ki je unikaten v sistemu oneM2M.

- Razred *AreaNetwork* (ali njegov podrazred) je instanciran z vrednostjo atributa *descriptor* vira *semanticDescriptor*, ki je otrok vira, kateri instancira razred *InterworkedDevice*.

- Razred *Service* (ali podrazred) je instanciran v vrednosti atributa *descriptor* vira *semanticDescriptor*, ki je otrok vira *genericInterworkingService* (specializacija vira *flexContainer*). Instanca je identificirana z uporabo atributa *rdf:about*, ki vsebuje URI naprave skupaj z imenom razreda storitve. Ločena sta z znakom "*", npr. *00:11:22:1A:2B:3C-*MojaStoritev*. Vir *genericInterworkingService* je otrok vira AE, *container* ali *flexContainer*, ki vsebuje *semanticDescriptor*, ki instancira razred *Device*. Vsebuje reference na vire tipa *container* oz. *flexContainer*, ki predstavljajo vhode (predtavljene z razredom *InputDatapoints*) oz. izhode (predstavljene z razredom *OutputDatapoints*) storitve.

- Razred *Functionality* (ali podrazred) je instanciran v vrednosti atributa *descriptor*, vira *semanticDescriptor*, ki je otrok vira, ki instancira razred

Device. Instanca je identificirana z uporabo atributa *rdf:about*, ki vsebuje URI naprave, skupaj z imenom razreda funkcionalnosti, ločena z znakom "*" (npr. 00:11:22:1A:2B:3C*MojaFunkcionalnost).

- Razred *Command* (ali podrazred) je instanciran v vrednosti atributa *descriptor* vira *semanticDescriptor*, ki je otrok vira, ki instancira razred *Device*. Instanca je identificirana z uporabo atributa *rdf:about*, ki vsebuje URI naprave skupaj z imenom razreda ukaza, ločena z znakom "*" (npr. 00:11:22:1A:2B:3C*MojUkaz).

- Razred *Operation* (ali podrazredi) je instanciran v vrednosti atributa *descriptor* vira *semanticDescriptor*, ki je otrok vira *genericInterworkingOperationInstance* (specializacija vira *flexContainer*). Instanca je identificirana z uporabo atributa *rdf:about*, ki vsebuje URI naprave skupaj z imenom storitve in z imenom razreda operacije, ločenih z znakom "*". Na koncu dodamo številko, ki poskrbi za unikatno ime instance, dokler operacija obstaja. Storitev ima lahko tekom izvajanja naslednje identifikatorje instanc operacij:

  - 00:11:22:1A:2B:3C*MojaStoritev*MojaOperacija1,
    00:11:22:1A:2B:3C*MojaStoritev*MojaOperacija4
  - 00:11:22:1A:2B:3C*MojaStoritev*MojaDrugaOperacija1,
  - 00:11:22:1A:2B:3C*MojaStoritev*MojaTretjaOperacija13.

Dodatek B

Specifikacija GATT

B.1 Atributi (attributes)

Atributi so najmanjša podatkovna entiteta, ki jo določa GATT (oz. ATT). Posamezni atributi so naslovljivi. Vsebujejo lahko uporabniške podatke (ali metapodatke) o strukturi in grupiranju različnih atributov znotraj strežnika. GATT in ATT delata le z atributi, zato so informacije v odjemalcih in strežnikih oblikovane v atribute.

Specifikacija definira atribute le konceptualno in ne zahteva od implementacij ATT oz. GATT, da uporabljajo določene mehanizme in formate za shranjevanje podatkov. Vsak atribut vsebuje informacije o samem sebi in nato dejanske podatke [27].

B.2 Oprimek (handle)

Oprimek atributa je unikaten 16-bitni identifikator vsakega atributa na strežniku GATT. Oprimek nam omogoča naslavljanje atributa in se garantirano ne spreminja znotraj transakcij oz. znotraj obstoječe povezave. Oprimek 0x0000 predstavlja neveljavno vrednost oprimka, zato je na vsakem strežniku GATT možnih 0xFFFE oz. 65534 oprimkov. V praksi je število oprimkov par deset [27].

B.3 Tip (type)

Atribut tip je predstavljen z UUID. Ta je lahko 16-, 32- ali 128-biten. Tip določa vrsto podatkov, ki so zapisani v vrednosti atributa, in mehanizme, ki so na voljo za odkrivanje atributov glede na njihov tip.

Poznamo več vrst UUID. Lahko so standardni UUID, ki določajo hierarhijo strežnika GATT (kot so UUID storitve ali UUID karakteristike), UUID profila, ki določa vrsto podatkov v atributu, in UUID, ki jih določijo proizvajalci svojih naprav in so odvisni od implementacije [27].

B.4 Dovoljenja (permissions)

Dovoljenja so metapodatki, ki določajo operacije. Operacije za vsak atribut določajo varnostne zahteve [27]. Poznamo naslednje operacije:

- Dovoljenja dostopa: določajo ali lahko odjemalec prebere ali zapiše (ali oboje) vrednost atributa. Vsak atribut ima eno od naslednjih dovoljenj:
  - Brez dovoljenj (ang. *none*): Odjemalec atributa ne more prebrati in ga ne more zapisati.
  - Bralno dovoljenje (ang. *readable*): Odjemalec lahko prebere atribut.
  - Pisalno dovoljenje (ang. *writable*): Odjemalec lahko zapiše atribut.
  - Bralno in pisalno dovoljenje (ang. *readable and writable*): Odjemalec lahko prebere in zapiše atribut.
- Šifriranje: določa nivo šifriranja, ki je potrebno za dostop do tega atributa. V GATT so dovoljena naslednja šifriranja:
  - Brez šifriranja (ang. *No encryption required*): vrednost atributa je čistopis.

  - Šifriranje brez preverjanja pristnosti (ang. *Unauthenticated encryption required*): za dostop do tega atributa mora biti povezava šifrirana, ampak ni nujno, da je preverjena pristnost.
  - Šifriranje s preverjeno pristnostjo (ang. *Authenticated encryption required*): povezava mora biti šifrirana s šifrirnim ključem za dostop do tega atributa.

- Pooblastilo (ang. *Authorization*): določa kateri uporabnik ima pooblastila za dostop do atributa. Atribut lahko izbira le med opcijama za zahtevanje pooblastil ob dostopu ali dostop brez pooblastil.

B.5 Vrednost (value)

Vrednost atributa vsebuje dejansko vrednost podatka atributa. Vrednost je lahko poljubna, omejena je le dolžina in sicer na 512 bajtov.

Glede na tip atributa lahko vrednost vsebuje dodatne informacije glede atributov ali dejanske, uporabne, aplikacijske podatke [27].

B.6 Storitev (service)

Storitve GATT grupirajo konceptualno povezane atribute v eno skupno sekcijo atributov na strežniku GATT. Vsi atributi znotraj sekcije so po specifikaciji definicija storitve (ang. *service definition*). Vsaka definicija storitve se začne z atributom, ki označuje začetek storitve, in se imenuje deklaracija storitve (ang. *service declaration*).

Storitev je deklarirana na naslednji način:

- oprimek: 0xNNNN,
- zip: $\text{UUID}_{primary}$ ali $\text{UUID}_{secondary}$,
- dovoljenja: samo za branje,
- vrednot: UUID storitve,

- dolžina vrednosti: 2, 4 ali 16 bajtov.

Primarna storitev ($\text{UUID}_{primary}$) ima vrednost 0x2800, sekundarna storitev ($\text{UUID}_{secondary}$) ima vrednost 0x2801. Vrednosti so standardne, specificirala jih je SIG, in predstavljajo vrsto storitve.

Primarna storitev je standardni tip storitev v GATT, ki opisuje funkcionalnost strežnika GATT. Namen sekundarne storitve je vključevanje v ostalih primarnih storitvah, uporablja se za opisovanje in nima pravega pomena kot primarna storitev. V praksi se sekundarne storitve redko uporabljajo [27].

Deklaracija storitve se vedno pojavi na prvem mestu v definiciji, kateri navadno sledijo karakteristike in deskriptorji (do deklaracije naslednje storitve).

Znotraj definicije storitve se lahko pojavijo deklaracije vključenih storitev (ang. *include declaration*), ki vsebujejo vse potrebne informacije za odjemalca, da lahko naslovi vključeno storitev. Vključena storitev je deklarirana na naslednji način:

- oprimek: 0xNNNN,
- tip: $\text{UUID}_{include}$,
- dovoljenja: samo za branje,
- vrednost: oprimek vključene storitve, zadnji oprimek storitve, UUID vključene storitve
- dolžina vrednosti: 6, 8 ali 20 bajtov.

$\text{UUID}_{include}$ ima vrednost 0x2802 in je standardni UUID, določen s strani SIG. Vrednost vsebuje prvi in zadnji oprimek storitve (da vemo, s katerim atributom se storitev začne in s katerim konča), kot tudi UUID vključene storitve [27].

B.7 Karakteristika (characteristic)

Karakteristika je vsebnik za uporabniške podatke. Vedno vključujejo dva atributa: deklaracijo, ki vsebuje metapodatke o uporabniških podatkih, in

vrednost, ki vsebuje uporabniške podatke. Vrednosti lahko sledijo deskriptorji, ki vsebujejo dodatne metapodatke o vrednosti karakteristike. Vse skupaj tvori definicijo karakteristike, ki ji na kratko pravimo karakteristika.

Karakteristika je definirana na naslednji način:

- Deklaracija karakteristike:
  - oprimek: 0xNNNN;
  - tip: $\text{UUID}_{characteristic}$;
  - dovoljenja: samo za branje;
  - vrednost: Lastnosti, oprimek vrednosti (0xMMMM), UUID karakteristike;
  - dolžina vrednosti: 5, 7 ali 19 bajtov.
- Vrednost karakteristike:
  - oprimek: 0xMMMM;
  - tip: UUID karakteristike;
  - dovoljenja: katerokoli;
  - vrednost: dejanska vrednost;
  - dolžina vrednost: spremenljiva (odvisno od vrednosti).

Karakteristike so vedno del storitve, zato jih vedno najdemo v njih.

Vrednost $\text{UUID}_{characteristic}$ je enaka 0x2803 in je standardna, določena s strani SIG. Vedno označuje začetek karakteristike.

Naslednja tri polja so vsebovana v vrednosti deklaracije karakteristike:

- lastnosti: 8 bitov za določanje dovoljenj na karakteristiki;
- oprimek vrednosti: 16 bitov za določanje oprimka, na katerem se nahaja vrednost karakteristike;
- UUID karakteristike: 2, 4 ali 16 bajtov za določanje UUID-ja karakteristike.

Lastnosti zavzemajo 8 bitov in dodatna 2 bita, ki sta določena v razširjenih lastnostih (ang. *extended properties*). Lastnosti so lahko naslednje:

- Broadcast: če je ta bit nastavljen, se vrednost karakteristike lahko pojavi v oglaševalskih paketih.
- Read: če je ta bit nastavljen, je odjemalcu omogočena uporaba vseh bralnih operacij, ki jih določa ATT.
- Write without response: če je ta bit nastavljen, lahko odjemalci uporabijo operacijo *Write Command*, ki jo določa ATT.
- Write: če je ta bit nastavljen, je odjemalcem dovoljena uporaba operacije *Write Request/Response*, ki jo določa ATT.
- Notify: če je ta bit nastavljen, je strežniku dovoljena uporaba operacije *Handle Value Notification*, ki jo določa ATT.
- Indicate: če je ta bit nastavljen, je strežniku dovoljena uporaba operacije *Handle Value Indication/Confirmation*, ki jo določa ATT.
- Signed Write Command: če je ta bit nastavljen, je odjemalcem dovoljena uporaba operacije *Signed Write Command*, ki jo določa ATT.
- Queued Write (razširjene lastnosti): če je ta bit nastavljen, je odjemalcem dovoljena uporaba operacije *Queued Writes*, ki jo določa ATT.
- Writable Auxiliaries (razširjene lastnosti): če je ta bit nastavljen, lahko odjemalec piše v deskriptor z uporabniškim opisom karakteristike (ang. *Characteristic User Description Descriptor*).

Odjemalec lahko prebere lastnosti, da izve, katere operacije so dovoljene na karakteristiki.

Oprimek vrednosti v 16-ih bitih vsebuje oprimek atributa, ki določa vrednost karakteristike. Navadno je oprimek vrednosti za ena večji od oprimka deklaracije karakteristike, vendar tega ne smemo domnevati, saj to ni določeno v specifikaciji (se pa vrednost pojavi takoj po deklaraciji).

UUID karakteristike je lahko standarden UUID, določen s strani SIG, ali pa UUID, ki ga določi proizvajalec naprave.

Vrednost karakteristike vsebuje dejanske uporabniške podatke, ki jih lahko odjemalec prebere iz karakteristike oz. zapiše v karakteristiko. Tip tega atributa je vedno enak UUID, ki se nahaja v vrednosti deklaracije karakteristike. Vrednosti karakteristike tako nimajo več tipov storitev oz. karakteristik, ampak konkretne UUID, ki se nanašajo na vrednost, prebrano iz senzorja, ali vrednost pritisnjene tipke. Vrednost tega atributa je lahko poljubna (vse kar je možno poslati preko BLE in lahko zapolni prostor vrednosti) [27].

B.8 Deskriptor (descriptor)

Deskriptor priskrbi odjemalcu dodatne metapodatke o karakteristiki in njeni vrednosti. Nahaja(jo) se znotraj definicije karakteristike, takoj po vrednosti karakteristike. Deskriptorji vedno obsegajo en atribut - deklaracija deskriptorja karakteristike - čigar UUID je tip deskriptorja. Vrednost deskriptorja vsebuje, kar opisuje ta tip deskriptorja. Poznamo dva tipa deskriptorjev:

- Deskriptorji GATT: so temeljni, pogosto uporabljeni tipi deskriptorjev, ki vsebujejo metapodatke o karakteristiki.

- Deskriptorji specifičnih proizvajalcev: profil Bluetooth je lahko izdelan po specfikacijah SIG ali po specifikacijah proizvajalca naprave; v vsakem primeru lahko ti deskriptorji vsebujejo kakršnekoli informacije o vrednosti karakteristike.

Pogosto uporabljeni deskriptorji, ki jih določa GATT:

- Deskriptor razširjenih lastnosti (ang. *Extended Properties Descriptor*): vsebuje dva dodatna bita, ki jih dodamo lastnostim v vrednosti deklaracije karakteristike.

- Deskriptor z uporabniškim opisom karakteristike (ang. *Characteristic User Description Descriptor*): vsebuje berljivo vsebino - niz znakov, kodiran z UTF-8.

- Deskriptor *Client Characteristic Configuration Descriptor* oz CCCD: je najbolj pogosto uporabljen in se uporablja kot stikalo, ki omogoča ali onemogoča posodobitve s strani strežnika (ang. *Server-Initiated Updates*), ampak le za karakteristiko v kateri se nahaja. CCCD vsebuje dva bita, enega za obvestila (ang. *notifications*), drugega za naznanila (ang. *indications*). Razlika med obvestilom in naznanilom je ta, da je za vsako naznanilo potrebno poslati potrditev, da je to paket, ki smo ga potrebovali. Zaradi tega so naznanila počasnejša. Odjemalec lahko poljubno nastavlja oba bita. Strežnik ju preveri, ko karakteristika spremeni vrednosti in je posodobitev pripravljena za pošiljanje. Kadar želi odjemalec omogočiti obvestila oz. naznanila, uporabi operacijo *Write Request* in zapiše 1 v ustrezen bit. Strežnik odgovori z *Write Response* in začne pošiljati posodobitve, ko se vrednost karakteristike spremeni.

- Deskriptor *Characteristic presentation format descriptor*: vsebuje format vrednosti karakteristike, ki je predstavljen s sedmimi biti. Med formate spadajo tipi: logični tip (*boolean*), niz znakov (*string*), cela števila (*integer*), števila s plavajočo vejico (*floating-point*) idr.