УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Саша Радовановић

Реализација апликативног спрежног слоја послужиоца за *Internet of Things* екосистеме

МАСТЕР РАД

Нови Сад, 2014. година

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Дипломски – мастер рад
Аутор, **АУ**:		**Саша Радовановић**
Ментор, **МН**:		**Др Милан Бјелица**
Наслов рада, **НР**:		**Реализација апликативног спрежног слоја послужиоца за Internet of Things екосистеме**
Језик публикације, **ЈП**:		Српски / латиница
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		**2014**
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		**7/79/56/10/21/0/0**
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кqучне речи, **ПО**:		**Internet of Things, послужилац, управљање**
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		**У раду су реализовани модули апликативног спрежног слоја послужиоца за *Internet of Things* екосистеме који омогућују повезивање са другим апликацијама.**
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	**Др Иштван Пап**
	Члан:	**Др Растислав Струхарик** — Потпис ментора
	Члан, ментор:	**Др Милан Бјелица**

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Master Thesis	
Author, **AU**:		**Saša Radovanović**	
Mentor, **MN**:		**Milan Bjelica, PhD**	
Title, **TI**:		**Implementation of a server API for Internet of Things ecosystems**	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		**2014**	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		**7/79/56/10/21/0/0**	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S**/**KW**:		**Internet of Things, server, configuration**	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		**This paper describes implementation of Northbound API modules for Internet of Things server which enables communication between the server and applications.**	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	**Ištvan Pap, PhD**	
	Member:	**Rastislav Struharik, PhD**	Menthor's sign
	Member, Mentor:	**Milan Bjelica, PhD**	

Zahvalnost

Zahvaljujem se Milanu Bjelici na stručnoj pomoći, savetima, utrošenom vremenu i naporu zbog kojeg sam zavoleo ovu oblast struke.

Posebno se zahvaljujem rukovodstvu Instituta RT-RK na ukazanoj prilici da se bolje upoznam sa načinom rada u inženjerskom okruženju i budem uključen u proces razvoja novih programskih rešenja.

Zahvaljujem se Nataši na moralnoj podršci i strpljenju tokom izrade ovog rada.

Na kraju se zahvaljujem svima onima koji su na bilo koji način doprineli izradi ovog završnog rada.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

IoT	*- Internet of Things, Internet stvari*
API	*- Application Programming Interface, Sprežni pristupni sloj*
M2M	*- Machine-to-machine, Mašina-mašina komunikacija*
D2D	*- Device-to-device, Uređaj-uređaj komunikacija*
D2S	*- Device-to-server, Uređaj-poslužilac komunikacija*
S2S	*- Server-to-server, Poslužilac-poslužilac komunikacija*
GUI	*- Graphical User Interface, Grafički korisnički interfejs*
BDA	*- Big Data Analysis, Obrada velike količine podataka*
NBI	*- Northbound Interface, Aplikativni sprežni slojevi TR-069 poslužioca*
MI	*- Maintanence Index, Indeks lakoće održavanja koda*
UI	*- Usability Index, Indeks korisnosti*

1. Uvod

Razvoj Interneta je izmenio način na koji ljudi komuniciraju i koegzistiraju, omogućivši informaciju lako dostupnom svakom korisniku. Kontinuirano integrisanje svih sfera života je nastavljeno, pa se tako i poslovanje, ekonomija i industrija sve više oslanjaju na Internet usluge, učinivši čoveka centralnim faktorom oko kojeg se razvijaju nove mogućnosti i usluge (*Human-centric Internet*). Najnoviji talas razvoja Interneta umesto čoveka u fokus postavlja "*pametne*" i povezane uređaje (*connected devices*).

Kako bi interagovali sa drugim uređajima i čovekom, ovi uređaji moraju zajedno raditi brzinom, mogućnostima i obimom daleko iznad onih kojima se čovek kao korisnik Interneta služi. Ovakva umreženost – "Internet stvari" (*Internet of things, IoT*) menja globalnu mrežu i svet dublje nego što je evolucija čovek-centričnog Interneta to radila.

Povezani uređaji mogu biti različite fizičke prirode a zajedničko im je da komuniciraju kako međusobno, tako i sa jednom ili više centralnih tačaka sistema koje se nazivaju poslužioci (*servers*). Takvi sistemi sastavljeni od krajnjih uređaja i poslužioca se nazivaju **IoT ekosistemi (*IoT ecosystems*).** Uloga poslužioca je komunikacija sa krajnjim uređajima kako bi omogućili upravljanje, nadgledanje, automatsko konfigurisanje i adekvatnu grafičku prezentaciju.

U ovom radu je prikazano rešenje aplikativnog sprežnog sloja (*Application programming interface, API*) za jedan opšti IoT poslužilac. API omogućuje razmenu podataka poslužioca sa drugim aplikacijama i poslužiocima različitih zahteva, pružajući informacije kroz realizovane module koji koriste različite ISO OSI protokole transportnog nivoa, uz primenu sigurnosnih mehanizama čime se štiti integritet podataka u poslužiocu. Programsko rešenje ovog aplikativnog sprežnog sloja je realizovano u programskom jeziku Java uz korišćenje GWT RPC okruženja, REST/JSON protokola i web servisa zasnovanih na WSDL jeziku. Tako realizovan aplikativni sprežni sloj omogućuje prilagođavanje komunikacije tipu uređaja čime se postižu

bolje performance. Uz to, glavni doprinos rešenja predstavlja i podrška za isporuku velike količine nestrukturiranih podataka što omogućuje distribuiranje obrade između drugih poslužioca.

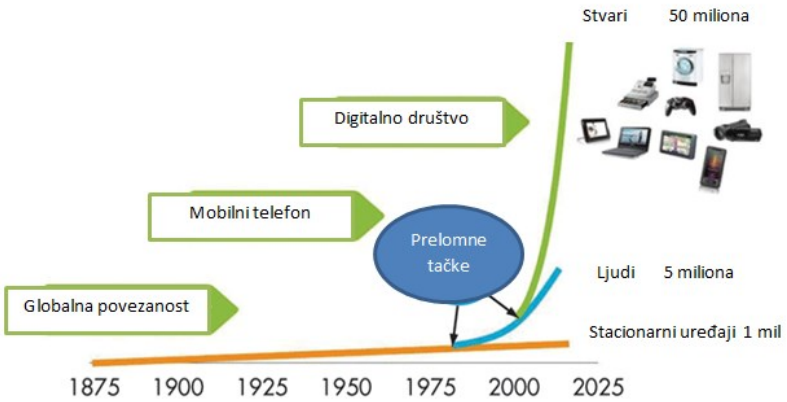
Rad je organizovan u nekoliko celina:

- Teorijske osnove – pokrivaju osnovne koncepte, zahteve i ulogu komunikacionih slojeva kod IoT poslužioca, uporedni prikaz karakteristika ISO OSI protokola transportnog i nivoa sesije korišćenih u IoT komunikaciji, kao i ulogu sprežnih slojeva u različitim vrstama IoT poslužioca postavljenih u *Cloud* okruženju.
- Koncept rešenja – opisuje funkcionalnosti koje realizovani aplikativni sprežni slojevi omogućuju spoljašnjim aplikacijama, sigurnosne mehanizme i pojedinosti različitih modula.
- Programsko rešenje – opis svih metoda aplikativnih sprežnih slojeva sa ulaznim parametrima i povratnim vrednostima.
- Ispitivanje i rezultat – prikazani su rezultati ispitivanja u realnim uslovima i poređenje sa sličnim sistemima.
- Zaključak – pokriva ispunjenost zadatka i budući rad.

2. Teorijske osnove

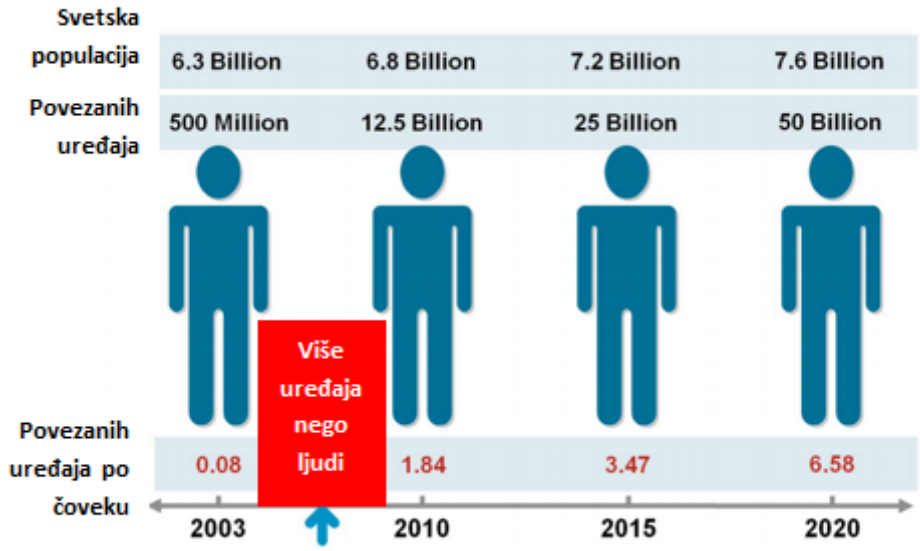
2.1 Internet of Things koncept

Današnji Internet povezuje web stranice i krajnje stacionarne uređaje (*workstations*). Po trenutnim podacima postoji i aktivno je približno milijardu web stranica [1]. Mobilna revolucija označava period kada povezivanje na Internet prestaje da bude odlika samo stacionarnih uređaja (PC računara) i kada prenosivi, mobilni uređaji postaju aktivni korisnici Internet usluga. Mobilnost ljudi daje zamajac razvoju pa je ovaj period počeo pre pet godina, a doveo je do toga da kvantitativno najveći korisnici Internet usluga postaju mobilni telefoni (*smartphones*), tablet, netbook i laptop računari. Broj prenosivih uređaja od nastanka ima eksponencijalni rast i trenutno dostiže broj od preko šest milijardi, tako prestigavši broj ljudi na Zemlji [2]. Projekcije govore da će na svetu do 2020. godine broj pametnih uređaja prevazići cifru od 50 miliona, kao što je prikazano na Slici 1. Zadatak IoT je da ih sve poveže na takav način da one formiraju distribuirane M2M (*machine-to-machine*) aplikacije. Pretpostavlja se da ću u punoj primeni, IoT umnožiti opterećenost i razmenu komunikacija na Internetu za faktor 50 [1].



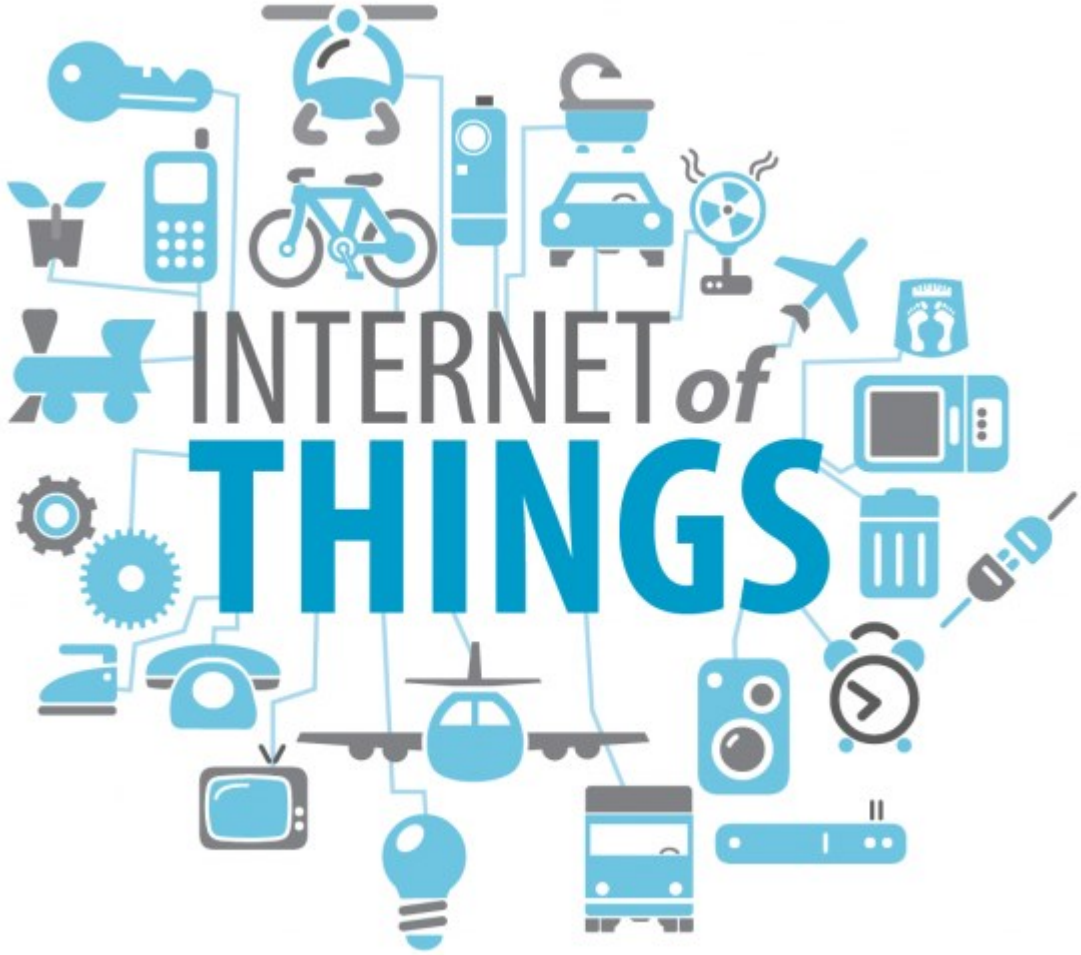
Slika 1. Projekcije broja entiteta na Internetu [1].

Pretpostavlja se da će Internet of Things umrežiti deset puta više uređaja nego što je mobilna revolucija povezala i da će predstavljati pokretačku snagu našeg vremena [1]. Veliki broj ustanova se bave istraživanjem razvoja tržišta u IoT domenu, i projekcije [1] govore da će *device-per-human* faktor rasti eksponencijalno. Projekcija promene tog faktora iz istraživanja kompanije Cisco [3] je prikazana na Slici 2. Prema podacima kompanije Cisco, 20 domaćinstava sa razrađenim IoT sistemom će 2025. godine generisati više razmene podataka između sebe nego što je to radio kompletan Internet u 2008. godini.



Slika 2. Projekcija promene *device-per-human* koeficijenta [3].

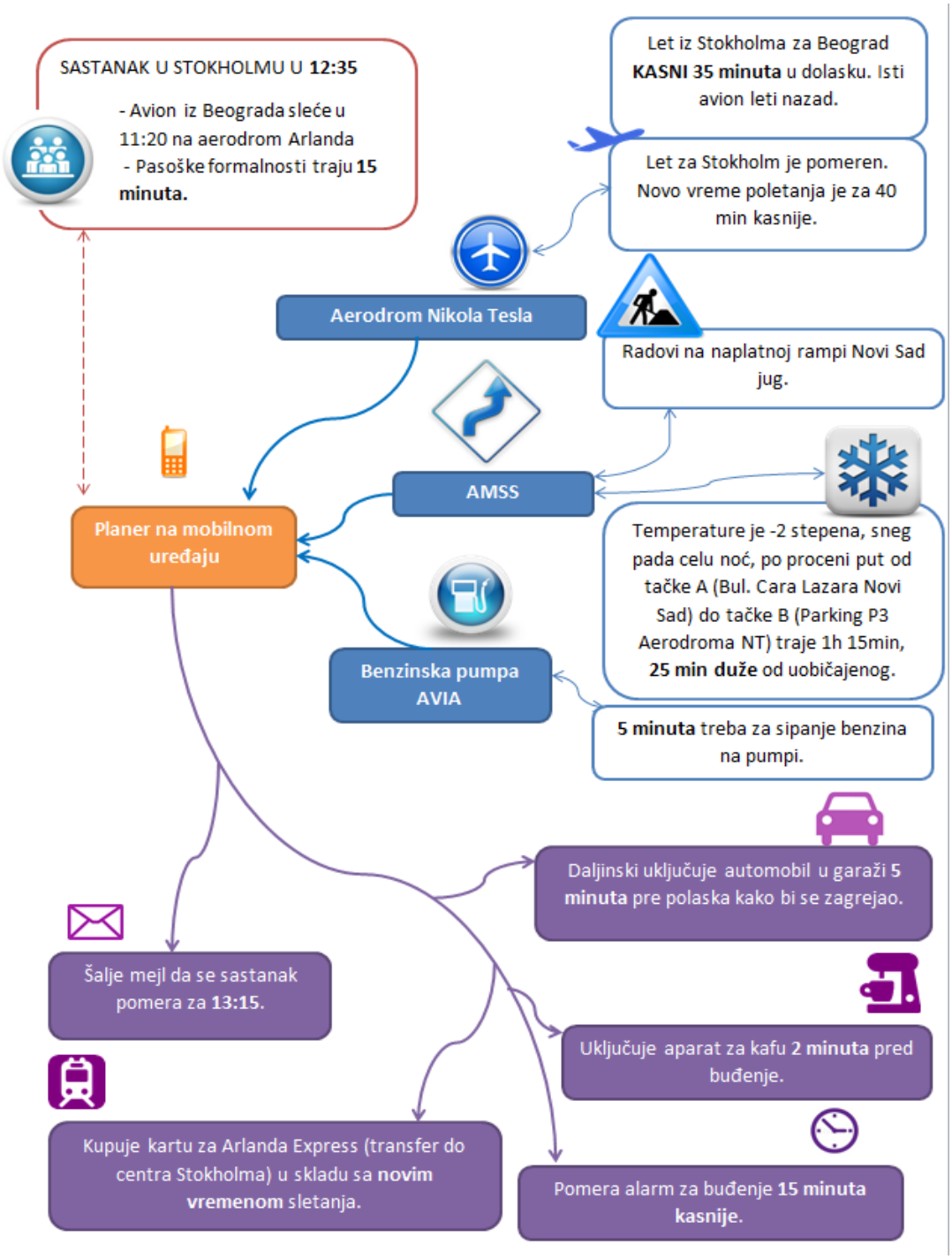
Stvari (*things*) koje će biti povezane ovakvim sistemom su već svuda prisutne u životu i svaki čovek ih poseduje. Automobili sa najnovijim tehnologijama koriste više od 100 procesora. Bela tehnika je sve kompleksnija, dok multimedijalni uređaji već nekoliko godina omogućuju povezanost. Pametni uređaji su integralni delovi industrijskih postrojenja, bolnica, kuća, transportnih sistema, farmi [4]. Danas su ovi uređaji slabo međusobno povezani ali se to ubrzano menja. Međupovezanost onog nivoa omogućuje aplikacije koje do sada nije bilo moguće realizovati. Raznorodnost u pogledu povezanih uređaja se ogleda u njihovoj veličini, koja može ići od senzora veličine nekoliko milimetara do uređaja veličine nekoliko metara. Funkcije uređaja mogu biti raznovrsne, od sklopova u svemirskim programima do senzora na životinjama na velikim farmama, kao što je prikazano na Slici 3. *Stvari* ne moraju biti samo nežive prirode, npr. čovek sa monitorom srčanog rada i životinja na farmi sa biočipom (*biochip transponder*) su mogući činioci sistema.



Slika 3. Uređaji u *Internet of Things* [44].

Ekosistem u IoT predstavlja skup čvorova (povezanih uređaja) koji delovanjem i komunikacijom međusobno i sa poslužiocima omogućuju kreiranje pametnih aplikacija. Primer jednog takvog mogućeg ekosistema prikazan je na Slici 4. Na mobilnom uređaju korisnik poseduje planer (*organizer*) koji je u prošlosti bio isključivo *user-driven*, podrazumevajući da je korisnik uređaja unosio informacije i podatke na osnovu kojih je aplikacija funkcionisala. Razvojem mobilnih aplikacija i IoT uređaja, planer tehnološki može omogućiti distribuciju uloge unosioca informacija (*information feeder*) između korisnika i ostalih relevantnih čvorova za određene događaje i tako omogućiti poboljšane funkcionalnosti korisniku.

U slučaju sa Slike 4, korisnik je pružio ulazne informacije (sastanak u određeno vreme u drugom gradu i način transporta do tog grada) tako što ih je uneo eksplicitno ili indirektno (čuvanje podataka iz elektronske pošte). Uvezivanje pametnih čvorova u IoT ekosistem omogućuje korišćenje uređaja direktnom komandom i čitanjem informacija, *device-to-device* (D2D) komunikacija, kao i indirektno preko poslužioca (*servera*) korišcenjem *device-to-server* (D2S) komunikacije.



Slika 4. Primer jednog IoT ekosistema.

U primeru sa slike planer na mobilnom uređaju pristupa sistemu za upravljanje i nadgledanje avio i drumskog saobraćaja preko poslužioca (Eurocontrol/Aerodrom odnosno Auto-moto Savez Srbije) dok procesorima na automobilu, sistemu za prodaju karata i aparatu za kafu pristupa direktno.

Na osnovu ulaznih informacija različite prirode, aplikacija (planer) donosi odluke, prenosi informacije i čini programirane komande koje za krajnji cilj imaju ostvarivanje cilja

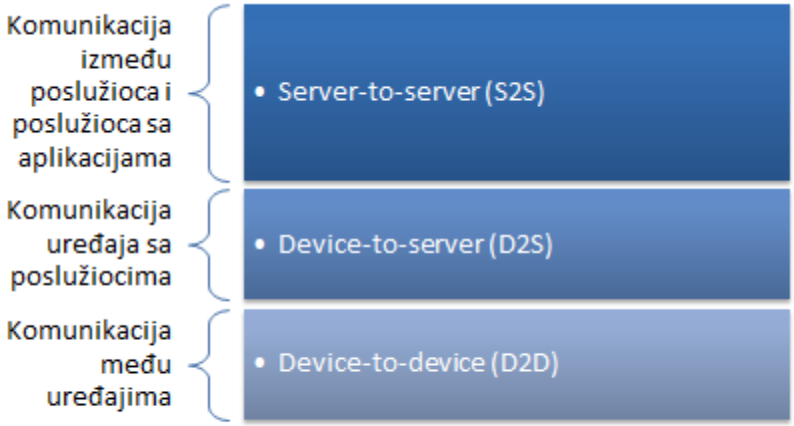
(stizanje na let i sastanak) uz efikasnije trošenje resursa (vreme, novac). Izvori tih informacija su raznovrsni i mogu biti senzori u avionima, radari, meteorološka oprema, ali i kompletni uređaji – automobili, aparat za kafu, mobilni telefon. Sve informacije i komande koje planer u ovom primeru prikuplja i isporučuje moraju biti odobrene od korisnika, odnosno pružaoca usluga. IoT koncept koji formuliše ovakve napredne mogućnosti mora biti dobro sigurnosno obezbeđen što predstavlja i jedan od glavnih problema ovakvih sistema.

2.2 Tipična arhitektura IoT sistem

Ceo IoT koncept počiva na scenariju gde svi objekti imaju *jedinstveni identifikator (UI)* i mogućnost da automatski prenose podatke preko mreže bez potrebe za čovek-čovek (*human-to-human*) ili čovek-mašina (*human-to-machine*) interakcijom [5]. Koncept se razvio iz pojave bežičnih komunikacija (*wireless*), mikro-elektromehaničkih sistema (MEMS) i evolucije Interneta. Rad na ideji je fokus već decenijama, pa je tako 1980. godine rashladna vitrina Coca Cola bila prvi uređaj priključen na Internet. Uprkos tome, tek u poslednjih deset godina on poprima i formu i formalni naziv – Internet of Things, a zbog neunificiranih istraživanja, standarda i pristupa problemu, sam koncept je i danas heterogen [6].

Ono što je usvojeno kao polazni postulat u svim rešenjima i istraživanjima je da je objekat u IoT ekosistemima predstavljen minimalno IP adresom kao najlakšim načinom za praktičnu implementaciju koncepta identifikatora. Ograničenost adresnog prostora (IPv4) je problem koji se prevazilazi sa postepenim prelaskom na veći IPv6 sistem, koji omogućuje konekciju većem broju uređaja [7] [1].

Heterogenost IoT sistema je dovela do brojnih pokušaja integracije sa različitim protokolima, standardima i arhitekturama. Ipak komunikacija kompletnog sistema se može razložiti u tri nivoa, kao što je prikazano na Slici 5.



Slika 5. Nivoi komunikacije u IoT sistemima.

Najniži nivo komunikacije [1] (***Device-to-device, D2D***) predstavlja interakciju između čvorova uglavnom bez kontrole poslužioca i/ili korisnika sistema. Poznat i kao M2M (*machine-to-machine*), ovaj sloj unosi najveće opterećenje za mrežnu infrastrukturu ali unosi i povezanost koja se u nekim istraživanjima predstavlja i kao vid veštačke inteligencije (*Artificial Inteligence*) [8]. Mogućnost prikljupljanja informacija i donošenja odluka bez uplitanja korisnika sistema (čoveka) je jedna od najvećih inovacija IoT sistema.

Komunikacija poslužioca sa svim svojim čvorovima se obavlja preko ***Device-to-server, D2S*** sloja. Ovaj sloj predstavlja spregu između funkcionalnosti uređaja i naprednih korisničkih aplikacija. Uređaji po pravilu ne poseduju velike resurse koji bi poskupeli proces proizvodnje, pa se teži rešenju u kojem se napredna logika i dodatna statistika, upravljanje i distribucija podataka prebacuje na poslužioce. Komunikacija sa poslužiocem ne uskraćuje ili eliminiše D2D sloj komunikacije, a za razliku od njega ova komunikacija nije intenzivna, česta i ne opterećuje mrežne resurse u toj meri.

Poslužioci međusobno i sa spoljašnjim aplikacijama (*third-party applications*) ostvaruju razmenu informacija radi ostvarivanja pune funkcionalnosti. Jedan ekosistem je uspešan onoliko kolika je njegova upotrebna vrednost, a kako na jedan ekosistem utiču razne vrste događaja, on kontroliše pojave različite prirode. Kako bi sistem obezbedio kvalitetnu prezentaciju i obradu podataka, neophodna je komunikacija između poslužioca najvišeg nivoa, koja se odvija preko ***S2S (Server-to-server)*** sloja.

Vremenski zahtevi za komunikacijom u pojedinačnim slojevima [1] su predstavljeni u Tabela 1.

	Device-to-device	Device-to-Server	Server-to-Server
Vremenski odziv	10us – 10ms	10ms – 1s	>1s

Tabela 1. Vremenski zahtevi slojeva komunikacije u IoT.

2.3 Uloga poslužioca

Poslužioci u IoT imaju ulogu da prikupljaju podatke od uređaja (kroz D2S sloj), da te informacije podele sa odgovarajućim entitetima u ekosistemu (druga aplikacija, drugi uređaj, čovek), da analiziraju dati problem, i da po potrebi te podatke predstave krajnjem korisniku ili komanduju krajnjim uređajem [9]. Poslužioci se mogu podeliti po svojoj nameni:

1. Opšti poslužioci,
2. Prezentacioni poslužioci,
3. Komandni poslužioci.

Opšti poslužioci (*general servers*) prikupljaju podatke sa uređaja, skladište ih i po potrebi obrađuju periodično ili nakon prijema određenog događaja. Pored logike, oni imaju sprežne slojeve (*API*) koji omogućuju razmenu podataka sa drugim poslužiocima ili IoT aplikacijama. Većina opštih poslužioca poseduje i svoj korisnički interfejs koji se najčešće naziva kontrolni panel (*control panel*). Kod opštih poslužioca se intenzivno koriste sva tri sloja komunikacije. Primer jednog opšteg poslužioca je *ProSyst* poslužilac [10] namenjen senzorima i aktuatorima.

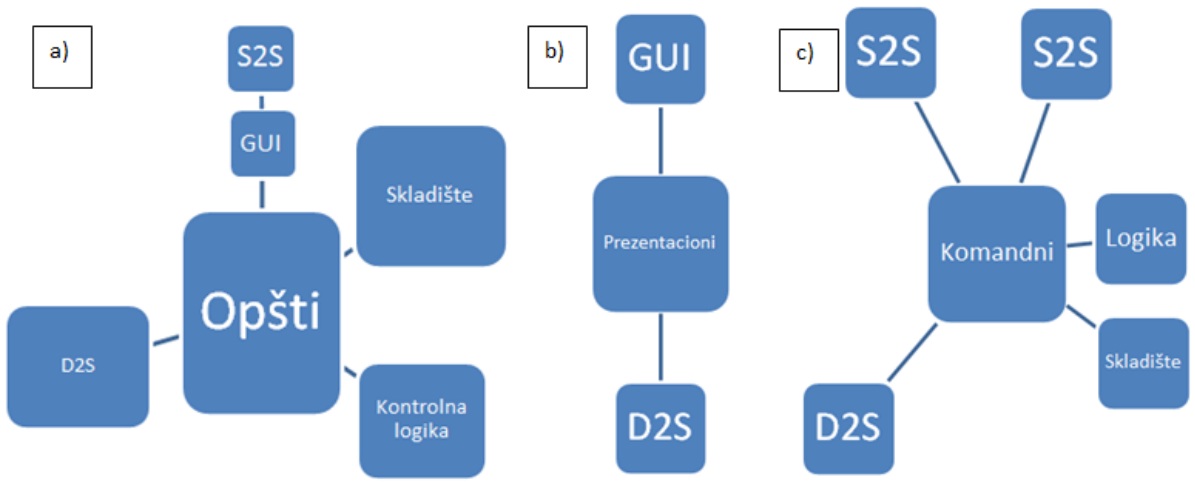
Prezentacioni poslužioci su oni poslužioci koji imaju samo C&P (*collect and present*, prikupi i predstavi) ulogu. Njihov zadatak je dobavljanje podataka sa krajnjih uređaja i prezentovanje krajnjem korisniku kroz svoju grafičku korisničku spregu (*Graphical User Interface, GUI*). Najčešće GUI je integrisan sa poslužiocem, mada može biti izdvojen posebno, ali sam poslužilac ne obezbeđuje vezu sa drugim poslužiocima. Nazivaju se nekada i informacioni poslužioci i kod njih se S2S sloj jednosmerno upotrebljava, a svrha im je isključivo da informišu korisnike o pojedinim promenama u sistemu.

Nimbits [11] je primer prezentacionog poslužioca za IoT ekosisteme. Predstavlja rešenje otvorenog koda i bazira se na intenzivnoj D2S komunikaciji. Prilikom svake komunikacije krajnjeg uređaja sa poslužiocem, iniciraju se događaji koji aktuelizuju grafičku korisničku spregu koja je integrisana zajedno sa poslužiocem.

Komandni poslužioci uglavnom ne poseduju svoj GUI u većini slučajeva, ili je on minimalističke prirode. Skladištenje podataka i reagovanje na sistemske promene im je glavni zadatak. Zbog svog zadatka imaju više fizičkih resursa na raspolaganju i poštuju striktne vremenske okvire. D2D sloj je kod njih maksimalno iskorišćen, a najčešće prezentovanje podataka prepuštaju prezentacionim poslužiocima ili drugim aplikacijama. Primeri komandnih poslužioca su *Lightweight M2M* (LWM2M) [12], Arkessa [13], Xively [14] i Etherios [15].

Gradivni blokovi tipova poslužioca su prikazani na Slici 6, opšti poslužilac (a), prezentacioni poslužilac (b) i komandni poslužilac (c).

Sva tri tipa poslužioca mogu biti sastavni delovi većih ekosistema kao jednog prikazanog na Slici 4. Sem funkcionalne podele, poslužioci se mogu podeliti i po fizičkim karakteristikama (na poslužioce sa slabijim i jačim performansama), po načinu funkcionisanja (lokalno postavljene i postavljene u *Cloud* okruženje), po prirodi krajnjih uređaja (na generalne poslužioce koji opslužuju bilo koji tip uređaja i specifične poslužioce koji poslužuju samo uređaje iz određene grupe).



Slika 6. Opšti (a), prezentacioni (b) i komandni poslužioci (c).

2.4 Protokoli za IoT

Mogućnosti za razvoj i integraciju IoT sistema u svakodnevni život preko elektronskih uređaja privlače dosta naučnih i komercijalnih institucija za tehnološki razvoj u ovoj oblasti. Problem usaglašenosti je jedna od najvećih barijera za potrošača i jedna od najvećih briga za nove generacije proizvođača softvera i hardvera. Problem usaglašenosti formata je već prisutan sa dva najprisutnija operativna sistema u mobilnim uređajima – Android i iOS operativnom sistemu.

Ukoliko korisnik na iOS operativnom sistemu kupi film u Apple prodavnici iTunes, taj isti film nije moguće gledati na uređaju zasnovanom na Android OS. IoT kompanije će ovaj problem uvećati s obzirom na heterogenost operativnih sistema u uređajima. Da bi koncept IoT bio koristan u punoj meri, podaci sa uređaja treba da budu otvoreni što bi podrazumevalo dostupnost API slojeva sa krajnjih uređaja i poslužioca. Bez otvorene platforme i regulatornih standarda, IoT ekosistemi nikada ne mogu dostići puni korisnički potencijal.

Dosta pokušaja regulatornih tela, između ostalih i IEEE, je bilo da se kompletna komunikacija standardizuje, međutim vođeni komercijalnim potrebama i tržištem, vodeći proizvođači su mahom razvijali i plasirali sopstvene standarde. Za svaki sloj komunikacije postoje desetine protokola ISO OSI aplikativnog nivoa od kojih je veliki broj i dalje ostao prisutan pa je odabir pravog protokola za integraciju u ekosistem težak, jer nijedan protokol u sva tri sloja nije idealan, pa se korišćeni protokol prilagođava tipu uređaja i načinu komunikacije.

Različiti slojevi u IoT ekosistemima zahtevaju različite perfomanse u komunikaciji pa se tako protokoli razlikuju [16]. Neki od najčešće korišćenih protokola aplikativnog ISO OSI nivoa su:

1. **CoAP** (*Constrained Application Protocol*) [17] predstavlja nisko zahtevan protokol primarno namenjen veoma jednostavnim elektronskim uređajima koji im dopušta da komuniciraju interaktivno preko Interneta. Cilja uglavnom senzore niske potrošnje, prekidače, ventile i slične elektromehaničke komponente. Koristi se isključivo u D2D i nekada u D2S komunikaciji.
2. **Continua** (*Continua Health Alliance*) [18] predstavlja protokol razvijen od strane preko dve stotine kompanija iz industrije medicinskih aparata i pomagala. Zdravstveno-medicinski IoT ekosistemi se sve više razvijaju i pojavljuju a Continua pruža specijalizovan protokol za ovaj deo tržišta. U jezgru je sastavljen od *Bluetooth LowEnergy* i *ZigBee* protokola.
3. **DDS** (*Data Distribution Service*) [19] je protokol iz M2M porodice protokola. Zamišljen je kao skalabilni, zavisni protokol u realnom vremenu zasnovan na entitetima i objektima. Grane industrije gde se primenjuje su finansijsko poslovanje, kontrola avio-saobraćaja, pametne mreže (*smart-grid*) i druge gde je zahtev za velikim protokom podataka primaran.
4. **HTTP/REST** (*Hypertext Transfer Protocol/Representational State Transfer*) [20] je kombinacija dva protokola u širokoj upotrebi danas. Pretpostavlja se da preko 35% današnjih IoT ekosistema koriste ovu kombinaciju, što u tolikoj heteregenosti predstavlja veliki broj. HTTP je jedan od osnovnih protokola za Internet komunikaciju a REST predstavlja programsku arhitekturu i preporuku za korišćenje HTTP protokola u komunikaciji. Koriste se u sva tri sloja komunikacije, a njegovo korišćenje zahteva napredne sigurnosne mehanizme, što mu se i pripisuje kao jedna od najvećih mana uz nemogućnost održavanja komunikacionih sesija.
5. **MQTT** (*MQ Telemetry transport*) [21] je "*objavi/pretplati se*" tip protokola. Nisko-zahtevan protokol je zasnovan na porukama koje se koriste preko TCP/IP arhitekture. Protokol nije otvorenog tipa a vremenom su se razvile i verzije protokola koje ne koriste TCP/IP već ZigBee za svoju osnovu.
6. **UPnP** (*Universal Plug and Play*) [22] je skup mrežnih protokola koji omogućuje uvezanim uređajima (personalni računari, štampači, Wi-Fi pristupne tačke i mobilni uređaji) da bez uticaja sa strane međusobno otkriju prisustvo uređaja i ostvare funkcionalne mrežne usluge za deljenje podataka, komunikaciju i zabavu. Inicijalno je zamišljen za mreže koje površinski nisu velike a koriste se uglavnom kod D2D sloja i D2S sloja (u paru sa prezentacionim poslužiocima).
7. **ZeroMQ** (*0MQ*) [23] je protokol za mreže visokih performansi. Koristi asinhroni sistem razmene poruke i prisutan je u distribuiranim i konkurentnim sistemima.

Njegova specifičnost se ogleda u postojanju reda za poruke (*message queue*) ali za razliku od drugih protokola orijentisanih na poruke, ne zahteva dispečera. Koristi se kako u D2S komunikaciji velikog obima, tako i u S2S komunikaciji između velikih poslužiočkih sistema.

8. **XMPP** (Extensible Messaging and Presence Protocol) [24] je protokol namenjen uređajima i sistemima zasnovanim na porukama. Koristi se za brze poruke, otkrivanje prisustva i radi u realnom vremenu. Funkcioniše kao "*objavi/pretplati se*" tip protokola i uz MQTT predstavlja protokol koji se najviše koristi u IoT.
9. **SNMP** (Simple Network Management Protocol) [25] je standard za kontrolisanje uređaja u mrežama zasnovanim na IP protokolu. SNMP pruža spoljašnjoj aplikaciji ili poslužiocu direktan podskup parametara za čitanje/upravljanje čiji je broj direktno zavistan od konfiguracije krajnjeg uređaja.
10. **DPWS** (Devices Profile for Web Services) [26] definiše minimalni skup funkcionalnosti kako bi poruke zasnovane na Web servisima bile skalabilne u zavisnosti od robusnosti uređaja. Ima slične ciljeve kao UPnP protokol s ključnom razlikom da DPWS cilja Web servise specifično, pa tako ima i dodatne funkcionalnosti. DPWS nije sam po sebi protokol, ali je smernica koja je implementirana u većini sledećih protokola zasnovanim na XML ili SOAP porukama.
11. **WSDL** (Web Service Definition Language) [27] je jezik zasnovan na XML modelu podataka za definiciju web servisa. Omogućava pružanje API sloja udaljenim servisima kroz krajnje tačke sistema (*system endpoints*), koji su implementirani kao .wsdl datoteke. Kao fizički protokol koristi SOAP. Koristi se intenzivno u S2S komunikaciji, ali i u D2S komunikaciji sa entitetima gde ekstremna brzina nije ključna.
12. **SOAP** (Simple Object Access Protocol) [28] je protokol koji specificira strukturu poruka u web servisima. Predstavlja osnovu za dalju nadogradnju i neke od tih nadogradnji se koriste u IoT kao npr. **TR-069** ili **CWMP** (CPE WAN Management Protocol) [29] koji je u upotrebi sa multimedijalnim uređajima, skladištima podataka, smart-grid sistemima i sličnim krajnjim uređajima koji zahtevaju intenzivnu i obimnu razmenu podataka. CWMP se koristi skoro isključivo u D2S komunikaciji i smatra se direktnom konkurencijom SNMP protokolu.

Uzevši u obzir početni postulat IoT sistema da je svaki uređaj dostupan preko Interneta na osnovu svog jedinstvenog identifikatora (u najvećem broju slučajeva IP adrese) većina protokola koristi UDP ili TCP protokol kao bazični protokol transportnog nivoa na koji se nadograđuju. Postoje u manjem broju i sistemi koji zaobilaze ovakvo pravilo korišćenjem drugih ISO OSI

transportnih protokola (ZigBee). U osnovi komunikacije leže mahom UDP datagram ili "*TCP socket*" mehanizmi. Uz to svi protokoli se mogu podeliti na *"zahtev/odgovor"* (*Request/response, R/R*) ili "*objavi/pretplati se*" (*Publish/subscribe, P/S*) protokole.

Mehanizam *Zahtev/Odgovor* protokola je zasnovan na pojedinačnim porukama u klijent/server arhitekturi. Uvek odgovoru prethodi zahtev, bilo da je on stigao od poslužioca ili krajnjeg uređaja. "*Objavi/pretplati se*" je noviji sistem koji je više prilagođen prirodi IoT ekosistema. Ovakav sistem smanjuje opterećenje na mrežu i lakše vrši kontrolu pristupa određenim funkcionalnosti, gde se za razliku od *"zahtev/odgovor"* mehanizma, ona ne obavlja pri svakoj razmenjenoj poruci.

Protokoli imaju različite odzive pri različitim brzinama i raspolažu različitim sigurnosnim mehanizmima. Uz to zahtevaju različite fizičke resurse i koriste razne arhitekturu, pa su tako polja gde su zabeležili uspehe raznorodna, od medicinskih do vojnih sistema.

Uporedni prikaz karakteristika svih pomenutih protokola sa osnovnim odlikama kao što su transportni nivo, način razmene poruka i karakteristike u specifičnim sistemima je dat u Tabela 2, a uporedni prikaz arhitekture i najčešće upotrebe je prikazan u Tabela 3.

Protokol	**Transport**	**Poruke (R/R – request/response, P/S – publish/subscribe)**	**Podrška za veće brzine**	**Adaptivnost na niskoenergetske i lossy sisteme**
CoAP	UDP	R/R	Odlična	Odlična
Continua	UDP	R/R i P/S	Delimična	Delimična
DDS	UDP	R/R i P/S	Delimična	Loša
HTTP/REST	TCP	R/R	Odlična	Delimična
MQTT	TCP	P/S i R/R	Odlična	Dobra
SNMP	UDP	R/R	Odlična	Delimična
UPnP	-	P/S i R/R	Odlična	Dobra
XMPP	TCP	P/S i R/R	Odlična	Delimična
ZeroMQ	UDP	P/S i R/R	Delimična	Delimična
WSDL	TCP	R/R	Delimična	Delimična
SOAP	TCP	R/R	Dobra	Delimična

Tabela 2. Uporedni prikaz protokola (osnovne odlike) [16].

Protokol	Sigurnost (op-opcionalno)	Najuspešniji primer primene	Arhitektura
CoAP	Osrednja – op	Energetski sistemi	Stablo
Continua	Nema	Medicinski sistemi	Zvezda
DDS	Visoka – opcionalna	Vojni sistemi	Magistrala
HTTP/REST	Niska – opcionalna	Pametne mreže	Klijent/server
MQTT	Osrednja- op	IoT upravljanje	Stablo
SNMP	Visoka – op	Nadgledanje mreža	Klijent/server
UPnP	Nema	Potrošačke mreže	P2P klijent/server
XMPP	Visoka – op	Upravljanje udaljenim objektima	Klijent/server
ZeroMQ	Visoka – opcionalna	CERN	P2P
WSDL	Osrednja – op	Web servisi	Klijent/server
SOAP	Osrednja – op	Potrošačke mreže	Stablo

Tabela 3. Uporedni prikaz protokola (sigurnost i upotreba) [16].

2.5 Cloud tehnologije i IoT

Ubrzani razvoj i pojavljivanje sve više IoT ekosistema je u direktnoj korelaciji sa pojavom *Cloud* tehnologija. Poslužioci se u opštem slučaju mogu podeliti po načinu na koji koriste resurse na [30]:

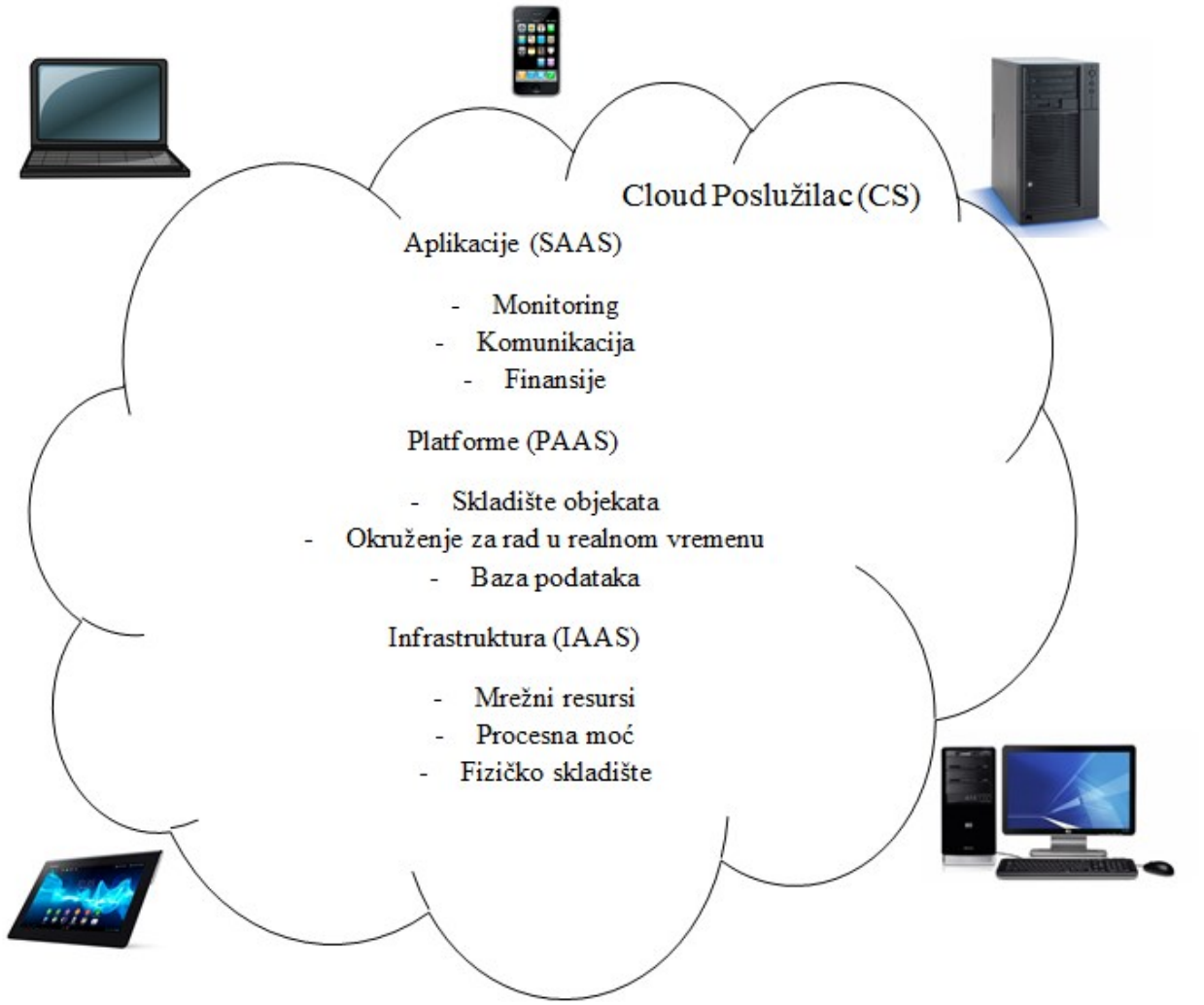
1. IoT poslužioci postavljeni u *Cloud* okruženju (*Cloud-based*),
2. IoT poslužioci postavljeni lokalno (*On Premise*).

Cloud predstavlja distribuiranje računarskih resursa preko mreže gde aplikacija može funkcionisati na više umreženih računara istovremeno. Najčešće predstavlja fizički računarski uređaj ili grupu njih, trivijalno nazivanih *Cloud* poslužilac (*Cloud Server, CS*), čiji je cilj da preko Interneta, Intraneta, WAN ili LAN mreže pruže određene usluge udaljenim korisnicima razne fizičke prirode, kao što je prikazano na Slici 7. Većina IoT poslužioca je postavljena u *Cloud* okruženju.

Pružene usluge su prikazane i mogu biti:

1. Aplikacija – CS pruža korisniku mogućnosti pokretanja aplikacija sa bilo koje fizičke lokacije bez obaveze da tu aplikaciju poseduje na svom računaru. Ovakva usluga se naziva ***Software as a Service (SaaS)***.

2. Platforma – Korisniku je omogućeno korišćenje skladišta podataka, virtuelne mašine i sličnih programsko-fizičkih resursa. Usluga se naziva i ***Platform as a Service (PaaS)***.
3. Resursi – Sa udaljene lokacije je omogućeno korišćene mrežnih adaptera, fizičkog skladišta podataka i procesne moći procesora. Ovakva upotreba CS je ***Infrastructure as a Service (IaaS)***.



Slika 7. Cloud okruženje i nivoi pruženih usluga.

Cloud tehnologija produbljuje ulogu Interneta i njegovu definiciju, nudeći komercijalne usluge širokom spektru korisnika time premostivši geografsku razdaljinu između korisnika i resursa. Ovo je omogućilo razvijanje snažnijih i zahtevnijih mrežnih okruženja i aplikacija, jer je do tada za ispitivanje, pokretanje i izvršavanje zadataka bilo potrebno posedovati resurse, kako aplikacione, tako i fizičke. *Cloud* tehnologija je doživela rapidni razvoj pojavom velikih korporacija u ovom domenu industrije – Amazon, Google, IBM, Oracle i Microsoft. Standardizovanje *Cloud* platformi dovelo je do eksponencijalnog rasta broja korisnika i pružaoca usluga.

Virtualizacijom resursa, pre svega fizičkih, *Cloud* tehnologija je pružila podstrek razvoju IoT ekosistema omogućivši obradu velike količine podataka (*Big Data*). Oni zahtevaju izdašne fizičke resurse od postavljanja (*hosting*) aplikacija (GUI) preko bazi podataka, procesne moći i mrežnih adaptera. Fizički resursi su automatski postavljali finansijsku i organizacionu prepreku pred razvoj manjih i namenskih IoT ekosistema pre pojave *Cloud* tehnologija.

Upliv *Cloud* tehnologija u razvoj IoT ekosistema je doveo do dodatnih pravila prouzrokovanih ograničenjima korišćenja *Cloud* poslužioca [31]. Sprežni slojevi za IoT ekosisteme koriste sigurnosne i proceduralne mehanizme namenjene *Cloud*-u kako bi podržali interabilnost i kompatibilnost [32]. IoT poslužioci se mogu podeliti i prema načinu pristupa u *Cloud* okruženju:

1. Privatni IoT *Cloud* poslužioci,
2. Javni IoT *Cloud* poslužioci.

Privatni IoT *Cloud* poslužioci omogućuju pristup samo unapred definisanim korisnicima formirajući privatnu mrežu. Javni IoT *Cloud* poslužioci formiraju globalnu mrežu kojoj mogu pristupiti svi korisnici. Većina javnih poslužioca koristi sistem kontrole pristupa koji ograničava pristup i prava.

2.5.1 Obrada velike količine podataka (Big Data)

Big Data predstavlja količinu podataka toliko veliku da je njeno skladištenje i obrada tradicionalnim alatima i metodama otežana. Izazovi ovakve obrade se ogledaju u prikupljanju, skladištenju, pretrazi, prenosu, analizi i prezentaciji [33]. Oni su nastali kao rezultat transformacije u načinu na koji se podaci skupljaju u mrežnom okruženju i dostavljaju poslužiocu. Tradicionalna okruženja za analizu, statistiku i vizuelizaciju obrade uz relacione baze podataka (*Relational Database Management Systems, RDMS*) se teško snalaze sa sirovim, nestruktuiranim podacima koji se nalazi u jednom bazenu (*single unstructured data pool*) [34].

Zadatak obrade velike količine podataka (*Big Data Analysis, BDA*) je ispitivanje tih podataka radi pronalaženja skrivenih podataka, do tada nepoznatih korelacija i drugih korisnih informacija. Ovakva obrada i korišćenje ovakve metodologije obrade je bilo nemoguće pre pojave *Cloud* okruženja koje je pružilo alate.

BDA dolazi do izražaja u IoT ekosistemu gde krajnji uređaji imaju mogućnost isporuke velike količine informacija IoT poslužiocima kroz D2S komunikacioni sloj [35]. Primer ovakve obrade je IoT farmerski ekosistem kompanije John Deere [36].

Ovaj IoT poslužilac je postavljen u *Cloud* okruženju (MyJohnDeere.com) i omogućuje vlasnicima teške mehanizacije racionalno i pametno raspolaganje uz maksimizaciju profita. Sistem koristi informacije sa prosečno pedeset senzora po jednom vozilu koji komuniciraju

međusobno (D2D) i sa poslužiocem (D2S). Poslužilac kombinuje te informacije sa informacijama o vremenskim prilikama, osobinama gajenih kultura i tako vrši obradu velikog niza podataka koje prezentuje korisniku kroz interaktivnu korisničku spregu. Korisnik na osnovu ovih informacija može uštedeti i do 60% sredstava u potrošnji goriva, rezervnim delovima, rodu određene kulture i vremenskom iskorišćenju mehanizacije. Za samo jednog korisnika sa dva entiteta teške mehanizacije sistem obradi i do 1 terabajt podataka u minuti, što ne bi bilo moguće sa klasičnim okruženjima.

2.5.2 Sprežni slojevi za pristup Cloud resursima

U *Cloud* okruženju resursima poslužioca se pristupa preko određenih sprežnih slojeva (*Cloud APIs*). Ovi slojevi omogućuju spoljašnjim aplikacijama pristup uslugama preko direktnih i indirektnih sprežnih slojeva [37].

1. Direktni sprežni slojevi dopuštaju klijentu u Cloud komunikaciji momentalan i direktan pristup zatraženoj usluzi bez dodatne obrade i komunikacije. U ovu grupu sprežnih slojeva spadaju i oni direktno integrisani u poslužilačku arhitekturu bez raslojavanja.
2. Indirektni sprežni slojevi uglavnom kombinuju dve ili više funkcionalnosti unutar *Cloud* poslužioca kako bi pružili uslugu korisniku.

Cloud poslužioci koriste uglavnom sledeće standarde za realizaciju sprežnih slojeva [38]:

1. Standardizovani sprežni slojevi (REST, SOAP i XML-RPC). Većina sprežnih slojeva koristi ove mehanizme za generalizovani pristup. Ovakvi sprežni slojevi su dominantni i sastavni deo svakog *Cloud*-zasnovanog poslužioca.
2. Specifični sprežni slojevi. Zavise od pružaoca i prirode usluga koje se pružaju. Uglavnom ove protokole razvija pružalac usluga. Preko ovakvih sprežnih slojeva se dopušta pristup nekim specifičnim funkcionalnostima.

Nimbits, LWM2M i ProSyst poslužioci su postavljeni u *Cloud* okruženju i koriste isključivo standardizovane sprežne slojeve. LWM2M i Nimbits poslužioci koriste samo REST u svojim sprežnim slojevima dok ProSyst koristi REST, SOAP i JSON. Krajem 2013. godine raspodela protokola korišćenih u komunikaciji preko sprežnih slojeva između *Cloud*-zasnovanih poslužilaca i korisnika je prikazan u Tabela 4.

Protokol	Procentualni udeo
REST	75%
SOAP	13%
XML-RPC	3%
Specifični sprežni slojevi	3%
JavaScript	2%

Tabela 4. Procentualni udeo protokola u implementaciji sprežnih slojeva *Cloud*-zasnovanih poslužioca [39].

Svi sprežni slojevi moraju implementirati minimum komunikacionih i sigurnosnih zahteva. Ti zahtevi se uglavnom ogledaju u korišćenju HTTPS mehanizma za prenos podataka i autentifikacioni mehanizam. Autentifikacija se obavlja ili sa parom korisnik/šifra kombinacijom ili sa identifikacionim tokenom komunikacione sesije.

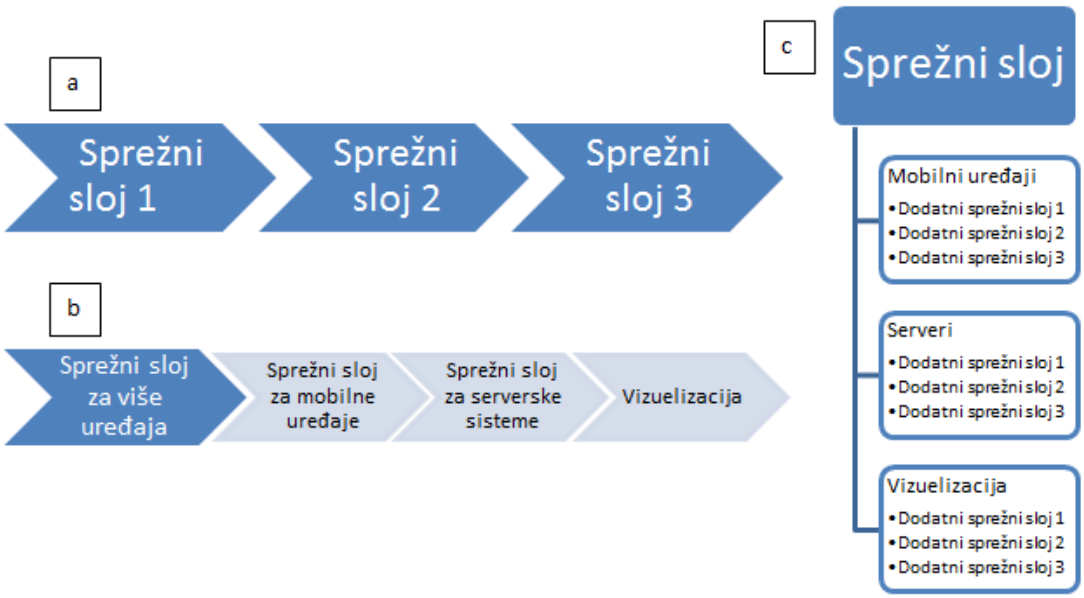
Dva pristupa projektovanju aplikativnih sprežnih slojeva *Cloud* poslužioca su namenski sprežni slojevi (*Device specific Cloud API, DCAPI*) i sprežni slojevi za različite uređaje (*Cross-platform Cloud API, CPCAPI*) [40][41].

Prvi predstavljaju raslojavanje sprežnih slojeva po nameni. Obično se ovakvi sprežni slojevi koriste kako bi se usluge raslojile po prirodi krajnjih uređaja. Na primer jedan skup usluga je dopušten mobilnim uređajima a drugi skup poslužiočkim sistemima. Prednost ovakvog pristupa je primena specifičnih sigurnosnih mehanizama i brzina.

CPCAPI pristup omogućuju postojanje samo jednog aplikativnog sprežnog sloja za sve korisnike usluga *Cloud* poslužioca. Ovakva arhitektura smanjuje kompleksnost sprežnih slojeva i omogućuje lakše održavanje i fleksibilnost na uštrb nešto slabije brzine [41].

Moderni poslužioci uglavnom koriste hibrid ove dve arhitekture gde se sprežni slojevi raslojavaju po nameni krajnjih uređaja ali se različiti nivoi prava pristupa razrešavaju u samom pristupnom sloju. Primer ovakvog hibridnog sistema je Amazon Cloud [42].

Grafički prikaz ove tri arhitekture je prikazan na Slici 8, gde je na slici a) predstavljena DCAPI arhitektura, na b) CPCAPI a na c) hibridna arhitektura.



Slika 8. Sprežni slojevi *Cloud* poslužioca.

2.6 Postojeća rešenja

Broj IoT poslužioca rapidno raste i sa razvojem tehnologija se pojavljuju rešenja raznih profila – od rešenja otvorenog koda do komercijalnih rešenja velikih korporacija Motorola i Alcatel Lucent [43]. Iako koncept IoT poslužioca u osnovi podrazumeva komunikaciju sa proizvoljnim tipom uređaja, većina poslužioca je projektovana za jedan tip ili jednu grupu krajnjih uređaja. U zavisnosti od namene, poslužioci imaju različito projektovane aplikativne sprežne slojeve.

Control4 Anywhere Access [44] je *Cloud*-zasnovani poslužilac koji nema otvoreni API koji bi omogućio integrisanje sa drugim poslužiocima i aplikacijama. Uz poslužilac dolazi integrisana web i mobilna aplikacija koja je čvrsto vezana sa jezgrom poslužioca.

Axiros Axess [45] je lokalno postavljen poslužilac koji podržava D2S komunikaciju sa proizvoljnim protokolom uz posebnu podršku za TR-069. Ono po čemu je Axess poslužilac specifičan jeste zahtev da se sve Big Data obrade moraju obavljati u okviru njegovog jezgra. Ne postoji mehanizam koji dopušta prenos velike količine podataka drugom poslužiocu.

Dimark Server [46] ima mogućnost postavljanja i u *Cloud* okruženje i lokalno. Razvijen je u Java EE okruženju i specifični aplikativni slojevi zahtevaju Java klijentsku aplikaciju.

ProSyst [10] poslužilac je namenski poslužilac otvorenog koda za praćenje parametara na krajnjem uređaju bez obzira na D2S protokol. Omogućuje jednostavnu grafičku prezentaciju na ugrađenoj aplikaciji uz ograničen skup usluga.

Nimbits [11] je poslužilac otvorenog koda namenjen uređaji male snage – senzorima i elektromehaničkim aktuatorima. Može se koristiti kao poslužilac zasnovan na Cloud-u ali može biti i lokalno postavljen. Sličnu funkcionalnost i arhitekturu ima i *LWM2M* rešenje.

ThinkSpeak [47] je opšti poslužilac otvorenog tipa koji omogućuje integrisanu grafičku korisničku spregu. Prednost mu je što postoji u varijantama realizovanim u različitim programskim jezicima.

Uporedni prikaz postojećih rešenja IoT poslužioca sa njihovim osobinama je prikazan u Tabela 5.

Poslužioc	Namena poslužioca	Cloud/Lokalni	Arhitektura API	Protokoli u API/Specifični (spec) API	Podrška za Big Data obradu
Control4 Anywhere Access	Opšti	Cloud	CPCAPI	Spec	Ne
Axiros Axess	Opšti	Lokalni	DCAPI	SOAP, XML RPC, REST	Ne kroz API
Dimark Server	Opšti	Cloud/Lokalni	CPCAPI	XML, SOAP	Ne
ProSyst	Komandni	Cloud	Hibridna	SOAP, REST, JSON	Ne
ThinkSpeak	Opšti	Lokalni	Hibridna	REST	Ne
Nimbits/LWM2M	Prezentacioni	Cloud	CPCAPI	REST	Ne

Tabela 5. Uporedni prikaz postojećih IoT poslužioca.

Poslužioci orijentisani na komercijalne multimedijalne mreže (digitalna televizija, VoIP, mobilna telefonija) imaju jedinstven i integrisan OSS/BSS sistem (*Axiros Axess*) ili omogućuju konekciju aplikacije pružaoca usluga koja obavlja te zadatke. OSS/BSS sistemi omogućuju pružaocu usluga da void računa o kvalitetu usluge i naplati u svojoj mreži. Za uspešnu konekciju spoljašnjeg OSS/BSS sistema, poslužilac mora da pruži funkcionalnosti kao što su npr. grupisanje uređaja i upravljanje korisničkim nalozima.

Od sigurnosnih mehanizama, svi poslužioci pružaju podršku za HTTPS komunikaciju i unutrašnju kontrolu pristupa ili podršku za povezivanje sa spoljašnjim autorizacionim modulima.

Podrška za komunikaciju sa drugim poslužiocima je ograničena (*Axiros Axess*) ili potpuno onemogućena (*Control4*) a obrada Big Data podataka je pružena isključivo u okviru samih poslužioca bez podrške za izmeštanje takve obrade u namenske poslužioce čime bi se dobilo na iskorišćenju resursa i skalabilnosti.

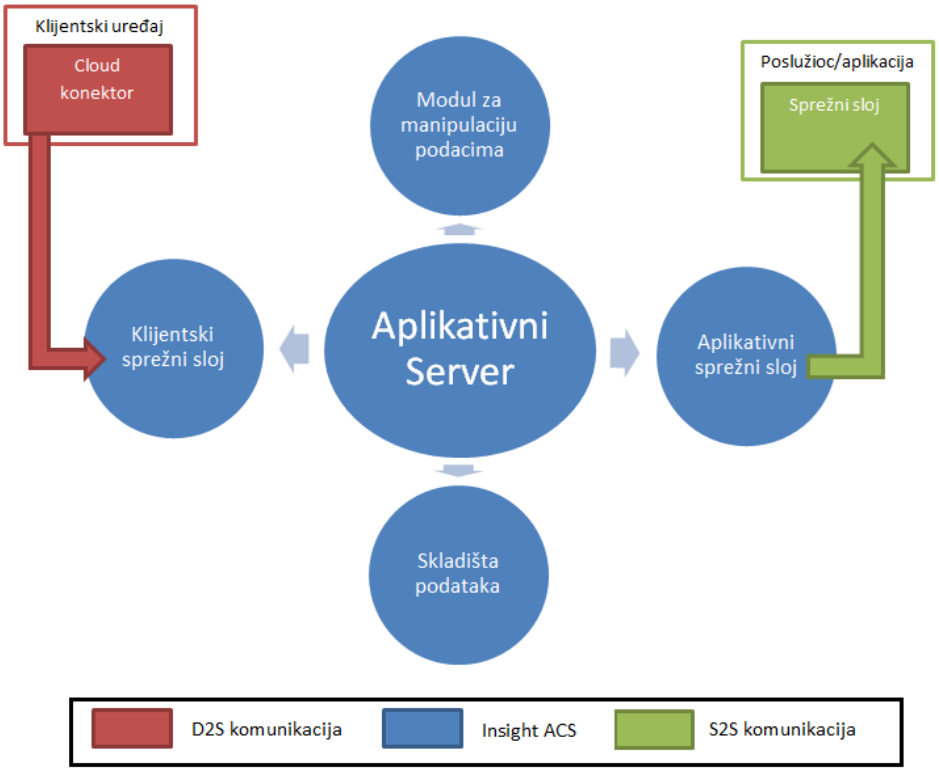
Najveći problem u IoT poslužiocima je heterogenost korišćenih aplikativnih ISO OSI protokola. IoT poslužioci uglavnom ne omogućuju pristup aplikacijama preko optimizovanih sprežnih slojeva čime se gubi vreme i dodatno opterećuju resursi. Uz to, koliko je autoru rada poznato, iako IoT poslužioci podržavaju Big Data obradu, oni ne omogućuju izmeštanje njene obrade u namenski poslužilac preko aplikativnih sprežnih slojeva. Aplikativni sprežni slojevi su često čvrsto uvezani sa programskog podrškom poslužioca i mogu se isključivo koristiti sa datim poslužiocem.

Pregledom i analizom postojećih rešenja utvrđeno je da bi sa stanovišta performansi, funkcionalnosti i integracije, najbolje rešenje za IoT poslužilac bilo implementiranje slojevitog aplikativnog sprežnog sloja koji će omogućiti optimizovanu razmenu podataka u zavisnosti od tipa uređaja. Ovakva arhitektura sprežnog sloja omogućuje korišćenje različitih protokola i prilagođavanje kontrole pristupa. Kako bi poslužilac mogu biti uspešno postavljen u *Cloud* okruženje, za API je najbolje pružiti i standardizovane mehanizme i razrađene protokole (npr. REST) ali i obezbediti specifična sredstva komunikacije (WSDL, GWT RPC).

Ovakav sprežni sloj bi omogućio isporuku Big Data podataka drugim poslužiocima čime bi se matični poslužilac rasteretio i što bi dovelo do boljih perfomansi i bolje iskorišćenosti resursa. U cilju lakšeg održavanja i mogućnosti ponovnog korišćenja (*re-use*) programskog rešenja, aplikativni sprežni sloj je nezavisno implementiran od jezgra poslužioca, što omogućava lakše korišćenje API sloja u slučaju da se neke od funkcionalnosti jegra poslužioca ili D2S sloj promene. Ovakva implementacija unosi vremensku zadršku u odnosu na poslužioce sa ugrađenim pristupnim slojevima.

3. Analiza rešenja

Insight ACS je opšti poslužilac za IoT ekosisteme postavljen u *Cloud* okruženju. Arhitektura visokog nivoa je prikazana na Slici 9.



Slika 9. Arhitektura visokog nivoa Insight ACS-a

Za D2S komunikaciju koristi integrisani *Cloud* konektor u klijentskom uređaju. Klijentski, krajnji uređaji mogu biti različite prirode. U jezgru sistema je aplikativni server, koji omogućuje skladištenje, manipulaciju i tok podataka i pruža okruženje za rad u realnom vremenu za ostale

module. Za komunikaciju sa *Cloud* konektorom zadužen je klijentski sprežni sloj za komunikaciju sa uređajima. Poslužilac raspolaže i distribuira podatke preko modula za manipulaciju podacima. Insight ACS poseduje dualno skladište podataka, što znači da je sistemu dostupna i baza podataka i brza skrivena (*cache*) memorija. S2S aplikacija se ostvaruje preko aplikativnog sprežnog sloja.

D2S komunikacioni sloj je realizovan korišćenjem bidirekcionalnog SOAP/HTTP-zasnovanog protokola – TR-069. Predviđa komunikaciju između *korisničke opreme (Customer Premises Equipment, CPE)* i *automatskih konfiguracionih poslužioca (Auto-Configuration Server)*. Protokol pruža podršku raznim parametrima u zavisnosti od tipa uređaja kroz koncept modela podataka (*data model*), koji predstavlja XML-zasnovano stablo parametara. Za određene tipove uređaja postoje propisani modeli podataka, dok je ostavljena mogućnost korisnicima da sami definišu svoj model. Ovaj protokol se u IoT koristi kod robusnih uređaja sa vremenskim zahtevima visoke tolerancije (*High tolerance requirements*), što podrazumeva da D2S komunikacija preko njega po pravilu ne mora da ispunjava opšte zahteve u vremenskom odzivu ispod 10 ms.

Ovaj protokol se preporučuje za upotrebu u nadgledanju i upravljanju mrežama sastavljenih od STB uređaja, dok se u poslednje vreme sve više upotrebljava kao primarni izbor za kućne mrežne uređaje (ruteri i pametne kuće). Ovaj protokol se ne koristi intenzivno u D2D i S2S komunikacionim slojevima.

Insight ACS podržava uređaj proizvoljnog modela podataka čijim je parametrima moguće pristupiti preko aplikativnog sprežnog sloja.

TR-069 protokol preporučuje osnovnu funkcionalnost – komunikaciju i skladištenje, u poslužiocu dok bi se sva vizuelizacija i dodatna obrada izmestila u posebne poslužioce i aplikacije preko NBI aplikativnog sloja (*Northbound Interface, NBI*), koji predstavlja S2S komunikacioni sloj.

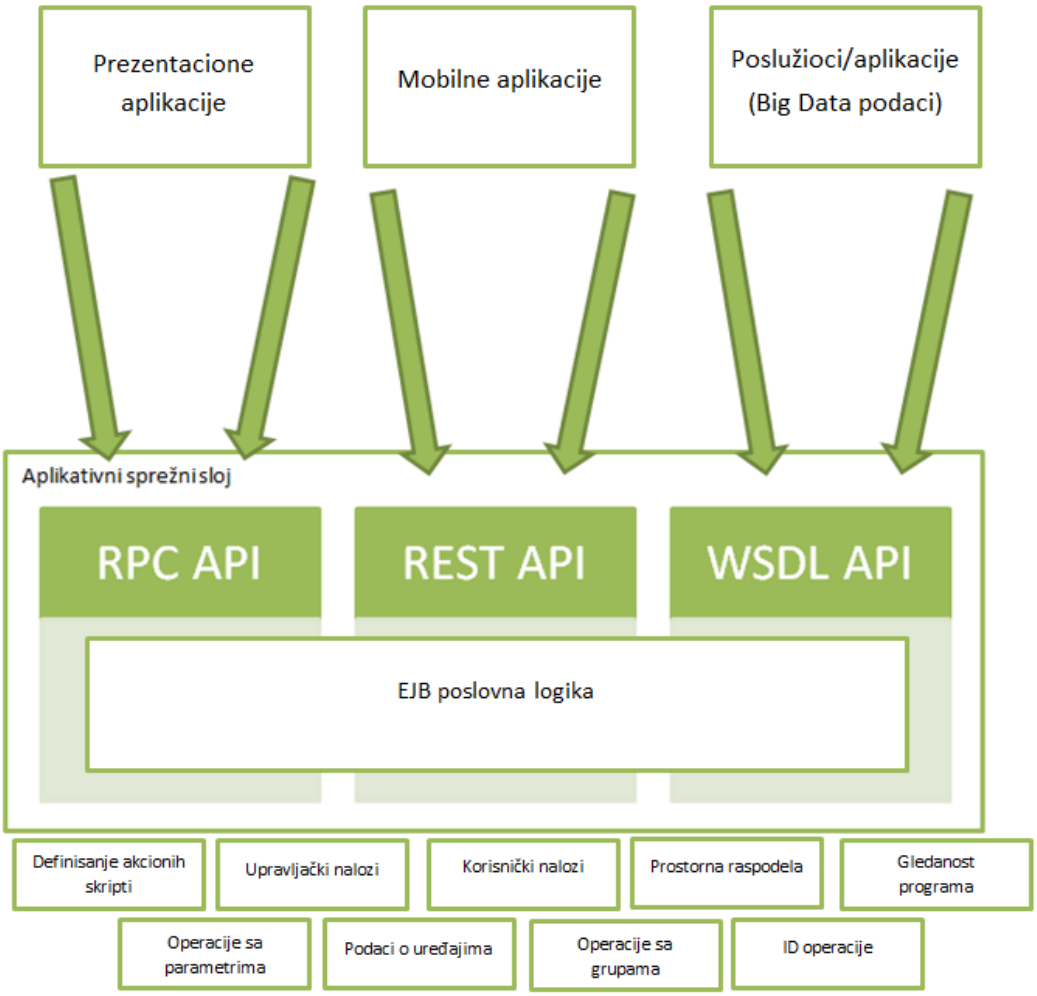
3.1 Uloga aplikativnog sprežnog sloja u Insight ACS rešenju

Zadatak aplikativnih sprežnih slojeva u IoT poslužiocima je da informacije dostave spoljašnjim aplikacijama ili drugim poslužiocima što brže i što bezbednije pritom ne narušavajući druge gradivne blokove. Aplikativni sprežni slojevi (API) specificiraju način na koji programske komponente komuniciraju među sobom. Insight ACS ima za zadatak da svoje podatke deli sa sledećim klijentima preko S2S sloja:

1. Mobilnim uređajima (mobilni telefoni, tablet računari),
2. Drugim poslužiocima i
3. Web aplikacijama.

U skladu sa zahtevom, izbor arhitekture sprežnog sloja u ovom radu je trodelna DCAPI (*Device specific Cloud API, DCAPI*) arhitekture, koja je u skladu sa praksom u *Cloud* okruženju. Mobilni uređaji trebaju biti u mogućnosti da pristupaju često poslužiocu ali za manjom količinom podataka. Ti podaci se tipično tiču vrednosti parametara i njihove geografske lokacije. Za ovakav modul sprežnog sloja je izabran REST/JSON mehanizam. Drugi poslužioci pristupaju retko Insight ACS poslužiocu ali sa zahtevom sa veliku količinu podataka. Izbor za ovakav sloj predstavlja web servis definisan preko WSDL jezika koji koristi SOAP format poruka. Grafička korisnička sprega zahteva širi spektar podataka i zahteva brz odziv API sloja Insight ACS-a. Odabir protokola za ovaj modul je GWT RPC mehanizam koji koristi JSON poruke.

Različiti korisnici ovog sprežnog sloja imaju drugačije specifičnosti u pogledu brzine, korišćenih protokola i bezbednosno-autentikacionih mehanizama zbog čega je DCAPI arhitektura u ovom slučaju bolje rešenje od CPCAPI arhitekture koja bi dodatno opteretila osnovnu funkcionalnost poslužioca. Gradivni blokovi aplikativnog sprežnog sloja su prikazani na Slici 10. Aplikativni sprežni sloj predstavlja NBI sprežni sloj poslužioca zasnovanog na TR-069 protokolu.



Slika 10. Slojevi aplikativnog sprežnog sloja.

Koristeći različite bazne tehnologije – GWT RPC, REST sa JSON protokolom i WSDL jezik za web servise, poslužilac u sprežnim slojevima postiže fleksibilnost u isporuci podataka, uz ispunjavanje funkcionalnih i vremenskih zahteva. Klijenti u ovakvom okruženju mogu biti aplikacije za vizuelizaciju koje zahtevaju mahom manju količinu podataka u striktnom vremenskom okviru ali i drugi poslužioci kojima je vremenski okvir širi ali je zahtevana količina podataka daleko veća.

Insight ACS podržava isporuku proizvoljnih parametara iz modela podataka ali uključuje i podršku za obradu podataka vezanih za multimedijalne uređaje. Zbog prirode protokola koji se koristi u D2S komunikaciji, poslužilac podržava i isporuku velike količine nestruktuiranih podataka (*Big Data*) drugim poslužiocima na post-procesiranje.

3.2 RPC API modul

RPC modul aplikativnog sprežnog sloja je projektovan kao specifični sprežni sloj namenjen komunikaciji logike IoT poslužioca i web aplikacije za grafičko okruženje. RPC modul koristi GWT (*Google Web Toolkit*) okruženje.

GWT predstavlja Java okruženje za razvoj asinhronih JavaScript aplikacija. Omogućuje kreiranje web aplikacija korišćenjem Java programskog jezika kojeg okruženje prevodi u optimizovani JavaScript kod koji se izvršava u web pregledaču (*Google Chrome, Mozilla Firefox, Internet Explorer* itd).

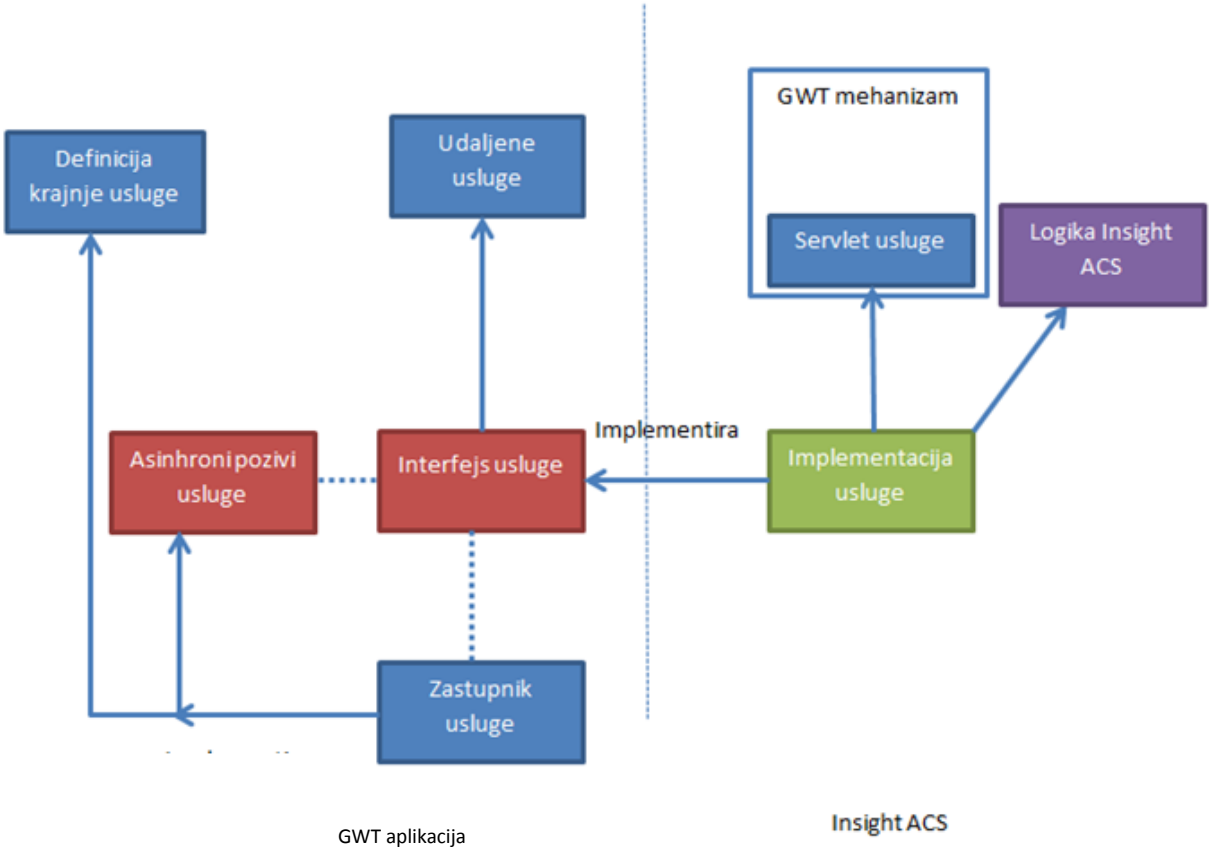
RPC modul je realizovan korišćenjem GWT mehanizma RPC HTTP poziva koji omogućuju transparentne pozive ka poslovnoj logici poslužioca. Na strani Insight ACS poslužioca se nalaze automatski kreirani *Java Servleti* koji omogućuju komunikaciju i implementirane usluge (*Service*) koje se omogućuju kroz pristupni sloj. GWT mehanizam preuzima realizaciju i prenos protokolima ISO OSI nivoa sesije (*low level*) – JSON i HTTP, dok se aplikaciji omogućuje da uslugama pristupa na visokom nivou (*high level*) preko objekata u Java jeziku. GWT RPC mehanizam Java objekte serijalizuje kako bi se brzo i bezbedno preneli do klijentske aplikacije.

Ova komunikacija je *server/klijent* arhitekture što podrazumeva da se usluge dobijaju iniciranjem isključivo od strane grafičke aplikacije. GWT RPC mehanizam dozvoljava raslojavanje operacija po različitim modulima, uslugama (*decoupling*), na taj način omogućujući lakše održavanje slojeva ali i balansiranje opterećenja.

Prednosti korišćenja ovakvog mehanizma se ogledaju u olakšanoj komunikaciji i kreiranju GUI aplikacija s obzirom da sva obrada i dobavljanje podataka sa poslužioca se može realizovati u programskom jeziku Java. GWT RPC mehanizam ne definiše način kreiranja grafičkog

okruženja već samo način komuniciranja između poslužioca i web aplikacije, koja može biti kreirana i GWT alatima ali i drugim alatima i programskim jezicima (*JavaScript*).

Primer fizičke realizacije ovog aplikativnog sloja je prikazan na Slici 11.



Slika 11. Mehanizam GWT RPC poziva.

Sa strane GWT aplikacije se nalaze neophodni moduli u klijentskoj komunikaciji dok se sa InsightACS strane nalaze delovi aplikativnog sloja poslužioca. Plavom bojom je prikazan deo GWT mehanizma ili ono što on generiše. Zelenom bojom je prikazan implementirani deo aplikativnog sloja, dok je crvenom bojom prikazan deo koji klijentska aplikacija mora da implementira kako bi koristila aplikativni sloj.

Korišćenje GWT mehanizma dozvoljava raslojavanje usluge omogućenih u ovom modulu sprežnog sloja, pa su tako ponuđene sledeće usluge sa uključenim operacijama:

1. Konfiguracione usluge (*ConfigurationService*),
    a. Operacije sa akcionim skriptama,
    b. Operacije sa podacima sa poslužioca,
    c. Operacije sa upravljačkim nalozima.
2. Usluge za upravljanje uređajima (*CpeManagementService*),
    a. Operacije sa identifikacionim podacima uređaja,

b. Operacije sa parametrima uređaja,
c. Operacije sa grupama,
d. Operacije sa korisničkim nalozima.

3. Analitičke usluge (*StatsService*),
    a. Operacije sa geografskom lokacijom uređaja.

Svaka usluga je ponuđena kao posebna pristupna tačka za klijentsku aplikaciju. U okviru RPC modula se obavlja kontrola pristupa nad pojedinačnim operacijama čime se štiti integritet podataka u logici Insight ACS poslužioca. RPC modul koristi HTTP protokol sa SSL/TLS mehanizmom.

3.3 REST API modul

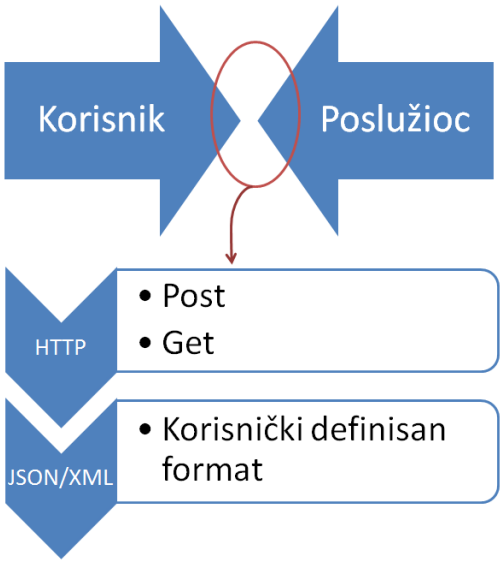
REST modul aplikativnog sprežnog sloja predstavlja specifični sprežni sloj namenjen širokom spektru mobilnih uređaja. REST (*Representational State Transfer*) predstavlja arhitekturu programske podrške u razmeni podataka preko Interneta. Ovakva arhitektura nije zadužena za fizičku realizaciju razmene podataka (protokol) a najčešće se koristi JSON format iako se u zadnje vreme popularizuje i XML/SOAP format poruka. REST propisuje niz pravila i ograničenja i on ukratko koristi metode HTTP protokola kako bi na njih mapirao CRUD (*create retrieve, update, delete*) operacije.

REST arhitektura je definisana nizom komponenti, poveznika i ograničenja podataka. Tako definisana arhitektura obezbeđuje sledeće osobine:

- Visoke performanse – intentzivna komunikacija je veoma bitna u mrežnim sistemima.
- Skalabilnost – mogućnost da podrži dosta entiteta koji međusobno komuniciraju u jednom sistemu.
- Jednostavnost korisničke sprege.
- Dinamičnost – mogućnost brze promene čak i dok sistem radi.
- Portabilnost – moguće je ovako realizovane sprežne slojeve preneti sa uređaja na uređaj.
- Pouzdanost – zbog svoje jednostavnosti otporan je na otkaze u sistemskom nivou.

Ove osobine se postižu implementiranjem propisanih pravila u komunikacija. Ta pravila su da je komunikacija *klijent/server* arhitekture što podrazumeva da klijenti nisu zaduženi za kontrolisanje podataka unutar servera već na svaki poziv dobijaju odgovor. Ovo omogućuje da se poslužioci i korisnici usluga mogu razvijati nezavisno. REST nameće komunikaciju bez stanja sesije (*stateless*). Na poslužiocu se ne čuva kontekst komunikacije ili bilo kakve informacije o zahtevima sa korisnika. Mane korišćenja ovakvih mehanizma su nefleksibilnost prema velikoj

količini podataka i nepostajanje standardizovanog mehanizma koji bi omogućio komunikacione sesije. Primer razmene poruka korišćenjem REST arhitekture je prikazan na Slici 12.



Slika 12. Razmena podataka u REST arhitekturi.

Korisnički aplikativni slojevi i web servisi koji poštuju REST arhitekturu se nazivaju RESTful. RESTful slojevi se karakterišu po sledećem:

1. Njihovim uslugama je moguće pristupiti preko univerzalnog identifikatora (*Universal Resource Identifier*).
2. Koriste podatke pogodne za prenos preko internet – JSON i XML strukture, slike.
3. Koriste standardne HTTP metode – GET, PUT, POST i DELETE.

Ukoliko korisnik recimo želi da dobije listu studenata on će proslediti HTTP GET poruku na http://fakultet.org/studenti, dok recimo kreiranje određenog studenta je moguće sa HTTP POST porukom na http://fakultet.org/studenti/SasaRadovanovic.

Insight ACS kroz REST API modul koristeći poruke JSON formata omogućuje komunikaciju pogodnu za mobilne aplikacije otkrivajući manji skup specifičnih operacija kroz manje zahtevnu (*lightweight*) komunikaciju. Sigurnosni mehanizam se sastoji od HTTPS protokola i kontrole pristupa kroz pojedinačne operacije. REST API omogućava lako održavanje i brzu promenu i zbog toga je namenjen komunikaciji sa većim brojem krajnjih uređaja nego RPC API modul.

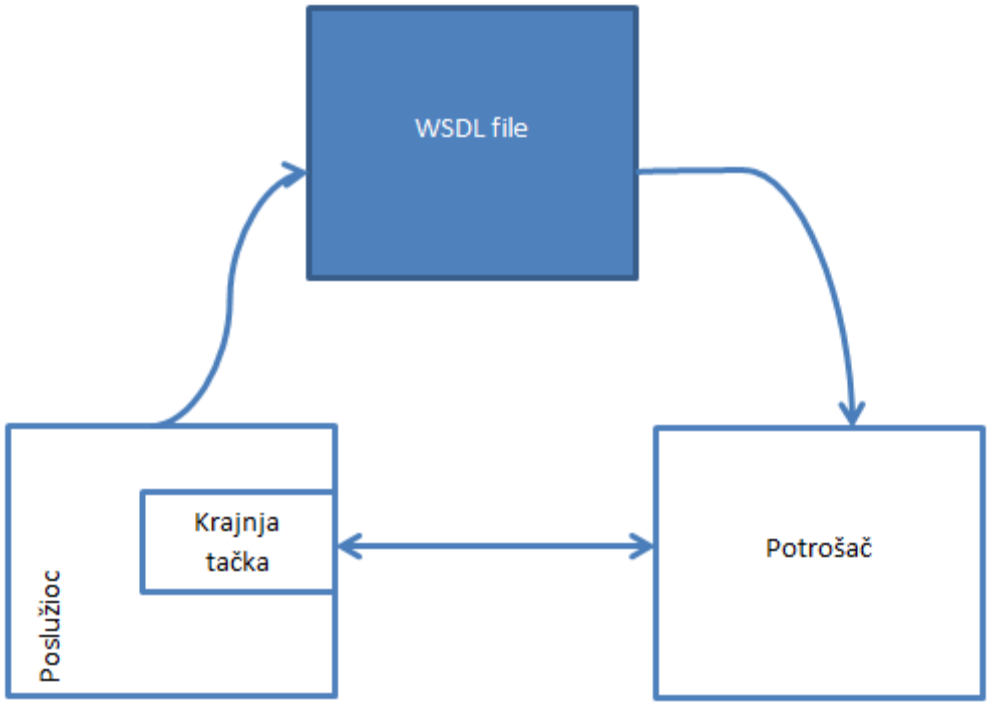
3.4 WSDL API modul

WSDL API predstavlja komunikacioni modul aplikativnog sprežnog sloja namenjen za komunikaciju sa robusnijim korisničkim aplikacijama i drugim poslužiocima. Ovaj modul je zasnovan na WSDL (*Web Service Description Language*) opisnom jeziku i otkriva pomoću njega određene funkcionalnosti poslužioca kao tipične web usluge. Ovaj mehanizam je u

aplikativnom sloju implementiran koristeći JAX-WS (*Java API for XML Web Services*) mehanizam.

WSDL-zasnovane web usluge funkcionišu tako što poslužilac svoje funkcionalnosti otkriva kroz XML datoteku (*WSDL file*) koja u sebi nosi definiciju i opis web usluga . Ova datoteka je kako ljudski, tako i mašinski čitljiva i na osnovu ograničenja i informacija u njoj se odvija komunikacija između poslužioca i korisnika, koji se naziva i potrošač. Potrošač na osnovu WSDL datoteke generiše klijentski kod, koji razrešava fizički deo komunikacije. WSDL datoteka sve usluge koje poslužilac otkriva predstavlja kao *mrežne krajnje tačke* (*network endpoints*). Razmena poruka između entiteta u ovoj komunikaciji je prikazana na Slici 13.

Korišćeni ISO OSI protokol aplikativnog nivoa može biti proizvoljan a u slučaju Insight ACS poslužioca se koriste SOAP poruke. Protokol definiše pružaoc usluga. Ovakav mehanizam je povoljan za razmenu veće količine podataka manjem broju korisnika i uglavnom je namenjen komunikaciji između Insight ACS i drugih poslužioca.



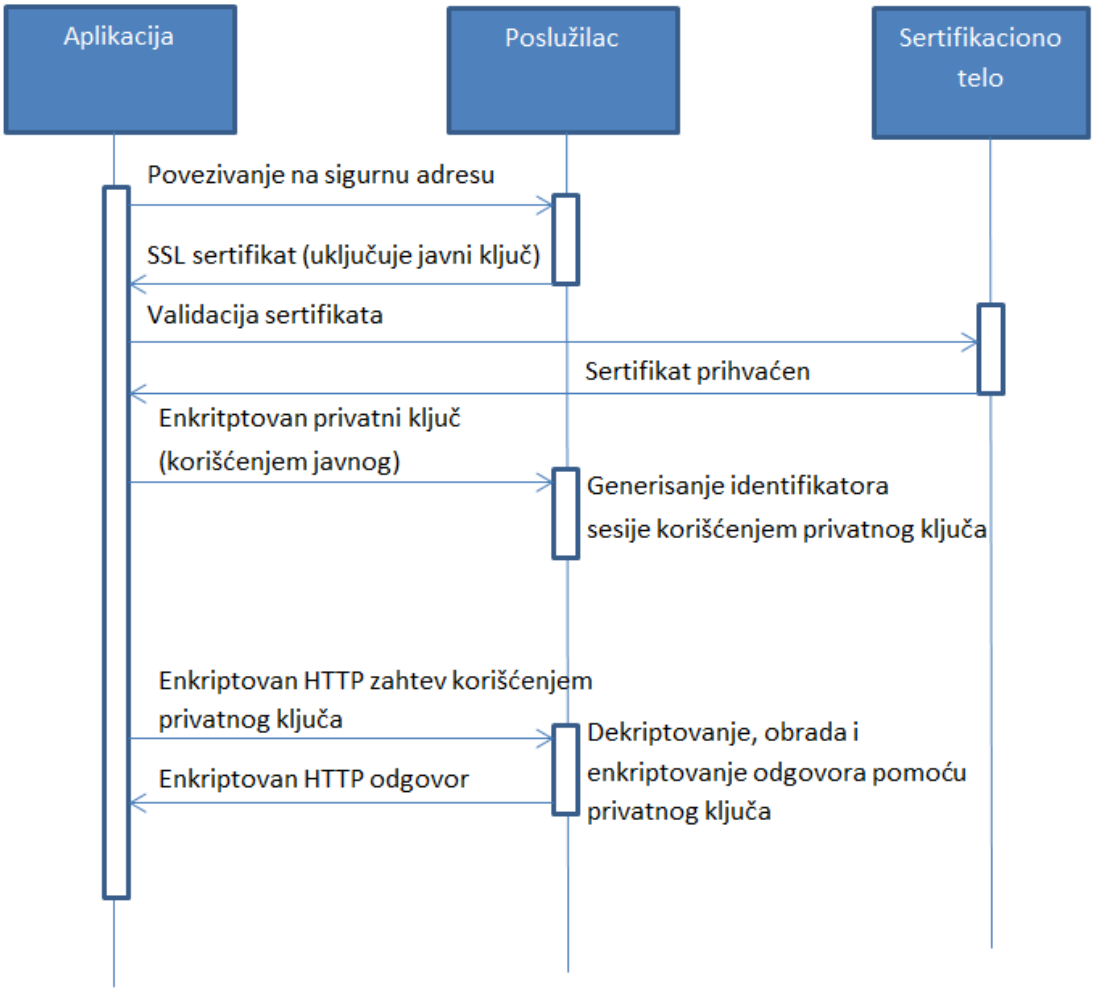
Slika 13. WSDL komunikacija.

Ovakva komunikacija unosi dosta viška podataka u komunikaciji i nije fleksibilna na promene u toku rada, pa se ovaj modul sprežnog sloja koristi za prenošenje Big Data podataka na dalje procesiranje. Sigurnosni mehanizmi se sastoje od kontrole pristupa i HTTPS protokola.

3.5 Bezbednosni mehanizmi

U okviru sva tri modula se obavlja obavezna kontrola pristupa. Kontrola pristupa podrazumeva ispitivanje identiteta pozivaoca - korisnika metoda u sprežnim slojevima. Svi *Cloud*-zasnovani sistemi moraju integrisati dodatne bezbednosne mehanizme sem uobičajene zaštite transportnih nivoa protokola. Sprežni slojevi Insight ACS poslužioca koriste *HTTPS* u svim modulima.

HTTPS predstavlja komunikacioni protokol ISO OSI nivoa sesije koji se dobija kombinovanjem *HTTP* protokola sa *SSL/TLS* enkripcionim protokolom [48]. *HTTPS* se koristi za zaštitu komunikacije preko otvorenih mreža gde štiti od prisluškivanja komunikacije i presretanja poruka (*Man-in-the-middle attacks, MITM*). SSL je protokol ISO OSI nivoa sesije koji zahteva korišćenje sertifikata koji izdaju ovlašćena sertifikaciona tela (*Certification Authority, CA*). Sertifikati se koriste kako bi entiteti u komunikaciji generisali enkripcione i dekripcione ključeve kojima se poruke šifruju i dešifruju [49]. Tok komunikacije korišćenjem *HTTPS* protokola je prikazan na Slici 14.



Slika 14. Komunikacije korišćenjem HTTPS protokola.

Dodatni nivo zaštite predstavlja kontrola pristupa. Kontrola pristupa je distribuirana što podrazumeva da se ne obavlja u jedinstvenoj tački pristupa već da svaki modul obavlja svoju kontrolu pristupa. Prednosti ovakvog modela su lakše promene, bolja kontrola pristupa na nivou pojedinačnih metoda, efikasnost kontrole pristupa u odnosu na korišćene metode u modulima. Mane se uglavnom ogledaju u kompleksnosti i održavanju. Ukoliko se pojavi potreba za novim funkcionalnostima u sva tri sloja to podrazumeva prilagođavanje sva tri modula i njihovih kontrola pristupa.

Kontrola pristupa preko *HTTP* i *HTTPS* protokola je moguća korišćenjem jednog od dva mehanizma:

1. Osnovna kontrola pristupa (*Basic authentication*) i
2. Sistemska kontrola pristupa (*Digest authentication*).

Razlika u funkcionisanju između te dve je način na koji se korisnici usluga predstavljaju svojim korisničkim imenom i lozinkom onome ko te usluge pruža. Osnovna kontrola pristupa podrazumeva da aplikacija korisničko ime i lozinku prenese uz *HTTP* zahtev u vidu neenkriptovanog teksta u okviru *HTTP* poglavlja. Ovakva kontrola pristupa se koristi isključivo uz *HTTPS* protokol i to predstavlja manu. Prednosti ovog mehanizma su jednostavna implementacija, manje vremene se gubi u komunikaciji i korisnička aplikacija je brža.

Sistemska kontrola pristupa zahteva dodatnu razmenu poruka jer aplikacija prvo mora poslati zahtev poslužiocu, koji na to odgovora sa specijalnim kodom. Sa tim kodom, korisničkim imenom i načinom enkripcije, aplikacije se javljaju poslužiocu i tek onda kreće razmena podataka. Prednosti sistemske kontrole pristupe se ogledaju u enkriptovanju jer za razliku od osnovne, korisničko ime i lozinka nikada se ne šalju u neenkriptovanom obliku. Mane se ogledaju u višku komunikacije. Ukoliko je moguće garantovati korišćenje SSL sertifikata sa *HTTP* protokolom, osnovna komunikacija predstavlja efikasniji mehanizam i biće izbor za rešenje ovog rada.

U okviru modula aplikativnog sprežnog sloja, postoji i mehanizam distribucije resursa korišćenjem ugrađenih alata iz razvojnog okruženja. Ovaj mehanizam podrazumeva da posle kontrole pristupa na nivou protokola sesije postoji mehanizam kontrole pristupa u samom sloju [50]. Određene operacije su dozvoljene samo određenom tipu aplikacija pa su tako resursi sem kontrole pristupa na nivou HTTPS protokola, zaštićeni i korišćenjem unutrašnje distribucije resursa.

3.6 Operacije u aplikativnom sprežnom sloju

Insight ACS omogućuje čuvanje podataka, standardizovanu komunikaciju sa krajnjim uređajima i određene pretprocesne funkcionalnosti. Kao IoT poslužilac opšteg tipa, omogućuje

razmenu podataka sa drugim poslužiocima i aplikacijama. Funkcionalnosti kojima se može pristupiti kroz sprežni sloj poslužioca mogu se podeliti u nekoliko grupa:

- Operacije sa identifikacionim podacima uređaja,
- Operacije sa parametrima uređaja,
- Operacije sa grupama,
- Operacije za upravljanje korisničkim nalozima i
- Specifične operacije.

3.7 Operacije sa identifikacionim podacima uređaja

IoT koncept podrazumeva postojanje jedinstvenog identifikatora krajnjih uređaja. U D2S komunikaciji, u većini slučajeva to je IP adresa ili varijacije u zavisnosti od korišćenog protokola. U S2S komunikaciji nije predviđen jedinstveni mehanizam za identifikovanje uređaja i on se razlikuje od sistema do sistema. Operacije između poslužioca i aplikacija zasnovane na razmeni podataka vezanih za identifikacione podatke krajnjih uređaja se nazivaju ID-zasnovane operacije (*ID-based operations, IDO*).

Identifikator krajnjeg uređaja je osnovni parametar svakog uređaja registrovanog u IoT poslužiocu preko kojeg se pristupa njegovim parametrima i funkcionalnostima. Odabir identifikatora zavisi od nekoliko komponenti. D2S protokol utiče jer na osnovu njega poslužilac mora identifikovati svaki uređaj pojedinačno. Priroda poslužioca formira način na koji se sa podacima manipuliše u sistemu.

Insight ACS poslužilac kao jedinstveni identifikator uređaja koristi identifikacionu strukturu sastavljenu od parametara tako da jedinstveno definišu uređaj u svim slojevima komunikacije. Identifikaciona struktura je sastavljena od sledećih parametara:

1. Korisnički definisan identifikacioni parametar,
2. Ugrađeni serijski broj,
3. Klasa uređaja i
4. Proizvođač.

Aplikativni sprežni slojevi Insight ACS-a moraju omogućiti operacije sa identifikacionom strukturom jednog uređaja i operacije nad nizovima (listama) uređaja u sistemu. Sledeći podaci vezani za identifikacione strukture su omogućeni u sprežnim slojevima:

- Dobavljanje liste uređaja.
    - Dobavljenje liste uređaja po proizvoljnom kriterijumu,
    - Dobavljanje liste aktivnih uređaja.
- Dobavljanje broja uređaja na poslužiocu.
    - Dobavljanje broja aktivnih uređaja,

    - Dobavljanje broja uređaja po proizvoljnom kriterijumu.
- Upisivanje novog korisnički definisanog identifikacionog parametra.
- Uklanjanje jednog ili više uređaja iz poslužioca.
- Pronalaženje uređaja na osnovu identifikacione strukture.

3.8 Operacije sa parametrima uređaja

Svaki parametar uređaja u poslužiocu je predstavljen parom *ime parametra-vrednost* (*parameter name-value*). U zavisnosti od implementacije, mogu se nazivati raznim imenima, a najčešća su parametri (*parameters*), atributi (*attributes*) i osobine (*properties*), pa se tako ovakve operacije nazivaju *property-based operations*, *PO*. Operacije sa parametrima u Insight ACS sistemu su oslonjene na prirodu D2S protokola, TR-069. Protokol predviđa postojanje modela podataka vezanog za svaki uređaj koji virtualizuje njegovu prirodu i funkcionalnost. Ovaj model podataka, realizovan kao stablo sastavljeno od parametara, objekata i atributa je ono što karakteriše uređaj u Insight ACS poslužiocu.

Rad sa parametrima preko sprežnih slojeva u Insight ACS poslužiocu zahteva i određene ulazne parametre koji su najčešće identifikaciona struktura uređaja i putanja u stablu modela podataka koja refencira parametar ili grupu parametara. Parametar se u poslužiocu sastoji od para - ime i vrednost.

Operacije zasnovane na parametrima koje sprežni slojevi Insight ACS poslužioca omogućavaju su sledeće:

- Dobavljanje parametara određenog uređaja.
    - Dobavljanje svih parametara,
    - Dobavljanje parametara u zavisnosti od određenog kriterijum.
- Dobavljanje određenog parametra na određenom uređaju.
- Dobavljanje, pravljenje i brisanje objekata u modelu podataka.
- Dobavljanje i izmenu atributa parametara.
- Dobavljenje istorije promene vrednosti parametra.
    - Po broju unosu,
    - Po vremenu.
- Postavljanje nove vrednosti.
    - Postavljenje vrednosti jednog parametara na svim uređajima,
    - Postavljanje vrednosti jednog parametra na jednom uređaju,
    - Postavljanje vrednosti više parametara na jednom uređaju,
    - Postavljanje vrednosti više parametara na više uređaja.

3.9 Operacije sa grupama

IoT poslužioci mogu posluživati i do nekoliko miliona krajnjih uređaja, pa zbog toga je grupisanje uređaja važna funkcionalnost koju podržavaju. Grupisanje (*grouping*) omogućuje da se određene funkcije i operacije izvršavaju nad već definisanom grupom i tako smanjuje vreme obrade i opterećenje na poslužilac i komunikaciju.

Sortiranje uređaja po grupama je dozvoljeno isključivo iz drugog poslužioca ili spoljašnje aplikacije i najčešće se koristi iz grafičkih sprega. Može biti n-n, 1-n i 1-1 u zavisnosti od unutrašnjih ograničenja u relacijama uređaja i grupa.

Grupisanje u Insight ACS je dopušteno po n-n principu što znači da jedan uređaj se može nalaziti u više grupa i jedna grupa može sadržati više uređaja. Grupa se odlikuje imenom i nizom uređaja asociranim sa njom.

Sprežni slojevi Insight ACS omogućuju sledeće operacije sa grupama:

- Dodavanje i brisanje grupa.
    - Dodavanje i brisanje jedne grupe,
    - Dodavanje i brisanje više grupa.
- Promena imena grupe.
- Dobavljanje svih uređaja u grupi.
- Dobavljanja svih grupa asociranih sa datim uređajem.
- Dobavljanje svih grupa prema parcijalnom imenu.
- Dodavanje i brisanje uređaja iz grupe.
    - Dodavanje i brisanje jednog uređaja u/iz jedne grupe,
    - Dodavanje i brisanje jednog uređaja u/iz više grupa,
    - Dodavanje i brisanje više uređaja u/iz jedne grupe.

3.10 Operacija za upravljanje korisničkim nalozima

Svi poslužioci okrenuti ka komercijalnim mrežama zasnovanih na isporuci sadržaja moraju podržati rad sa korisničkim nalozima u većoj ili manjoj meri. Korisnik u sistemu je vlasnik korisničkog naloga odnosno krajnjeg uređaja (*Subscriber*). Količina informacija i način manipulacije sa njima zavisi od uloge poslužioca u mreži.

Povezivanje korisnika sa uređajima u Insight ACS sistemu se odvija u odnosu n-1, što podrazumeva da jedan korisnik može imati više zaduženih uređaja na svom nalogu, ali da jedan uređaj može imati samo jednog vlasnika.

Osnovne operacije za upravljanje korisničkim nalozima koje sprežni slojevi Insight ACS sitema dopuštaju su:

- Postavljanje korisnika određenom uređaju.
- Promenu podataka korisnika.
    - Ime,
    - Prezime,
    - Telefon,
    - E-mail,
    - Id korisnika,
    - Status u sistemu,
    - Lista uređaja asociranih sa korisničkim nalogom.
- Dobavljanje određenog korisnika ili korisnika.
    - Po određenom kriterijumu,
    - Po identifikatoru korisnika.
- Dobavljanje broja korisnika u sistemu.
    - Svih korisnika,
    - Aktivnih korisnika.
- Brisanje liste korisnika.

3.11 Specifične operacije poslužioca

3.11.1 Operacije sa geografskom lokacijom uređaja

Poslužilac dozvoljava postavljanje uređaja na određenu geografsku lokaciju i iščitivanje te lokacije sa spoljašnjih aplikacija. Geografska lokacija uređaja je definisana geografskim koordinatama.

Sprežni slojevi Insight ACS poslužioca isporučuju podatke specifične obrade vezane za geografsku interpolaciju. Na osnovu geografski definisanog prostora kao ulaznog parametra, moguće je preko aplikativnih sprežnih slojeva dobaviti interpolirane informacije o geografskoj raspodeli vrednosti određenog parametra.

3.11.2 Operacije sa upravljačkim nalozima

Upravljački nalozi predstavljaju korisnike podataka iz sprežnih slojeva (*API Users*), koji dopuštaju dobavljanje određenih podataka na osnovu prioriteta pristupa. Najčešće su asocirani za druge aplikacije i poslužioce. Nekim operacijama nije moguće pristupiti ukoliko zahtev nije stigao od administratora sistema. Ovaj mehanizam je deo bezbednosnih struktura aplikativnih sprežnih slojeva.

Operacije sa upravljačkim nalozima dopuštaju dobavljenje i promenu podataka određenim korisnicima. Takođe je dozvoljena promena nivoa pristupa korisniku.

3.11.3 Operacije sa akcionim skriptama

Aplikativni sprežni slojevi omogućuju razmenu podataka sa poslužiocem namenjenim za interpretiranje akcionih skripti. Mehanizam akcionih skripti dopušta spoljno definisanje akcionih skripta u datom programskom jeziku i na taj način se omogućuje specifična kontrola nad uređajima uz punu fleksibilnost. To podrazumeva da skripta može izvršiti predefinisane akcije i tako automatizovati određene procese, uglavnom vezane za čitanje i postavljenje nove vrednosti parametara. Za izvršenje akcione skripte spoljašnji poslužilac ili aplikacija koristi operacije sa parametrima, objašnjene u poglavlju 3.8.

Sprežni slojevi Insight ACS poslužioca omogućuju:

- Primanje sadržaja skripte sa spoljašnje aplikacije ili poslužioca.
- Brisanje i unošenje novih skripti u sistem.
- Isporuku sadržaja skripte i njenih parametara.
- Izmenu parametara skripte.
    - Uređaja na koje se primenjuje skripta,
    - Događaje na koje se skripta aktivira.

3.11.4 Operacije specifične za televizijske sisteme (Big Data informacije)

Insight ACS preko svojih aplikativnih slojeva može isporučiti nestrukturirane podatke spremne za naknadnu obradu a usko vezane za krajnje uređaje sa TV funkcionalnostima. Jedna od osnovnih primena IoT poslužioca u televizijskim mrežama je proračun gledanosti programa. Ukoliko krajnji uređaj je takve prirode i poštuje standard modela podataka za uređaje sa TV funkcionalnostima, sprežni slojevi mogu preneti podatke vezane za broj i gledanost kanala na tom uređaju.

Sledeće podatke o broju televizijskih programa i njihovoj gledanosti sprežni slojevi omogućuju:

- Broj krajnjih uređaja sa TV funkcionalnostima u sistemu,
- Listu svih uređaja sa TV funkcionalnostima u sistemu,
- Broj kanala na jednom uređaju sa TV funkcionalnostima u sistemu i
- Informacije o gledanosti kanala na datom uređaju sa TV funkcionalnošću.

Iako za ove operacije je moguće dobiti i podatke preko operacija sa parametrima, Insight ACS obavlja pretprocesiranje podataka vezanih za uređaje sa TV funkcionalnostima čime se postiže manje opterećenje na sprežne slojeve i bazu podataka.

4. Programsko rešenje

U ovom poglavlju je objašnjeno rešenje aplikativnog sprežnog sloja IoT poslužioca. Programsko rešenje je realizovano u programskom jeziku Java korišćenjem *Enterprise JavaBeans* (EJB) mehanizma u paru sa JAX-WS (*Java API for XML Web Services*) definisanih u okviru Java EE standarda i GWT mehanizma RPC poziva. Java EE okruženje je namenjeno kreiranju skalabilnih, web-orijentisanih programskih rešenja. Činioci rešenja API za IoT poslužilac su prikazani u Tabela 6 ujedno sa svojim opisom.

Modul	Opis
RPC API modul	Programsko rešenje RPC API modula.
• **ConfigurationServiceImpl**	Rukovanje korisnicima sistema, akcionim skriptama i aktivnostima poslužioca.
• **CpemanagementServiceImpl**	Rukovanje parametrima vezanim za krajnje uređaje, listama uređaja i grupama uređaja.
• **StatsServiceImpl**	Rukovanje prostornom raspodelom uređaja i vrednostima njihovih parametara.
REST API modul	Programsko rešenje REST API modula.
• **AndroidServices**	Funkcionalnosti namenjene mobilnim aplikacijama.
• **ActionScriptServices**	Funkcionalnosti namenjene aplikaciji za rukovanje akcionim skriptama.
WSDL API modul	Programsko rešenja WSDL API modula.
• **NBIServiceBean**	Rukovanje Big Data podacima i aktivnošću mreže.

Tabela 6. Programsko rešenje Insight ACS API.

4.1 Implementacija RPC API modula

4.1.1 ConfigurationServiceImpl

- ***public UserShared getLogedInUser()***

Opis: Pozivom ove operacije dobijaju se informacije o aktivnom korisniku koji poziva API metode iz date aplikacije.

Ulazni parametri: Nema.

Povratna vrednost: Struktura podataka sa podacim za korisnika. Struktura UserShared sadrži sledeća polja:

- id – jedinstveni identifikator korisnika,
- username – korisničko ime korisnika,
- password – lozinku korisnika,
- firstName – ime korisnika,
- lastName – prezime korisnika,
- email – elektronsku poštu korisnika,
- phone – mobilni telefon korisnika,
- language – preferirani jezik korisnika,
- userActive – status korisnika,
- role – nivo pristupa korisnika (administrator, regularan korisnik).

- ***public Boolean isUserInRole(String role)***

Opis: Metoda koja obavlja proveru nivoa pristupa korisnika (aplikacije). Ova operacija omogućava aplikacijama da obezbede sopstvene nivoe pristupa.

Ulazni parametri:

1. *role* - Nivo pristupa (administrator - *admin*, regularan korisnik - *user*).

Povratna vrednost: Logička vrednost provere nivoa pristupa korisnika.

- ***public void logout()***

Opis: Odjavljivanje do tada prijavljenog korisnika.

Ulazni parametri: Nema.

Povratna vrednost: Nema.

- ***public Boolean createUser(String username, String password, String role)***

Opis: Kreiranje novog korisnika u bazi podataka poslužioca. Ovakva funkcionalnost omogućava grafičkim aplikacijama kreiranje više korisničkih naloga iz jedne aplikacije.

Ulazni parametri:

1. *username* – korisničko ime novog naloga.

2. *password* – lozinka novog naloga.
3. *role* – nivo pristupa novokreiranog naloga.

- ***public Boolean checkCredentials(String username, String password)***

Opis: Provera usklađenosti korisničkog imena i lozinke. Ova funkcionalnost ne ograničava poziv na prijavljenog korisnika već se može pozvati za bilo kog korisnika u sistemu.

Ulazni parametri:

1. *username* – korisničko ime za proveru.
2. *password* – lozinka za proveru.

- ***public UserDetailsShared getUserDetails(String username)***

Opis: Dobijanje svih detalja za korisnika sa određenim korisničkim imenom. Ova funkcija zahteva administratorski nivo pristupa.

Ulazni parametri:

1. *username* – korisničko ime za koje se zahtevaju detalji.

Povratna vrednost: UserDetailsShared struktura sastavljena od UserShared strukture i RoleShared strukture koja se sastoji od sledećih polja:

- *id* – identifikator nivoa pristupa.
- *role* – naziv nivoa pristupa.

- ***public List<UserDetailsShared> getDetails()***

Opis: Ovom operacijom se dobijaju detalji za sve postojeće korisnike u sistemu.

Ulazni parametri: Nema.

Povratna vrednost: Lista *UserDetailsShared* postojećih struktura.

- ***public void updateUser(UserDetailsShared details)***

Opis: Promena detalja vezanih za određenog korisnika.

Ulazni parametri:

1. *details* – nova struktura za osvežavanje podataka o korisniku.

Povratna vrednost: Nema.

- ***public void deleteUser(String username)***

Opis: Brisanje postojećeg korisnika iz sistema.

Ulazni parametri:

1. *username* – korisničke ime naloga koji se briše.

Povratna vrednost: Nema.

- ***public OwnerInfoShared getOwnerInfo(DeviceIdStructShared deviceIdStruct)***

Opis: Ova operacija dobavlja podatke o vlasniku krajnjeg uređaja ukoliko on postoji u sistemu.

Ulazni parametri:

1. *deviceIdStruct* – jedinstvena identifikaciona struktura podataka krajnjeg uređaja sastavljena od:
    - manufacturer – proizvođač uređaja,
    - oui – jedinstveni identifikacioni broj,
    - productClass – klasa uređaja,
    - serialNumber – serijskog broj krajnjeg uređaja.

Povratna vrednost: Struktura podataka koja jedinstveno identifikuje vlasnika uređaja. Struktura je sastavljena od sledećih polja:

- owner – ime i prezime vlasnika,
- provider – pružaoc usluge krajnjem uređaju,
- address – adresa vlasnika,
- phone – broj telefona,
- email – elektronska pošta i
- id – jedinstveni identifikator.

- ***public Date getServerDate()***

Opis: Ovom metodom se dobija lokalno vreme po kojem poslužilac funkcioniše. Po pravilu korisnici API sloja poslužioca ne moraju biti u istoj vremenskoj zoni i ova podatak služi za sinhronizaciju.

Ulazni parametri: Nema.

Povratna vrednost: java.util.Date objekat.

- ***public ArrayList<ACSActivityShared> getACSActivities()***

Opis: Ova metoda omogućava dobavljanje informacija o trenutnim aktivnostima na poslužiocu. Te aktivnosti nose informacije o dobavljanju parametara, povezivanjem sa krajnjim uređajem i slično.

Ulazni parametri: Nema.

Povratna vrednost: Lista aktivnosti sa poslužioca. Svaka aktivnost je predstavljena strukturom ACSActivityShared koja ima sledeća polja:

- id – identifikator aktivnosti,
- name – ime aktivnosti,
- progress – procentualni progres date aktivnosti ukoliko postoji,
- report – dodatni tekst koji opisuje aktivnost,
- type – tip aktivnosti,
- activityStarted – vreme početka aktivnosti,
- importantNotification – da li je aktivnost visokog prioriteta,

- eventForDevice – da li je aktivnost vezana za određeni krajnji uređaj,
- eventCpe – uređaj za koji je uređaj vezan,
- deviceList – lista uređaja na kojima se primenjuje aktivnost,
- signature – jedinstven potpis aktivnosti.

- ***public boolean saveScript(ScriptInfoShared scriptInfo)***

Opis: Čuvanje akcione skripte na poslužiocu. Poslužilac je smešta u bazu podataka ili drugi tip skladišta podataka.

Ulazni parametri:

1. *scriptInfo* – struktura koja sadrži podatke o akcionoj skripti. ScriptInfoShared sadrži sledeća polja:
   - scriptName – ime skripte koja se čuva,
   - scriptCode – LUA kod skritpe za izvršavanje,
   - lastTimeRun – datum poslednjeg pokretanja skripte,
   - devices – niz uređaja na kojima se skripta primenjuje,
   - groups – grupe na kojima se skripta primenjuje,
   - rules – pravila po kojima se skripta aktivira,
   - event – događaj koji aktivira skriptu,
   - isRunning – status skripte.

Povratna vrednost: Logički status operacije čuvanja skripte.

- public boolean deleteScript(String scriptToDeleteName)

Opis: Brisanje akcione skripte sa poslužioca.

Ulazni parametri:

1. *scriptToDeleteName* – ime skripte čije se brisanje aktivira.

Povratna vrednost: Logički status operacije brisanja.

- ***public boolean addNewScript(ScriptInfoShared scriptInfo)***

Opis: Dodavanje nove skripte na poslužilac.

Ulazni parametri:

1. *scriptInfo* – struktura ScriptInfoShared koja opisuje sve osobine skripte.

Povratna vrednost: Logički status operacije dodavanja nove akcione skripte.

- ***public ScriptInfoShared getScriptInfo(String scriptName)***

Opis: Dobavljanje osobina određene akcione skripte.

Ulazni parametri:

1. *scriptName* – ime skripte čiji se podaci dobavljaju.

Povratna vrednost: ScriptInfoShared struktura koja nosi informacije o datoj skripti.

- ***public List<ScriptInfoShared> getAllScripts()***

Opis: Dobavljanje informacija o svim postojećim akcionim skriptama u sistemu.

Ulazni parametri: Nema.

Povratna vrednost: Lista ScriptInfoShared objekata za sve skripte koje postoje na poslužiocu.

- ***public boolean runScript(String scriptName)***

Opis: Pokretanje izvršavanja skripte.

Ulazni parametri:

1. *scriptName* – ime skritpe čije se izvršavanje pokreće.

Povratna vrednost: Logička vrednost koja označava da li je skripta uspešno pokrenuta.

- ***public boolean stopScript(String scriptName)***

Opis: Stopiranje izvršavanja date skripte.

Ulazni parametri:

1. *scriptName* – ime skritpe čije se izvršavanje pokreće.

Povratna vrednost: Logička vrednost koja označava da li je operacija stopiranja uspešno završena.

4.1.2 CpeManagementServiceImpl

- ***public List<CpeShared> getCpeList(int offset, int limit, String criteria, String value, String manufacturer, String productClass)***

Opis: Dobavljanje liste uređaja na poslužioci.

Ulazni parametri:

1. *offset* – od koje pozicije u listi,
2. *limit* – koliko uređaja,
3. *criteria* – po kom kriterijumu obaviti pretragu,
4. *value* – vrednost kriterijuma,
5. *manufacturer* – uređaji kojeg proizvođača,
6. *productClass* – klasa uređaja

Povratna vrednost: Lista CpeShared struktura koje karakterišu uređaje. CpeShared struktura sadrži sve podatke i osobine krajnjeg uređaja.

- ***public List<CpeShared> advancedSearch(int offset, int limit, ArrayList<QueryShared> queryList, String manufacturer, String productClass)***

Opis: Napredna pretraga liste uređaja na poslužiocu. Dopušta prosleđivanje proizvoljnog upita kroz listu upita.

Ulazni parametri: (isto kao prethodni)

1. queryList – lista upita po kojima se vrši pretraga

Povratna vrednost: Lista CpeShared struktura koje karakterišu uređaje.

- ***public int advancedSearchCount(ArrayList<QueryShared> queryList, String manufacturer, String productClass)***

Opis: Pretraga broja uređaja po upitu.

Ulazni parametri:

1. queryList,
2. manufacturer,
3. productClass.

Povratna vrednost: Broj pronađenih uređaja koji odgovara datom kriterijumu.

- ***public int getNumberOfDevices()***

Opis: Pretraga ukupnog broja uređaja.

Ulazni parametri: Nema.

Povratna vrednost: Broj pronađenih uređaja u poslužiocu.

- ***public int getNumberOfDevicesByCriteria(String criteria, String value, String manufacturer, String productClass)***

Opis: Pretraga broja broja uređaja po kriterijumu.

Ulazni parametri:

1. *criteria* – kriterijum za pretragu,
2. *value* – vrednost kriterijuma,
3. *manufacturer* – proizvođač,
4. *productClass* – klasa uređaja.

Povratna vrednost: Broj pronađenih uređaja

- ***public CpeShared getCpe(DeviceIdStructShared deviceStruct)***

Opis: Dobavljanje podataka o određenom uređaju.

Ulazni parametri:

1. *deviceStruct* – identifikaciona struktura

Povratna vrednost:

- ***public ArrayList<ParameterShared> getFixedParameters(CpeShared device)***

Opis: Dobavljanje svih fiksno definisanih parametara it TR-069 modela podataka.

Ulazni parametri:

1. *device* – CpeShared struktura uređaja za koji se dobavljaju parametri.

Povratna vrednost: Lista ParameterShared struktura podataka koji definišu svaki parametar identifikacionim poljima, tipom, vrednošću i dodatnim podacima.

- ***public List<Firmware> getFirmwareList(int offset, int limit)***

Opis: Dobavljanje liste svih sačuvanih programskih podrški u sistemu.

Ulazni parametri:

1. *offset* – od kog početnog broja u listi tražiti.
2. *limit* – koliko broj programskih podrški dobaviti.

Povratna vrednost: Lista Firmware struktura koja definiše programske podrške. Struktura u sebi ima identifikaciona i opisna polja za svaku programsku podršku.

- ***public void upgradeCpe(ArrayList<DeviceIdStructShared> structList, String firmwareName)***

Opis: Pokreće proces zamene i unapređenja programske podrške na datim uređajima.

Ulazni parametri:

1. *structList* – lista svih uređaja na kojima se pokreće procedura.
2. *firmwareName* – ime datoteke programske podrška koje se postavlja na uređaj.

Povratna vrednost: Nema.

- ***public void deleteFirmware(String firmwareName)***

Opis: Brisanje datoteke programske podrške iz poslužioca.

Ulazni parametri:

1. *firmwareName* – ime datoteke programske podrške koja se briše.

Povratna vrednost: Nema.

- ***public void editCpeParameter(CpeShared device, String path, String value)***

Opis: Postavljanje nove vrednosti parametra u TR-069 modelu podataka na određenom uređaju.

Ulazni parametri:

1. *device* – uređaj na kome se postavlja nova vrednost,
2. *path* – putanja parametra u stablu podataka,
3. *value* – vrednost koja se postavlja.

Povratna vrednost: nema.

- ***public void setCpeDeviceId(CpeShared device, String newDeviceId)***

Opis: Postavlja deviceId vrednosti. DeviceId predstavlje ime uređaja na poslužiocu.

Ulazni parametri:

1. device – uređaj na kome se menja deviceId ime.
2. newDeviceId – novo ime uređaja koje se zadaje.

Povratna vrednost: Nema.

- ***public void deleteCpe(List<DeviceIdStructShared> deviceList)***

Opis: Brisanje uređaja iz skladišta podataka poslužioca.

Ulazni parametri:

1. *deviceList* – lista uređaja koji se brišu sa poslužioca.

Povratna vrednost: Nema.

- ***public ArrayList<CpeParametersShared> getAllParams(DeviceIdStructShared deviceIdStruct)***

Opis: Dobavljanje svih parametara iz modela podataka na datom uređaju.

Ulazni parametri:

1. deviceIdStruct – jedinstveni identifikator uređaja za koji se dobavljaju parametri.

Povratna vrednost: Lista CpeParameterShared struktura koje označavaju parametra iz modela podataka na datom uređaju..

- ***public ParameterShared getParameter(DeviceIdStructShared struct, String path)***

Opis: Dobavljanje određenog parametra za određeni uređaj.

Ulazni parametri:

1. *struct* – jedinstveni identifikator uređaja za koji se potražuje parametar.
2. *path* – putanja u modelu podataka parametra koji se potražuje.

Povratna vrednost:

- ***public boolean setParameterValue(DeviceIdStructShared struct, String path, String value)***

Opis: Postavljenje vrednosti određenog parametra na određenom uređaju.

Ulazni parametri:

1. *struct* – identifikator uređaja,
2. *path* – putanja u modelu podataka,
3. *value* – vrednost parametra.

Povratna vrednost: Logička vrednost da li je operacija uspela.

- ***public ParameterShared reload(DeviceIdStructShared struct, String path)***

Opis: Ponovno učitavanje parametra.

Ulazni parametri:

1. *struct* - identifikator uređaja,
2. *path* – putanja parametra u modelu podataka.

Povratna vrednost:

- ***public ArrayList<String> getMultipleTreeObject(DeviceIdStructShared struct, String path)***

Opis: Dobavljanje stabla podataka na datoj putanji u modelu podataka gde su podaci tipa *multiple*.

Ulazni parametri:

1. *struct* - identifikator uređaja,
2. *path* – putanja parametra u modelu podataka.

Povratna vrednost: Lista imena parametara u stablu.

- ***public int addTreeObject(DeviceIdStructShared struct, String path)***

Opis: Dodavanje novog parametra u stablo podataka na određenoj putanji.

Ulazni parametri:

1. *struct* - identifikator uređaja,
2. *path* – putanja parametra u modelu podataka.

Povratna vrednost: Status operacije.

- ***public int deleteTreeObject(DeviceIdStructShared struct, String path)***

Opis: Brisanje stabla podataka u datoj putanju u modelu podataka.

Ulazni parametri:

1. *struct* - identifikator uređaja,
2. *path* – putanja parametra u modelu podataka.

Povratna vrednost: Status operacije.

- ***public boolean setCPEParameterAttributes(DeviceIdStructShared struct, String path, int paramID)***

Opis: Postavljenje vrednosti atributa određenog uređaja.

Ulazni parametri:

1. *struct* - identifikator uređaja,
2. *path* – putanja parametra u modelu podataka,
3. *paramID* – identifikator koji se postavlja kao nova vrednost.

Povratna vrednost: logička vrednost koja pokazuje da li je operacija uspešno izvršena.

- ***ArrayList<ParameterHistoryItemShared> getParameterHistory(ParameterShared parameter, int fromIndex, int toIndex)***

Opis: Dobavljanje istorije promene vrednosti parametra u funkciji od broja zatraženih tačaka.

Ulazni parametri:

1. *parameter* – parametar za koji se traži istorija,
2. *fromIndex* – od koje tačke u vremenu,
3. *toIndex* – do koje tačke u vremenu.

Povratna vrednost: Lista ParameterHistoryItemShared struktura koje nose informacije o parametru i vremenskom trenutku kao i o vrednosti.

- ***public ArrayList<String> getCPELog(DeviceIdStructShared device, int offset, int limit)***

Opis: Dobavljanje log datoteke sa određenog uređaja u redovima zapisa.

Ulazni parametri:

1. *device* – uređaj za koji se traži log datoteka,
2. *offset* – od kog reda log datoteke se zahteva,
3. *limit* – koliko redova log datoteke.

Povratna vrednost: Lista unosa log datoteke.

- ***public ArrayList<String> getChild(DeviceIdStructShared device, String parent)***

Opis: Dobavljanje svih parametara bez obzira na njihove atribute sa određene putanje.

Ulazni parametri:

1. *device* – uređaj za koji se zahtevaju parametri,
2. *parent* – putanja sa koje se traže parametri.

Povratna vrednost: Lista imena parametara.

- ***public String getObjectDescription(DeviceIdStructShared device, String path)***

Opis: Dobavljanje opisa određenog objekta u stablu podataka.

Ulazni parametri:

1. *device* - identifikator uređaja,
2. *path* – putanja parametra u modelu podataka.

Povratna vrednost: Opis parametra.

- ***public void requestLogUpdate(DeviceIdStructShared device)***

Opis: Zahteva se osvežavanje log datoteke. Traži se od kranjeg uređaja da ponovo pošalje svoju log datoteku.

Ulazni parametri:

1. *device* – uređaj kome se zahteva log datoteka.

Povratna vrednost: Nema.

- ***public ArrayList<String> getChannelNames(DeviceIdStructShared device)***

Opis: Dobavljanje mogućih kanala na određenom krajnjem uređaju sa TV funkcionalnostima.

Ulazni parametri:

1. *device* – Uređaj za koji se kanali traže.

Povratna vrednost: Lista imena kanala na uređaju.

- ***public boolean setCPEParameter(DeviceIdStructShared struct, String path, String value, int paramID)***

Opis: Postavljenje vrednosti parametra i atributa na određenom uređaju.

Ulazni parametri:

1. struct – identifikator uređaja,
2. path – putanja parametra,
3. value – vrednost,
4. paramID – atribut.

Povratna vrednost: Logička vrednost uspešnosti operacije.

- ***public void deleteCPE(DeviceIdStructShared deviceIdStructShared)***

Opis: Brisanje uređaja iz poslužioca.

Ulazni parametri:

1. *deviceIdStructShared* – identifikator uređaja koji se briše iz poslužioca.

Povratna vrednost: Nema.

- ***public void deleteCPEs(ArrayList<DeviceIdStructShared> listOfDevices)***

Opis: Brisanje liste uređaja sa poslužioca.

Ulazni parametri:

1. *listOfDevices* – lista uređaja koji se brišu.

Povratna vrednost: Nema.

- ***public ArrayList<ParameterHistoryItemShared> getParameterHistory(ParameterShared parameter, Date fromTime, Date toTime)***

Opis: Dobavljanje istorije promene parametra na određenom uređaju u zavisnosti od vremena.

Ulazni parametri:

1. *parameter* – parametar za koji se traži istorija,
2. *fromTime* – od kog vremena se traži istorija promene,
3. *toTime* – do kog vremena se traži istorija promene.

Povratna vrednost: Lista ParameterHistoryItemShared struktura koje nose vrednost u svakoj tački u kojoj je nastala promena.

- ***public ArrayList<SubscriberShared> getSubscriberList(int offset, int limit, String criteria)***

Opis: Dobavljanje pretplatnika u poslužiocu po kriterijumu.

Ulazni parametri:

1. *offset* – od kog pretplatnika u sistemu,
2. *limit* – koliko pretplatnika,
3. *criteria* – kriterijum.

Povratna vrednost: Lista SubscriberShared struktura koje nose informacije o svakom pretplatniku u poslužiocu.

- ***public long getSubscriberListSize(String criteria)***

Opis: Dobavljanje broja pretplatnika u poslužiocu na osnovu kriterijuma.

Ulazni parametri:

1. *criteria* – kriterijum.

Povratna vrednost: Broj pretplatnika po određenom kriterijumu.

- public boolean deleteSubscribers(ArrayList<String> deleteList)

Opis: Brisanje datih pretplatnika sa poslužioca.

Ulazni parametri:

1. *deleteList* – Lista pretplatnika za brisanje.

Povratna vrednost: Logička vrednost uspešnosti operacije brisanja.

- ***public SubscriberShared getSubscriberById(String searchId)***

Opis: Dobavljanje informacija o pretplatniku na osnovu njegovog identifikatora.

Ulazni parametri:

1. *searchId* – identifikator pretplatnika koji se potražuje.

Povratna vrednost: SubscriberShared struktura pretplatnika ukoliko postoji u sistemu.

- ***public String generatePDFReport(DeviceIdStructShared deviceIdStructS, String owner, String reportType, boolean includeSubscriberInfo, boolean includeViewingHabits, boolean includeLog, Date fromDate, Date toDate)***

Opis: Generisanje .pdf izveštaja za određeni uređaj u kome se nalaze informacije o uređaju, pretplatniku, log datoteci.

Ulazni parametri:

1. *deviceIdStructS* – identifikator uređaja,
2. *owner* – ime pretplatnika,
3. *reportType* – tip izveštaja,
4. *includeSubscriberInfo* – uključivanje informacija o pretplatniku,
5. *includeViewingHabits* – uključivanje informacija o gledanosti programa na uređaju ukoliko je uređaj zasnovan na TV funkcionalnostima,
6. *includeLog* – uključivanje log datoteke ukoliko postoji,
7. *fromDate* – od kog datuma se zahteva izveštaj,
8. *toDate* – do kog datuma se zahteva izveštaj.

Povratna vrednost: Putanja na kojoj se nalazi generisani izveštaj koji posle generisanja može biti dobavljen klijentskoj aplikaciji.

- ***public boolean setSubscriber(String ownerId, String firstName, String lastName, String email, String phone, boolean userActive, ArrayList<String> cpeSerials)***

Opis: Postavljanje novih informacija o pretplatniku.

Ulazni parametri:

1. *ownerId* – identifikator pretplatnika,
2. *firstName* – ime,
3. *lastName* – prezime,
4. *email* – elektronska pošta,
5. *phone* – telefon,
6. *userActive* – status pretplatnika,

7. *cpeSerials* – lista uređaja koji su mu pridruženi.

Povratna vrednost: Logički status operacije.

- ***public HashMap<IssueEnumShared, String> getIssueList(DeviceIdStructShared device)***

Opis: Dobavljanje grešaka na uređaju. Ukoliko uređaj prijavi da su se na njemu desile greške one se registruju u poslužiocu.

Ulazni parametri:

1. *device* – identifikator uređaja.

Povratna vrednost: Mapa tipa problema koji se dogodio na uređaju i poruka uz svaku grešku.

- ***public Set<CpeShared> getCPEListByOwnerId(String searchId)***

Opis: Dobavljanje liste uređaja koji su pridruženi određenom pretplatniku.

Ulazni parametri:

1. *searchId* – identifikator pretplatnika.

Povratna vrednost: Lista krajnjih uređaja.

- ***public boolean hasMultpleChild(DeviceIdStructShared deviceIdStruct, String path)***

Opis: Ispitivanje da li određeni parametar na uređaju u modelu podataka ima stablo ispod sebe.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja.
2. *path* – putanja parametra u stablu.

Povratna vrednost: Logički rezultat operacije pretrage.

- ***public boolean isRemovalbeObject(DeviceIdStructShared deviceIdStruct, String path)***

Opis: Ispitivanje da li je određeni parametar na uređaju moguće obrisati.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja.
2. *path* – putanja parametra u stablu.

Povratna vrednost: Logički rezultat koji pokazuje da li je parametar moguće izbrisati.

- public boolean rebootDevice(DeviceIdStructShared deviceIdStruct)

Opis: Pozivanje restart procedure na određenom uređaju.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja.

Povratna vrednost: Logički rezultat koji pokazuje da li je procedura uspešno izvršena.

- public boolean addGroup(String groupName)

Opis: Dodavanje nove grupe na poslužilac.

Ulazni parametri:

1. *groupName* – ime nove grupe.

Povratna vrednost: Logički rezultat koji pokazuje da li je grupa dodata.

- public boolean refreshBriefParams(DeviceIdStructShared deviceIdStruct, ArrayList<String> paramList)

Opis: Osvežavanje vrednosti datih parametara na uređaju.

Ulazni parametri:

1. *deviceIdStruct* - identifikator uređaja.
2. *paramList* – lista parametara čije se osvežavanje zahteva.

Povratna vrednost: Logički rezultat koji pokazuje da li su parametri osveženi.

- ***public boolean removeGroup(String groupName)***

Opis: Brisanje grupe sa poslužioca.

Ulazni parametri:

1. *groupName* – Ime grupe koja se briše sa poslužioca.

Povratna vrednost: Logički status operacije.

- ***public boolean removeGroups(ArrayList<String> groupNames)***

Opis: Brisanje više grupa sa poslužioca.

Ulazni parametri:

1. *groupNames* – lista grupa koje se brišu sa poslužioca.

Povratna vrednost: Logički status operacije brisanja.

- ***public int numberOfCPEFromGroup(String groupName)***

Opis: Dobavljanje broja uređaja u određenoj grupi.

Ulazni parametri:

1. *groupName* – ime grupe za koju se zahteva broj uređaja.

Povratna vrednost: Broj uređaja dodatih u datu grupu.

- ***public ArrayList<CpeShared> getCPEListFromGroup(String groupName, int offset, int limit)***

Opis: Dobavljanje liste uređaja u datoj grupi.

Ulazni parametri:

1. *groupName* – ime grupe za koju se zahtevaju uređaji.
2. *offset* – od kog uređaja se zahteva.
3. *limit* – koliko uređaja se zahteva.

Povratna vrednost: Lista uređaja dodatih u datu grupu.

- ***public Set<GroupShared> allGroupsOfCPE(DeviceIdStructShared deviceIdStruct)***

Opis: Dobavljanje svih grupa u koje je dodat dati uređaj.

Ulazni parametri:

1. *deviceIdStruct* - identifikator uređaja.

Povratna vrednost: Lista svih grupa u koje je dodat uređaj.

- ***public boolean addCPEToGroups(DeviceIdStructShared deviceIdStruct, ArrayList<String> groupNames)***

Opis: Dodati uređaj u određene grupe.

Ulazni parametri:

1. *deviceIdStruct* - identifikator uređaja.
2. *groupNames* – imena grupa u koje se uređaj dodaje.

Povratna vrednost: Logički status operacije.

- ***public boolean addCPEsToGroups(ArrayList<DeviceIdStructShared> deviceIdStructs, ArrayList<String> groupNames)***

Opis: Dodavanje više uređaja u više grupa.

Ulazni parametri:

1. *deviceIdStructs* - identifikatori uređaja.
2. *groupNames* – imena grupa u koje se uređaj dodaje.

Povratna vrednost: Logički status operacije.

- ***public boolean removeCPEFromGroup(DeviceIdStructShared deviceIdStruct, String groupName)***

Opis: Uklanjanje uređaja iz grupe.

Ulazni parametri:

1. *deviceIdStruct* - identifikator uređaja.
2. *groupName* – ime grupe iz koje se uređaj briše.

Povratna vrednost: Logički status operacije.

- ***public boolean removeCPEsToGroup(ArrayList<DeviceIdStructShared> deviceIdStructs, String groupName)***

Opis: Uklanjanje više uređaja iz više grupa.

Ulazni parametri:

1. *deviceIdStructs* - identifikatori uređaja.
2. *groupName* – ime grupe iz koje se uređaji brišu.

Povratna vrednost: Logički status operacije.

- ***public ArrayList<CpeShared> searchCPEByCriteriaFromGroup(String groupName, String criteria, String value)***

Opis: Pretraživanje određenog uređaja u grupi prema kriterijumu.

Ulazni parametri:

1. *groupName* – ime grupe koje se pretražuje.

2. *criteria* – kriterijum za pretragu.
3. *value* – vrednost kriterijuma.

Povratna vrednost: Lista pronađenih uređaja koji zadovoljavaju kriterijume,

- ***public int getCPERebootStatus(DeviceIdStructShared deviceIdStruct)***

Opis: Dobavljanje statusa operacije restartovanja na uređaju.

Ulazni parametri:

1. *deviceIdStruct* - identifikator uređaja.

Povratna vrednost: Status restart operacije.

- ***public ArrayList<GroupShared> findGroupsByPartialName(int limit, int offset, String name)***

Opis: Pronalaženje grupa po delimičnom imenu.

Ulazni parametri:

1. *limit* – koliko grupa se traži,
2. *offset* – od koje grupe se traži,
3. *name* – parcijalno ime.

Povratna vrednost: Lista grupa koje zadovoljavaju kriterijume.

- ***public long getNumberOfGroupsByPartialName(String name)***

Opis: Dobavljanje broja grupa po parcijalnom imenu.

Ulazni parametri:

1. *name* – parcijalno ime.

Povratna vrednost: Broj grupa koje zadovoljaju kriterijum parcijalnog imena.

- ***public boolean removeCPEsFromGroups(ArrayList<DeviceIdStructShared> deviceIdStructs, ArrayList<String> groupNames)***

Opis: Uklanjanje uređaja iz grupa.

Ulazni parametri:

1. *deviceIdStructs* - identifikatori uređaja,
2. *groupNames* – ime grupa iz kojih se uređaji brišu.

Povratna vrednost:

- ***public ArrayList<DistributionParamShared> getOutletDistribution(DeviceIdStructShared deviceIdStruct, Date fromDate, Date toDate)***

Opis: Dobavljanje raspodele potrošnje struje na pametnim utičnicima ukoliko je uređaj tog tipa.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja,

2. *fromDate* – od kog datuma se zahteva potrošnja,
3. *toDate* – do kog datuma se traži vrednost.

Povratna vrednost: Lista DistributionParamShared struktura koji pokazuju raspodelu potrošnje energije između pametnih utičnica.

- ***ArrayList<HomeManagerItemShared> getHomeManagerDevices(DeviceIdStructShared deviceIdStruct)***

Opis: Dobavljanje svih pametnih utičnica i sijalica na datom uređaju za kontrolu osvetljenja i potrošnje energije.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja.

Povratna vrednost: Lista HomeManagerItemShared struktura koji opisuju svaki čvor na uređaju za kontrolu osvetljenja i potrošnje.

- ***ArrayList<ParameterHistoryItemShared> getDimmerHistory(DeviceIdStructShared deviceIdStruct, int index, Date fromDate, Date toDate)***

Opis: Dobavljanje istorije promene nivoa osvetljenja ukoliko takav parametar postoji na datom uređaju.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja,
2. *index* – redni broj pametne sijalice na uređaju koji kontroliše osvetljenje,
3. *fromDate* – od kog datuma se zahteva istorija,
4. *toDate* – do kog datuma se zahteva istorija.

Povratna vrednost: Lista ParameterHistoryItemShared struktura.

- ***ArrayList<ParameterHistoryItemShared> getOutletPowerConsumption(DeviceIdStructShared deviceIdStruct, int index, Date fromDate, Date toDate)***

Opis: Dobavljanje istorije promene potrošnje struke na određenoj pametnoj utičnici.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja,
2. *index* – redni broj pametne utičnice na uređaju koji kontroliše potrošnju,
3. *fromDate* – od kog datuma se zahteva istorija,
4. *toDate* – do kog datuma se zahteva istorija.

Povratna vrednost: Lista ParameterHistoryItemShared struktura.

- ***public ArrayList<ParameterShared> getParameters(DeviceIdStructShared device, ArrayList<String> paramlist)***

Opis: Dobavljanje detaljnih informacija o parametrima na osnovu njihovih imena i identifikatora uređaja.

Ulazni parametri:

1. *device* – identifikator uređaja,
2. *paramlist* – lista imena paramatara za koje se traže detalji.

Povratna vrednost: Lista ParameterShared struktura.

- ***public String getSignalStrengthOnFrontEnd(DeviceIdStructShared device, int feNumber)***

Opis: Dobavljanje vrednosti snage signala na IP TV uređajima.

Ulazni parametri:

1. *device* – identifikator uređaja,
2. *feNumber* – redni broj FE na datom IP TV uređaju.

Povratna vrednost:Vrednost snage signala.

- ***public String getCPUUsage(DeviceIdStructShared device)***

Opis: Procentualna iskorišćenost procesora ukoliko taj parametar postoji.

Ulazni parametri:

1. *device* – identifikator uređaja.

Povratna vrednost: Procenat iskorišćenosti procesora.

- ***public String getMemoryUsage(DeviceIdStructShared device)***

Opis: Procentualna iskorišćenost memorije ukoliko taj parametar postoji.

Ulazni parametri:

1. *device* – identifikator uređaja.

Povratna vrednost: Procenat iskorišćenosti memorije.

- ***public String getTemperature(DeviceIdStructShared device)***

Opis: Dobavljanje temperature uređaja ukoliko taj parametar postoji.

Ulazni parametri:

1. *device* – identifikator uređaja.

Povratna vrednost: Vrednost temperature u celzijusima ukoliko parametar postoji.

- ***public List<String> getGroupNames()***

Opis: Dobavljanje svih grupa koje postoje na poslužiocu.

Ulazni parametri: Nema.

Povratna vrednost: Lista imena grupa na poslužiocu.

- ***public IssueSolutionStepShared getIssueSolutionStep(int id)***

Opis: Dobavljanje koraka za ispravljanje problema definisanih u poslužiocu.

Ulazni parametri:

1. *id* – identifikator koraka čiji se detalji zahtevaju.

Povratna vrednost: IssueSolutionStepShared struktura koja nosi sve informacije o datom koraku za ispravljanje problema.

- ***public void setCPEBriefParamList(DeviceIdStructShared deviceIdStruct, ArrayList<String> configurablePaths)***

Opis: Postavljenje grupe parametara koji će imati često osvežavanje.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja,
2. *configurablePaths* – putanje parametara koji će se često osvežavati.

Povratna vrednost: Nema.

- ***public ArrayList<ParameterShared> getCPEBriefParamList(DeviceIdStructShared deviceIdStruct)***

Opis: Dobavljanje grupe parametara koji se često osvežavaju.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja.

Povratna vrednost: Lista ParameterShared struktura koje nose informacije o svakom parametru.

- ***ArrayList<ParameterShared> findCPEParametersBySubstringPath(DeviceIdStructShared deviceIdStruct, String str)***

Opis: Dobavljanje parametara na osnovu dela imena.

Ulazni parametri:

1. *deviceIdStruct* – identifikator uređaja,
2. *str* – deo imena na osnovu kojeg se pretražuje.

Povratna vrednost: Lista parametara koji su pronađeni na osnovu dela imena.

4.1.3 StatsServiceImpl

- ***public MapInfo getSNRLocationList()***

Opis: Dobavljanje informacija o geografskoj raspodeli.

Ulazni parametri: Nema.

Povratna vrednost: Povratna vrednost je MapInfo struktura koja ima sledeća polja:

- *resolution* – rezoluciju (razdaljinu) između tačaka na mapi
- *spotList* – list tačaka sa geografskim koordinatama i snagom signala u njima.
-

- ***public void refreshHeatMap()***

Opis: Iniciranje osvežavanje geografske raspodele signala. Pozivanjem ove rutine, procedura na poslužiocu se može inicirati iz spoljašnje aplikacije.

Ulazni parametri: Nema.

Povratna vrednost: Nema.

- ***public ArrayList<Spot> getRawData()***

Opis: Dobavljanje informacija o geografskoj raspodeli uređaja i njihovih parametara.

Ulazni parametri: Nema.

Povratna vrednost: Lista Spot struktura koje nose sledeće informacije:

  - *longitude* – geografska širina
  - *latitude* – geografska dužina
  - *ber* – bit error rate parametar
  - *snr* – signal-to-noise ratio parametar.

4.2 Implementacija REST API modula

REST moduli koriste JSON kao protokol fizičkog nivoa prenosa informacija, pa su tako sve *zahtev/odgovor* strukture podataka implementirane kao JSON-kompatibilne klase. To podrazumeva da su sve strukture za zahtev i odgovor lako serijalizabilne i da im format podataka odgovara formatu JSON poruka. Pomoćne klase predstavljaju i ulazni parametar i povratnu vrednost svake operacije. Sve klase sadrže zajednička polja kao što su *messageType* koji označava tip poruke koja se šalje i *errorCode* koji je tipičan za odgovor na zahtev.

4.2.1 AndroidServices

Uporedni prikaz pomoćnih klasa sa specifičnim poljima za *zahtev/odgovor* je prikazan u Tabela 7.

Funkcija klase	Zahtev	Odgovor
Login	• *password* – lozinka • *username* – korisničko ime	• status – logički status login operacije
CPEList	• *offset* – od kog uređaja • *limit* – koliko uređaja	• *cpeList* – lista uređaja
SearchCPEList	• *offset* – od kog uređaja • *limit* – koliko uređaja	• *cpeList* – lista uređaja

	• *criteria* – kriterijum pretrage • *value* – vrednost kriterijuma	
CPEStatus	• *device* - uređaj	• status uređaja
CPEParameters	• *device* – uređaj • *paths* – putanje do parametara	• *cpeParam* – lista parametara
HeatMap	Koordinate prostora: • *upperLeftLongitude* • *upperLeftLatitude* • *lowerRightLongitude* • *lowerRightLatitude*	• *spots* – lista Spots struktura koja definiše geografske tačke i kvalitet signala u njima
CPECoordinate	• *longitude* – koordinate • latitude • *device* - uređaj	• status
CPESetParam	• *device* – uređaj • *paths* – putanje parametara • *values* - vrednosti	• status
CPEParamHistory	• *deviceIdStruct* – uređaj • *path* – putanja do parametra	• *history* – lista ParameterHistoryItem struktura
CPETreeChilder	• *device* – uređaj • *parentPath* – putanja parametra	• *branchesPaths* – putanja parametara u podstablu

Tabela 7. Odgovor/zahtev strukture u AndroidServices podmodulu.

- ***public LoginResponse login(LoginRequest request)***

Opis: Operacije prijavljivanja korisnika na poslužilac.

Ulazni parametri: LoginRequest struktura.

Povratna vrednost: LoginResponse struktura.

- ***public CPEListResponse getCPEList(CPEListRequest request)***

Opis: Dobavljanje liste uređaja.

Ulazni parametri: CPEListRequest struktura.

Povratna vrednost: CPEListResponse struktura.

- ***public SearchCPEListResponse searchCPEList(SearchCPEListRequest request)***

Opis: Pretraga liste uređaja po zadatom kriterijumu.

Ulazni parametri: SearchCPEListRequest struktura.

Povratna vrednost: SearchCPEListResponse struktura.

- ***public CPEStatusResponse getCPEStatus(CPEStatusRequest request)***

Opis: Dobavljanje statusa određenog uređaja. Status može biti uključen/isključen.

Ulazni parametri: CPEStatusRequest struktura.

Povratna vrednost: CPEStatusResponse struktura.

- ***public CPEParametersResponse getCPEParameters(CPEParametersRequest request)***

Opis: Dobavljanje vrednosti parametara na određenim putanjama u uređaju.

Ulazni parametri: CPEParametersRequest struktura.

Povratna vrednost: CPEParametersResponse struktura.

- ***public HeatMapResponse getHeatMap(HeatMapRequest request)***

Opis: Dobavljanje geografske raspodele uređaja i njihovih parametara.

Ulazni parametri: HeatMapRequest struktura.

Povratna vrednost: HeatMapResponse struktura.

- ***public CPECoordinateResponse setCPECoordinate(CPECoordinateRequest request)***

Opis: Postavljanje novih koordinata uređaja u poslužiocu. Ove koordinate se koriste prilikom računanja geografske raspodele kvaliteta signala.

Ulazni parametri: CPECoordinateRequest struktura.

Povratna vrednost: CPECoordinateResponse struktura.

- ***public CPESetParamResponse setCPEParameter(CPESetParamRequest request)***

Opis: Postavljenje nove vrednosti određenog parametra na uređaju.

Ulazni parametri: CPESetParamRequest struktura.

Povratna vrednost: CPESetParamResponse struktura.

- ***CPEParamHistoryResponse getCPEParameterHistory(CPEParamHistoryRequest request)***

Opis: Dobavljanje istorije promene određenog parametra na poslužiocu.

Ulazni parametri: CPEParamHistoryRequest struktura.

Povratna vrednost: CPEParamHistoryResponse struktura.

- ***public CPETreeChildrenResponse getCPETreeChildren(CPETreeChildrenRequest request)***

Opis: Dobavljanje podstabla određenog parametra u modelu podataka na određenom uređaju.

Ulazni parametri: CPETreeChildrenRequest struktura.

Povratna vrednost: CPETreeChildrenResponse struktura.

4.2.2 ActionScriptServices

Ovaj podmodul REST API modula aplikativnog sprežnog sloja je namenjen za komunikaciju sa aplikacijama koje žele da čitaju parametre i upravljanju parametrima i modelom podataka. Prilagođen je komunikaciji sa aplikacijama koje koriste akcione skripte kako bi konfigurisale uređaje. Pomoćne JSON klase koje se koriste kao ulazni parametri i povratne vrednosti su prikazane u .

Funkcija klase	Zahtev	Odgovor
AddObject	• *objectPath* – putanja do modela podataka • *devId* – uređaj	• *addObjectStatus* – status addObject operacije
DeleteObject	• *objectPath* – putanja do modela podataka • *devId* – uređaj	• *deleteObjectStatus* – status deleteObject operacije
FactoryReset	• *devId* – uređaj	• status
Reboot	• *devId* – uređaj	• status
GetParameterValue	• *devId* – uređaj • *paramPath* – putanja parametra	• *parameters* – niz parametara
SetParameterValue	• *devId* – uređaj • *paramPath* – putanja parametra • *value* - vrednost	• *setParamStatus* – status operacije postavljanja vrednosti
GetParameterAttributes	• *devId* – uređaj • *paramPath* – putanja parametra	• *parameters* – atributi parametra
SetParamAttributes	• *devId* – uređaj • *paramPath* – putanja parametra • *notification* – promena	• status

	koja nije došla iz poslužioca • *accessList* - definisanje prava pristupa parametru	
ScriptStop	• *scriptName* – ime skriptež • *scriptID* – identifikator skripte	• *ack* - potvrda prijema informacije

Tabela 8. Odgovor/zahtev strukture u ActionScriptServices podmodulu.

- ***public AddObjectACSResponse actionAddObject(AddObjectACSRequest request)***

Opis: Dodavanje novog objekta u model podataka određenog uređaja.

Ulazni parametri: AddObjectACSRequest struktura.

Povratna vrednost: AddObjectACSResponse struktura.

- ***public DeleteObjectACSResponse actionDeleteObject(DeleteObjectACSRequest request)***

Opis: Brisanje objekta iz modela podataka na datom uređaju.

Ulazni parametri: DeleteObjectACSRequest struktura.

Povratna vrednost: DeleteObjectACSResponse struktura.

- ***public FactoryResetACSResponse actionFactoryReset(FactoryResetACSRequest request)***

Opis: Fabričko restartovanje uređaja.

Ulazni parametri: FactoryResetACSRequest struktura.

Povratna vrednost: FactoryResetACSResponse struktura.

- ***public RebootACSResponse actionReboot(RebootACSRequest request)***

Opis: Restartovanje i ponovo učitavanje programske podrške uređaja.

Ulazni parametri: RebootACSRequest struktura.

Povratna vrednost: RebootACSResponse struktura.

- ***GetParameterValueACSResponse actionGetParameterValue(GetParamValueACSRequest request)***

Opis: Dobavljanje vrednosti parametra na određenom uređaju.

Ulazni parametri: GetParamValueACSRequest struktura.

Povratna vrednost: GetParameterValueACSResponse struktura.

- ***SetParameterValueACSResponse actionSetParameterValue(SetParamValueACSRequest request)***

Opis: Postavljanje nove vrednosti parametra na određenom uređaju.

Ulazni parametri: SetParamValueACSRequest struktura.

Povratna vrednost: SetParameterValueACSResponse struktura.

- ***GetParameterAttributeACSResponse actionGetParameterAttribute(GetParamAttrACSRequest request)***

Opis: Dobavljanje atributa određenog paramatra iz modela podataka na uređaju.

Ulazni parametri: GetParamAttrACSRequest struktura.

Povratna vrednost: GetParameterAttributeACSResponse struktura.

- ***SetParameterAttributeACSResponse actionSetParameterAttribute(SetParamAttrACSRequest request)***

Opis: Postavljanje atributa određenog parametra na uređaju.

Ulazni parametri: SetParamAttrACSRequest struktura.

Povratna vrednost: SetParameterAttributeACSResponse struktura.

- ***public String notifyScriptStop(ScriptDataObject request)***

Opis: Obaveštavanje o završetku izvršenja skripte.

Ulazni parametri: ScriptDataObject struktura.

Povratna vrednost: String potvrde.

4.3 Implementacija WSDL API modula

Autorizaciona provera u WSDL API modulu utiče na povratnu vrednost operacija definisanih u ovom modulu pristupnog sloja. Ukoliko korisnik prilikom poziva operacije u zaglavlje HTTP poruke ne postavi odgovarajuće korisničko ime i lozinku, operacije će kao povratnu vrednost vratiti izuzetak (*exception*) tipa *SOAPFaultException*, definisanog u JAX-WS okruženju sa porukom "Unauthorized Exception".

4.3.1 NBIServiceBean

- ***public long getActiveDevices()***

Opis: Dobavljenje broja aktivnih uređaja u ekosistemu.

Ulazni parametri: Nema.

Povratna vrednost: Broj uređaja.

- ***public long getSNRofCPE(long id)***

Opis: Dobavljanja *Signal-to-Noise Ratio* parametra na uređaju ukoliko on postoji.

Ulazni parametri: Identifikator uređaja.

Povratna vrednost: Vrednost parametra.

- ***public long getBERofCPE(long id)***

Opis: Dobavljanja *Bit Error Rate* parametra na uređaju ukoliko on postoji.

Ulazni parametri: Identifikator uređaja.

Povratna vrednost: Vrednost parametra.

- ***public List<Long> getCPEids()***

Opis: Dobavljanje svih identifikatora u sistemu.

Ulazni parametri: Nema.

Povratna vrednost: Lista identifikatora uređaja.

- ***public List<Long> getBERValuesGEO(double lowest_lat, double highest_lat, double lowest_long, double highest_long)***

Opis: Dobavljanje vrednosti svih BER parametara sa određenog geografskog prostora ukoliko postoje.

Ulazni parametri:

1. *lowest_lat* – od koje geografske dužine,
2. *highest_lat* – do koje geografske dužine,
3. *lowest_long* – od koje geografske širine,
4. *highest_long* – do koje geografske širine.

Povratna vrednost: lista vrednosti BER parametara u zadatom opsegu.

- ***public List<Long> getSNRValuesGEO(double lowest_lat, double highest_lat, double lowest_long, double highest_long)***

Opis: Dobavljanje vrednosti svih SNR parametara sa određenog geografskog prostora ukoliko postoje.

Ulazni parametri:

1. *lowest_lat* – od koje geografske dužine,
2. *highest_lat* – do koje geografske dužine,
3. *lowest_long* – od koje geografske širine,
4. *highest_long* – do koje geografske širine.

Povratna vrednost: lista vrednosti SNR parametara u zadatom opsegu.

- ***public List<Channel> getChannelsOnCPE(long id)***

Opis: Dobavljanje svih gledanih kanala na određenom uređaju ukoliko takav parametar postoji (vezano za TV zasnovane uređaje).

Ulazni parametri: Identifikator uređaja.

Povratna vrednost: Lista Channel struktura koje karakterišu televizijske programe. Channel struktura se sastoji od sledećih polja:

- *channelName* – ima kanala,
- *duration* – dužina gledanja u sekundama.

- ***public List<NBIParameterHistoryItem> getActiveServiceHistory(String manufacturer, String oui, String productClass, String serialNumber, int fromInd, int toInd)***

Opis: Dobavljanje istorije promene aktivnog televizijskog programa na uređaju, ukoliko postoji.

Ulazni parametri:

1. *manufacturer* – proizvođač uređaja,
2. *oui* – identifikacioni broj,
3. *productClass* – klasa uređaja,
4. *serialNumber* – serijski broj,
5. *fromInd* – od kog indeks dostaviti promene,
6. *toInd* – do kog indeksa dostaviti promene.

Povratna vrednost: Lista NBIParameterHistoryItem struktura koje nose informacije o promeni kanali i vremenskom trenutku.

- ***public List<NBIDeviceIdStruct> getAllDevices()***

Opis: Dobavljanje svih uređaja sa njihovim identifikacionim strukturama.

Ulazni parametri: Nema.

Povratna vrednost: Lista NBIDeviceIdStruct struktura koja poseduje identična polja kao deviceIdStruct struktura.

- ***public String getParameterValue(NBIDeviceIdStruct deviceIdStruct, String paramPath)***

Opis: Dobavljanje vrednosti određenog parametra.

Ulazni parametri:

1. *deviceIdStruct* – identifikaciona struktura.
2. *paramPath* – putanja parametra čija se vrednost dobavlja.

Povratna vrednost: Vrednost parametra.

- ***public void setParameterValue(NBIDeviceIdStruct deviceIdStruct, String paramPath, String value)***

Opis: Postavljanje nove vrednosti parametra.

Ulazni parametri:

1. *deviceIdStruct* – identifikaciona struktura.
2. *paramPath* – putanja parametra čija se vrednost dobavlja.

3. *value* – vrednost parametra.

Povratna vrednost: Nema.

- ***public List<String> getMultipleObject(NBIDeviceIdStruct deviceIdStruct, String path)***

Opis: Dobavljanje parametara i objekata u stablu koji se nalaze pod određenom putanjom a tipa su Multiple.

Ulazni parametri:

1. *deviceIdStruct* – identifikaciona struktura.
2. *path* – putanja za koju se traži podstablo.

Povratna vrednost: Lista svih parametara koji zadovoljavaju kriterijum.

- ***public int addObject(NBIDeviceIdStruct deviceIdStruct, String path)***

Opis: Dodavanje novog objekta u stablo modela podataka na uređaju.

Ulazni parametri:

1. *deviceIdStruct* – identifikaciona struktura.
2. *path* – putanja objekta.

Povratna vrednost: Status operacije.

- ***public int deleteObject(NBIDeviceIdStruct deviceIdStruct, String path)***

Opis: Brisanje objekta iz modela podataka određenog uređaja.

Ulazni parametri:

1. *deviceIdStruct* – identifikaciona struktura.
2. *path* – putanja objekta.

Povratna vrednost: Status operacije.

- ***public String getObjectDescription(NBIDeviceIdStruct deviceIdStruct, String path)***

Opis: Dobavljanje opisa određenog objekta u modelu podataka.

Ulazni parametri:

1. *deviceIdStruct* – identifikaciona struktura.
2. *path* – putanja objekta.

Povratna vrednost: Opis objekta.

- ***public boolean hasMultpleChild(NBIDeviceIdStruct deviceIdStruct, String path)***

Opis: Ispitivanje da li na putanji u modelu podataka postoje objekti ili parametri u podstablu koji su tipa multiple.

Ulazni parametri:

1. *deviceIdStruct* – identifikaciona struktura.
2. *path* – putanja u modelu podataka.

Povratna vrednost: Logička vrednost operacije.

- ***public void reloadObject(NBIDeviceIdStruct deviceIdStruct, String path)***

Opis: Ponovno učitavanje objekta.

Ulazni parametri:

1. *deviceIdStruct* – identifikaciona struktura.
2. *path* – putanja za ponovno učitavanje.

Povratna vrednost: Nema.

- ***public List<NBIParameterHistoryItem> getParameterHistory(NBIDeviceIdStruct deviceIdStruct, String paramPath, Date fromDate, Date toDate)***

Opis: Dobavljanje istorije promene vrednosti proizvoljnog parametra.

Ulazni parametri:

1. *deviceIdStruct* – identifikaciona struktura.
2. *path* – putanja parametra.
3. *fromDate* – datum od kog se traži istorija.
4. *toDate* – datum do kog se traži istorija.

Povratna vrednost: Lista NBIParameterHistoryItem struktura koje definišu vremenske trenutke i vrednosti parametra za koji se traži istorija promene.

5. Ispitivanje i rezultati

Rešenje aplikativnog sprežnog sloja je integrisano u Insight ACS IoT poslužilac. Ispitivanje funkcionalnosti i karakteristika je obavljeno sa četiri korisničke aplikacije:

1. Web aplikacija,
2. Mobilna (Android) aplikacija,
3. Aplikacija za izvršavanje akcionih skripti i
4. Insight Stats - poslužilac za Big Data podatke.

Web aplikacija je korisnik RPC API modula, mobilna aplikacija REST API modula dok je razmena velike količine podataka obavljena sa drugim poslužiocem preko WSDL API modula. Aplikacija za izvršavanje akcionih skripti koristi REST API modul za upravljanje parametrima krajnjeg uređaja.

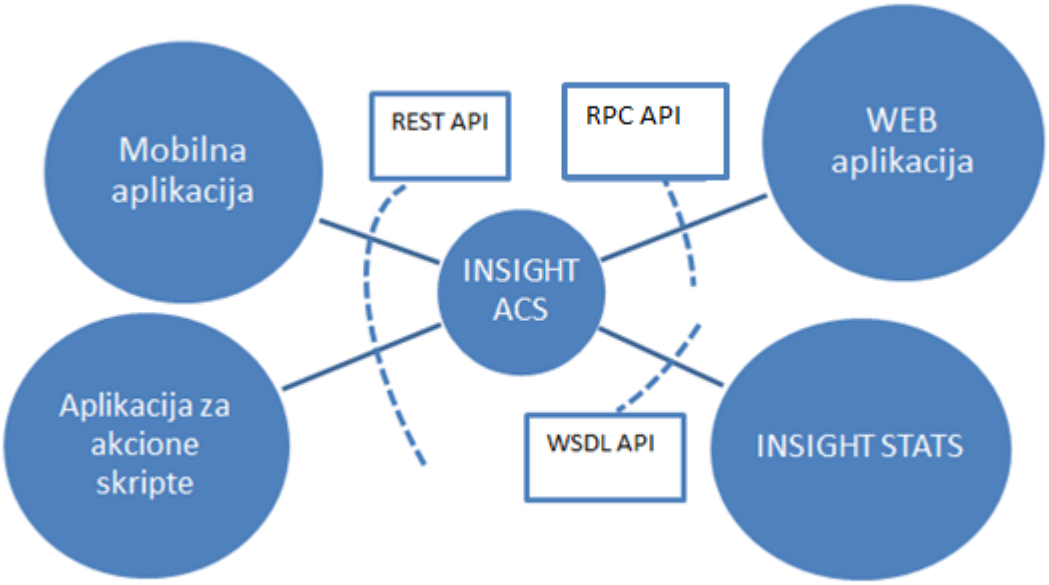
Web aplikacija je implementirana korišćenjem GWT okruženja. Namena takve aplikacije je da pruži alat operatorima i pružaocima multimedijalnih usluga za praćenje i daljinsko upravljanje uređajem. Uz to dopušta vlasnicima uvid u svoj uređaj preko URL-a.

Mobilna aplikacija omogućuje serviserima i korisnicima pristup određenim ograničenim funkcionalnostima.

Kako se ne bi preterano opteretile funkcionalnosti Insight ACS poslužioca, Big Data podaci koji se tiču statističkih informacija o gledanosti televizijskih programa u mreži se prenose drugom poslužiocu na obradu. Insight Stats predstavlja poslužilac koji se bavi obradom velike količine podataka i stvara slike koje pomažu analizi gledanosti programa.

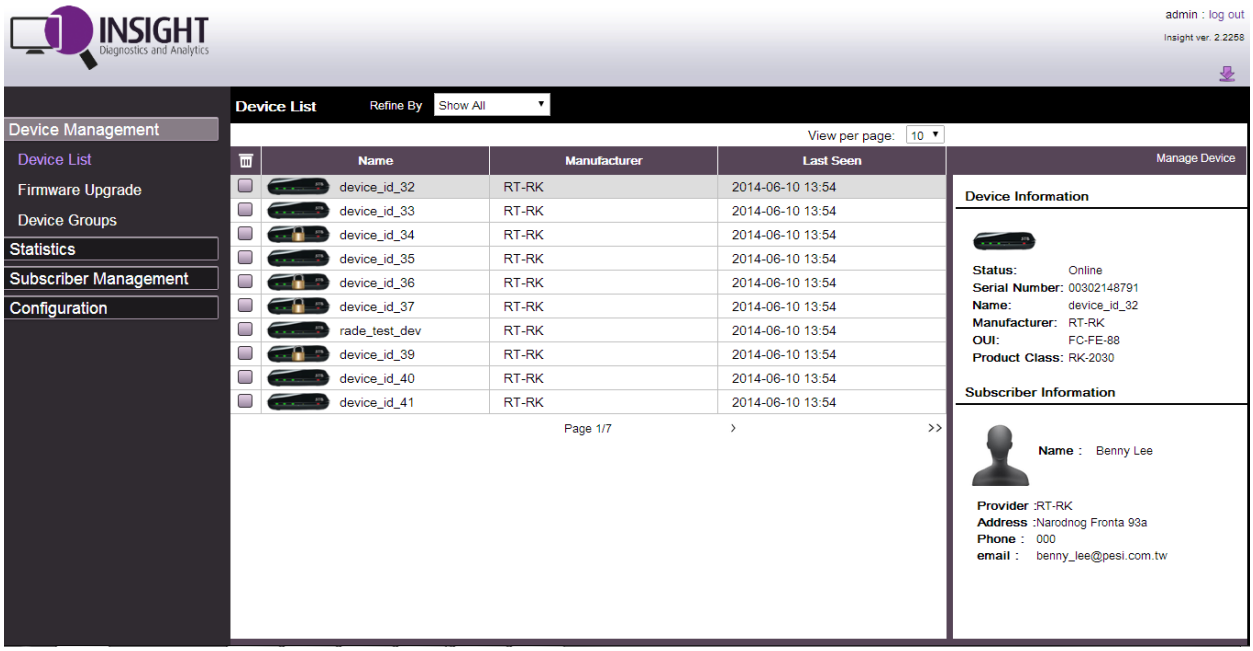
Mehanizam akcionih skripti dozvoljava definisanje akcionih skripti u web aplikacijama i njihovo izvršavanje u aplikacijama za izvršavanje akcionim skripti (*Script Executor*).

Opšti prikaz okruženja Insight ACS poslužioca i klijentskih aplikacija je prikazan na Slici 15.



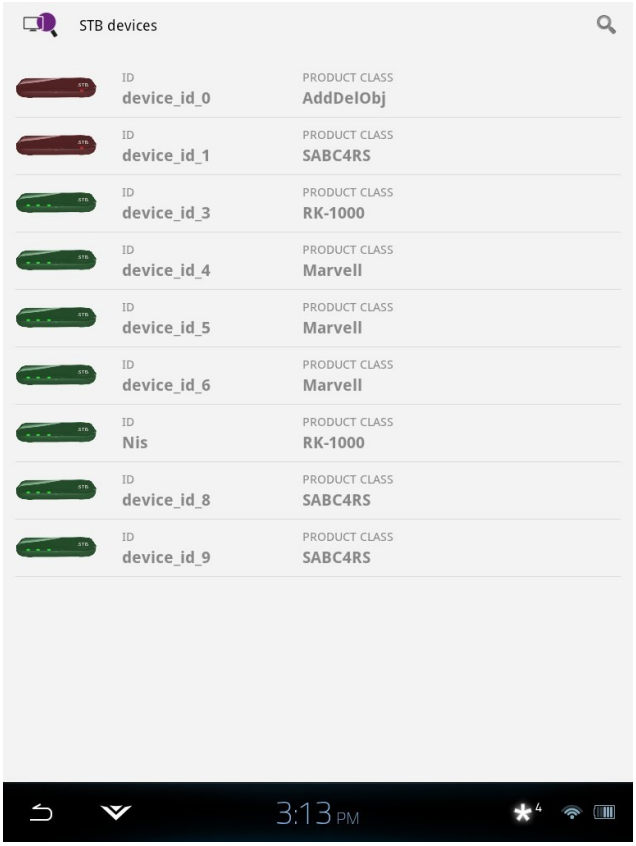
Slika 15. Ispitno okruženje – Insight ACS poslužilac i klijentske aplikacije.

Web aplikacija koristi ConfigurationServiceImpl, CpeManagementServiceImpl i StatsServiceImpl module kako bi komunicirala sa jezgrom poslužioca i grafički predstavljala podatka krajnjim korisnicima. Izgled aplikacije je prikazan na Slici 16.



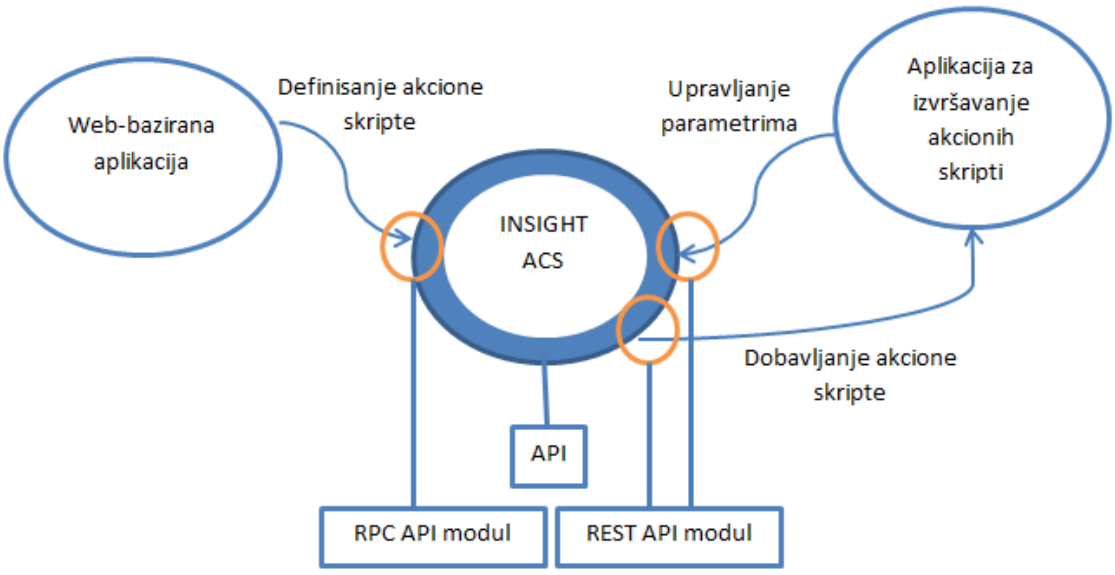
Slika 16. Web aplikacija.

Mobilna aplikacija namenjena upravljanju na terenu kako pružaocima usluga, tako i vlasnicima uređaja je prikazana na Slici 17.



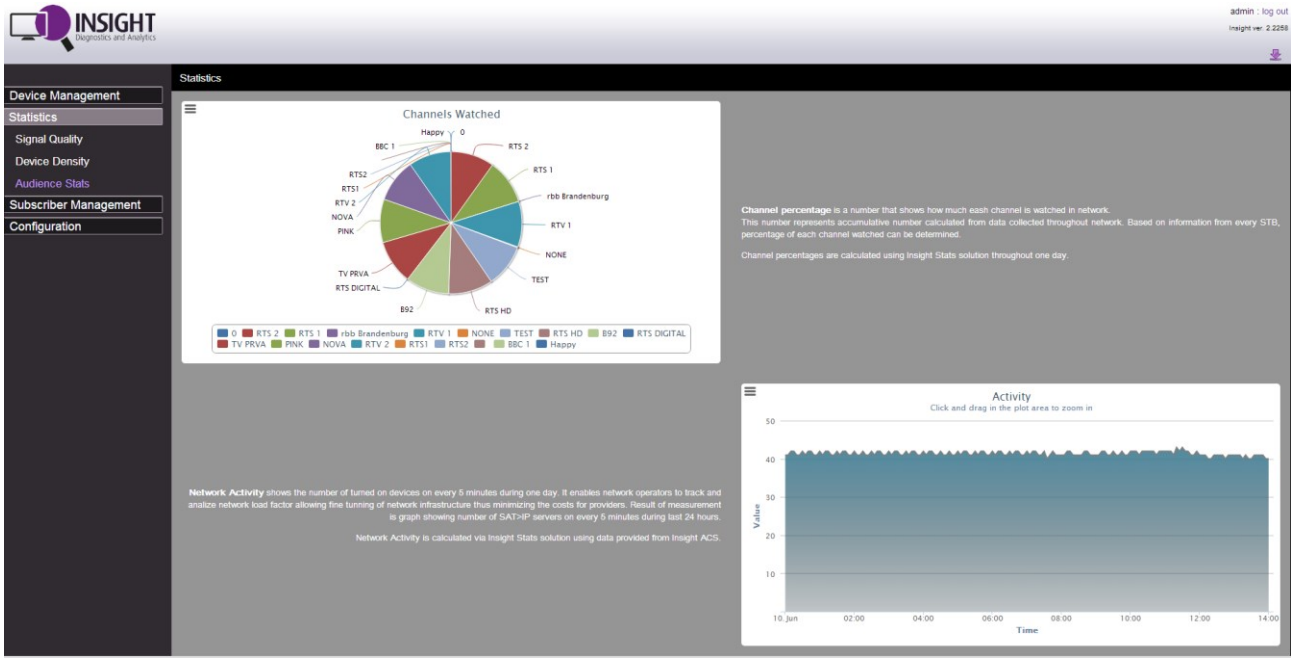
Slika 17. Mobilna aplikacija kao korisnik API.

Akcione skripte je moguće definisati u spoljašnjoj aplikaciji, posle čega se one skladište na Insight ACS poslužiocu i može ih koristiti druga aplikacija. Primer jednog toka podataka i rada sa akcionim skriptama korišćenjem modula aplikativnog sprežnog sloja u Insight ACS poslužiocu je prikazana na Slici 18.



Slika 18. Protok podataka vezanih za akcione skripte kroz API module.

Insight Stats je namenski prezentacioni poslužilac koji preko WSDL API modula prikuplja Big Data podatke i na osnovu njih pravi analizu gledanosti TV programa na TV-zasnovanim uređajima. Grafički prikaz obrađenih Big Data podataka u Insight Stats poslužiocu je prikazan na Slici 19.



Slika 19. Obrađeni Big Data podaci dostavljeni preko WSDL API modula.

5.1 Vremenski odziv API modula

Kvalitet sprežnih slojeva je direktno srazmeran brzini kojom odgovaraju na pozive operacija u njima. U različitim modulima sprežnog sloja Insight ACS poslužioca su korišćeni različiti fizički protokoli, pa moduli vremenski različito reaguju.

Uporedno ispitivanje API modula je rađeno kroz dva ispitna slučaja. Prvi slučaj pokazuje brzinu odziva na jednostavnu funkciju koja radi sa parametrima na jednom uređaju. Svi moduli aplikativnih slojeva koriste istu funkcionalnost jezgra poslužioca kako bi operaciju izvršili čime se izbegavaju razlike usled heterogenosti sistema. Moduli su ispitivani korišćenjem web aplikacije, mobilne aplikacije i poslužioca za Big Data podatke. Svi klijenti API modula su smešteni u istu spoljašnju mrežu.

Vreme potrebno za odgovor na poziv operacije iz API modula se može podeliti u nekoliko segmenata. Prvi je vreme potrebno za autorizovanje korisnika operacije. Ovo vreme se razlikuje u svim sprežnim modulima s obzirom da je kontrola pristupa distribuirana i realizovana različitim alatima. S obzirom da se isti poziv ka jezgru poslužioca obavlja, vreme obavljanja operacije je isto za svaki sloj. Sledeće kašnjenje unosi postprocesiranje podataka. Moduli pripremaju podatke kroz povratne vrednosti, ukoliko postoje. Ovo kašnjenje ne postoji ukoliko

operacija nema povratnu vrednost. Poslednje i najveće kašnjenje dolazi od samog fizičkog protokola korišćenog za komunikaciju. Ovo kašnjenje ne dolazi direktno od samog aplikativnog sloja već zavisi od mrežnih faktora (spoljašnja ili unutrašnja mreža) i tipa ISO OSI transportnog protokola korišćenog za prenos.

Platforma na kojoj je poslužilac bio postavljen ima procesor Intel Core i5, 4 GB RAM memorije i 500 GB HDD. Uporedno ispitivanje je obavljeno sa dva tipa operacija:

1. Operacije sa povratnom vrednošću (operacijom dobavljanja vrednosti parametara - *GetParameterValue*),
2. Operacije bez povratne vrednosti (operacijom osvežavanja vrednosti parametra - *Reload)*.

Oba ispitna slučaja su rađena nad istim skupom parametara (u modelu podataka je to *Device.DeviceInfo.*) i nad istim uređajem. Ispitivanje je obavljeno sa tri ponavljanja nad svakim parametrom sa svakim modulom i svakim uređajem, što podrazumeva *16 parametara u podstablu po 3 ponavljanja u 3 modula* ili zbirno *144 ispitna slučaja*.

Tačke merenja su postavljene u kodu na početku izvršenja operacija u aplikativnom sprežnom sloju i na pozivu isporuke povratne vrednosti.

Usrednjene vrednosti ispitivanja su prikazane u Tabela 9.

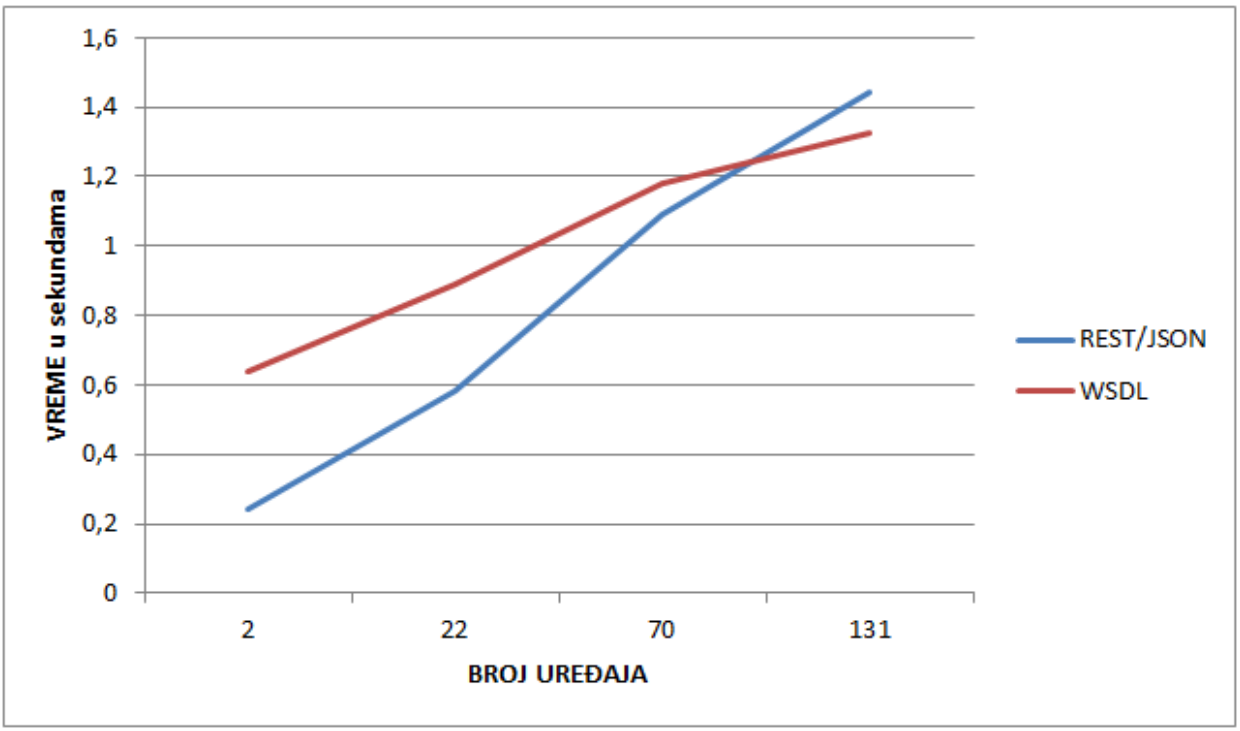
API modul	Autorizacija	Izvršenje operacije	Postprocesiranje	Ukupno	Usporenje API
GetParameterValue					
RPC API	11 ms	338 ms	32 ms	381 ms	12%
REST API	7 ms	338 ms	26 ms	371 ms	8,8%
WSDL API	31 ms	338 ms	32 ms	401 ms	15,7%
Reload					
RPC API	11 ms	224 ms	0 ms	235 ms	4,6%
REST API	7 ms	224 ms	0 ms	231 ms	3%
WSDL API	31 ms	224 ms	0 ms	255 ms	12,15%

Tabela 9. Uporedno ispitivanje vremena odziva API modula.

Iz rezultata se može primetiti da RPC i REST API reaguju u približnom vremenskom okviru dok WSDL API unosi zakašnjenje mahom zbog autorizacionog mehanizma. Ispitnim slučajevima je utvrđeno da je udeo aplikativnog sprežnog sloja u propagaciji podataka od 3% za REST API u operaciji bez povratne vrednosti do 15,7% za WSDL API za operacije sa povratnim vrednostima.

WSDL mehanizam donosi prednost kada se radi o velikim podacima. Ta prednost potiče iz mogućnosti SOAP poruka da prenose veliku količinu strukturiranih podataka. SOAP format poruka unosi kašnjenje i višak u komunikaciji pre svega zbog svoje struktura i veće količine kontrolnih informacija u odnosu na REST/JSON poruke. Kako broj podataka raste, udeo viška u WSDL-zasnovanoj komunikaciji na SOAP porukama procentualno opada i njene prednosti dolaze do izražaja, gde REST-JSON počinje da posustaje zbog nestrukturirane prirode podataka i obaveze dodatne obrade na klijentskoj strani.

Poređenje vremenskog odziva REST i WSDL modula sprežnog sloja je obavljeno korišćenjem metode za dobavljanje podataka o kvalitetu signala u određenom geografskom prostoru (*getSNRValuesGEO*). Ispitivanje je obavljano sa *4 ispitna slučaja* (2, 22, 70 i 131 krajnji uređaj) sa *36 ponavljanja* po svakom. Usrednjeni rezultati su prikazani na Slici 20.



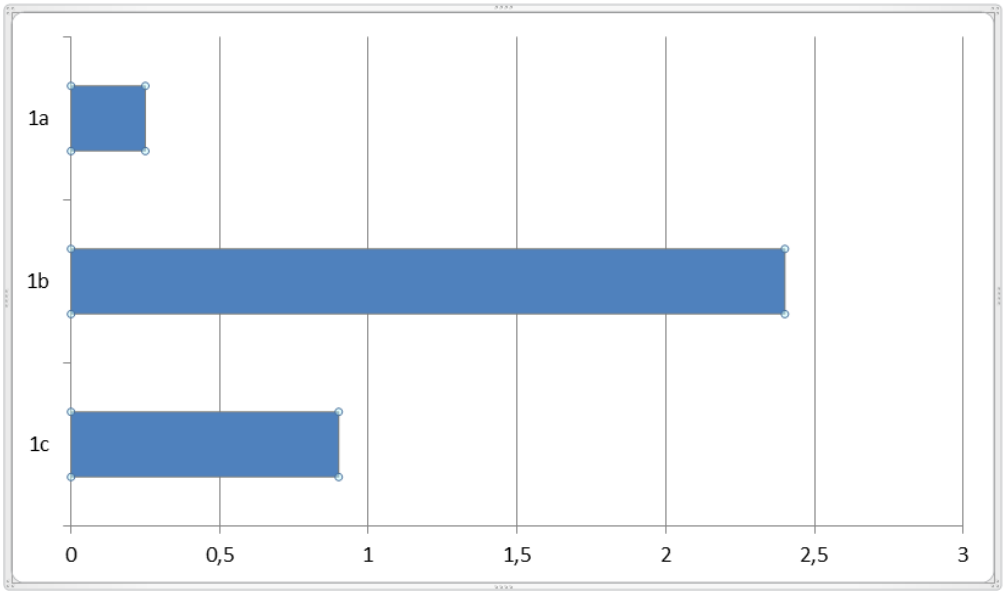
Slika 20. Uporedno ispitivanje odziva WSDL i REST/JSON modula sprežnoj sloja.

U poslužiocu sa 131 krajnjim uređajem kompletno prenošenje podataka o kvalitetu signala u određenom geografskom prostoru uz propagiranje kroz mrežu i ostale zadrške zahteva 9% manje vremena nego korišćenjem REST API modula i iznosi 1,326 sec.

Sistem je ispitivan i slanjem komandi krajnjim uređajima čime su se dobili rezultati koliko vremena zahteva obrada (Insight ACS jezgro), prijem i slanje (API sloj) sa jedne i propagaciju kroz mrežu i izvršenje komande na samom uređaju sa druge strane. Ispitni scenario je podrazumevao korišćenje web aplikacije sa koje se menjao trenutno aktivni servis na uređaju sa TV funkcionalnošću. Ispitivanje je obavljeno korišćenjem *metode crne kutije* [51] sa merenjem vremenskih trenutaka (*timestamp*) u aplikativnom sprežnom sloju (na poziv operacija u

sprežnom sloju i pri izdavanju komande pre uređaju) i na krajnjem uređaju (prihvatanje komande i vizuelno promene kanala), dok je propagiranje kroz mrežu približno određeno.

Ispitivanje je ponovljeno 48 puta sa 3 različita krajnja uređaja, što je ukupno *144* ponavljanja. Rezultati ovog ispitivanja su prikazani na Slici 21.



Slika 21. Ispitivanje odziva celog sistema na izdavanje komande u sekundama.

Na (1a) je prikazano vreme koje je potrebno za prijem, obradu i slanje komande (Insight ACS i API) izmereno u programskoj podršci, na (1b) je prikazano vreme koje je potrebno da podaci stignu do krajnjeg uređaja (mrežna propagacija) a (1c) pokazuje vreme potrebno za izvršenje promene programa na samom uređaju (*process and execution*).

Prosečno vreme za (1a) je tek 6% od ukupnog vremena koje protekne od izdavanja komande do izvršenja na samom uređaju. Ako uračunamo rezultate prethodnog ispitivanja gde od tog vremena (1a) prijem i slanje poruke kroz API sloj (Tabela 9) potroši prosečno 9,37%, dolazimo do zaključka da API slojevi ukupno u celom procesu od korisnika do uređaja (*user-to-device*) učestvuju tek 0,56% ukupnog vremena što je prihvatljivo za upravljanje i nadgledanje uređaja u IoT ekosistemima u poređenju sa drugim poslužiocima [52] [53].

5.2 Metrike koda

Kvalitet svake programske podrške se može kvantifikovati metričkim parametrima. Neki od poznatijih metričkih parametara za analizu programske podrške su:

- DSQI (Design Structure Quality Index)
- Ciklična kompleksnost (McCabe-ov indeks)
- Halsteadov metrički indeks
- Robert Cecil Martinov metrički indeks

- CISQ indeks

Jedna od najbitnijih metrika API modula generalno je ciklična kompleksnost iskazana kroz McCabe-ov indeks. On pokazuje koliko grana svaka metoda u određenom modulu ima, što za module koji čine API sloj podrazumeva koliko se dodatno tok obrade podataka ispituje i zadržava u njima. Svaka petlja (if, for, while, case), svaki logički operator i svaki izuzetak u kodu povećavaju indeks.

McCabe-ov indeks je meren korišćenjem alata Sonar i rezultati po pojedinačnim modulima su prikazani u Tabela 10.

Modul	McCabe indeks po metodi
CpeManagementServiceImpl	3,3
ConfigurationServiceImpl	2,4
StatsServiceImpl	1,1
AndroidServices	6,6
ActionScriptServices	4
NBIServiceBean	4,9

Tabela 10. Ciklična metrika koda.

Ne postoji zvanična granica za vrednosti ovog indeksa, a njegova vrednost tipično varira u zavisnosti od implementirane funkcionalnosti. Za slojeve koji su u direktnom kontaktu sa spoljašnjim aplikacijama se zbog brzine preporučuje da ovaj broj ne prelazi 10 [54], što je ispoštovano u svakom modulu.

Važna metrika koda je indeks održavanosti koda (*Maintenance Index, MI*) koji nema opšte prihvaćenu formulu po kojoj se računa. MI se obično kvalifikuje opisno a ne brojem, pa tako programska podrška može biti lako održiva ili teško održiva. API sloj je građen modularno i skalabilno tako da bilo kakve promene se mogu lako uraditi. API ne dopušta promene koda u toku rada poslužioca, pa je tako za svake promene u pristupnim slojevima obaveza ponovno pokretanje poslužioca, pa tako API se može oceniti sa osrednjim MI indeksom.

Indeks korisnosti programske podrške (*Usability Index, UI*) [55] se kod aplikativnih sprežnih slojeva meri kao odnos vremena koje je potrebno za povezivanje sa softverskim entitetom (poslužiocem) preko pristupnih slojevima i bez korišćenja alata pruženih pristupnim slojevima. Indeks korisnosti za korišćenje WSDL i GWT RPC tehnologija se kreće od 3 do 4,5 [56] što znači da korišćenje ovih mehanizama skraćuje vreme implementacije korisničkih aplikacija.

6. Zaključak

U radu je projektovan i implementiran aplikativni sprežni sloj koji omogućuje razmenu podataka sa različitim tipovima korisnika korišćenjem prilagođenih modula čime se ostvaruju visoke performanse. Moduli su realizovani tako da podrže širok spektar krajnjih korisnika poslužioca, čime su omogućili da poslužilac ima podršku i za mobilne uređaja ali i za prenos Big Data podataka u okviru jednog aplikativnog sloja, što nije prisutno u dosadašnjim rešenjima (datim u 2.6 ne podržavaju). Zahtevi ovog programskog rešenja su ispunjeni i time su otvorene nove mogućnosti za dalja unapređenja i proširenja. Fokus daljeg rada će biti integrisanje dodatnih bezbednosnih mehanizama u sloj koji prenosi Big Data podatke, kao i optimizacija drugih slojeva. U okviru svih slojeva će biti uveden jedinstveni autentikacioni mehanizam i biće dostupne nove funkcionalni za specifične tipove uređaja.

7. Literatura

[1] Understanding The Protocols Behind The Internet of Things, Stan Schneider, Electronic Design, electronicdesign.com/embedded/understanding-protocols-behind-internet-things, 2013.

[2] "What is the Internet of Things?", Jimmy Daly, Statetech magazine, www.statetechmagazine.com/article/2013/08/what-is-the-internet-of-things, 2013.

[3] The Internet of Things – How the Next Evolution of the Internet is Changing Everything, Dave Evans, Cisco Corporation, 2011.

[4] Internet of Things (IoT): A vision, architectural elements, and future directions, J. Gubbi et al, Science Direct Magazine, 2013.

[5] Architecture for the Internet of Things (IoT): API and Interconnect, I. Gronbaek, SENSORCOMM, Second International Conference, 2008.

[6] Internet of Things (IoT), Margaret Rouse, Tachtarget mahazine, whatis.techtarget.com/definition/Internet-of-Things , 2013.

[7] IoT: Architecture, Protocols and Services, Chonggang Wang et al, IEEE Sensors Journal, Vol 13. No 10., 2013.

[8] Internet of Intelligent Things: Bringing Artificial Intelligence into Things and Communication Networks, Artur Arsenio et al, Inter-cooperative Collective Intelligence: Techniques and Applications Studies in Computational Intelligence Volume 495, pp 1-37, 2014.

[9] IoT Startegic Research Roadmap, O.Vermesan, P. Friess, Internet of Things - Global Technological and Societal Trends, 2011.

[10] ProSyst, www.prosyst.com, 2014.

[11] Nimbits, www.nimbits.com, 2014.

[12] LWM2M, www.slideshare.net/zdshelby/oma-lightweightm2-mtutorial, 2014.

[13] Arkessa, www.arkessa.com, 2014.

[14] Xively, www.xively.com, 2014.

[15] Etherios, www.etherios.com, 2014.

[16] Internet of Things requirements and protocols, Kim Rowe, Embedded Computing Design 2014, embedded-computing.com/articles/internet-things-requirements-protocols, 2014.

[17] CoAP protokol, datatracker.ietf.org/doc/draft-ietf-core-coap/?include_text=1, 2014.

[18] Continua protokol, www.continuaalliance.org/about-continua, 2014.

[19] DDS protokol, portals.omg.org/dds/, 2014.

[20] Architectural Styles and the Design of Network-based Software Architectures, Roy Thomas Fielding, University of California, 2000.

[21] MQTT protokol, mqtt.org, 2014.

[22] UpnP Device Architecture 1.0, http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf, 2014.

[23] ZeroMQ protokol, zeromq.org, 2014.

[24] XMPP standard, xmpp.org, 2014.

[25] An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, D. Harrington et al, IETF, 2002.

[26] DPWS Specification Document, docs.oasis-open.org/ws-dd/ns/dpws/2009/01, 2009.

[27] WSDL 2.0 core language, www.w3.org/TR/2007/REC-wsdl20-20070626/, 2014.

[28] SOAP Specification, www.w3.org/TR/soap/, 2014.

[29] TR-069 protocol, www.broadband-forum.org/technical/download/TR-069.pdf, 2014.

[30] The cloud vs on-premises software, Ian Gotts, The Next Web, thenextweb.com/insider/2013/11/25/cloud-vs-premises-software-need-consider-business, 2014.

[31] Comparing Public, Private, and Hybrid Cloud Computing Options, Judith Hurwity et al, Cloud Computing For Dummies, edition 2013.

[32] Shaping Future Service Environments with the Cloud and Internet of Things: Networking Challenges and Service Evolution, Gyu Myoung Lee, Noel Crespi, Leveraging Applications of Formal Methods, Verification, and Validation Lecture Notes in Computer Science Volume 6415, pp 399-410, 2010.

[33] Data, data everywhere, The Economist magazine, 2010.

[34] Service-Generated Big Data and Big Data-as-a-Service: An Overview, Zibin Zheng et al, IEEE Big Data (BigData Congress), 2013.

[35] Towards a Quality-centric Big Data Architecture for Federated Sensor Services, Ramaswamy et al, IEEE Big Data (BigData Congress), 2013.

[36] MyJohnDeere agricultural management system, www.myjohndeere.com, 2014.

[37] Understanding Cloud APIs, and Why They Matter, Bill Kleyman, Data Knowledge Center, www.datacenterknowledge.com/archives/2012/10/16/understanding-cloud-integration-a-look-at-apis, 2012.

[38] Open APIs reach new high water mark as the Web evolves, Dion Hinchcliffe, Enterprise Web 2.0, www.zdnet.com/blog/hinchcliffe/open-apis-reach-new-high-water-mark-as-the-web-evolves/215 , 2008.

[39] 97 Cloud APIs: Twilio, Google App Engine and Tropo, Wendell Santos, Programmable Web, 2012. https://thingspeak.com/

[40] Towards a Cross Platform Cloud API, Dana Petcu, Ciprian Craciun, Massimiliano Rak, 1st International Conference on Cloud Computing and Services Science, Holandija, 2011.

[41] Towards Cross-Platform Cloud Computing, Magdalena Slawinska, Jaroslaw Slawinski, Vaidy Sunderam, Euro-par 2011: Parallel Processing Workshops, Computer Science Volume 7155, 2012, pp 5-14.

[42] Amazon Cloud, www.amazon.com/clouddrive, 2014.

[43] Alcatel Lucent Motive Home Device Manager, www.alcatel-lucent.com/products/motive-home-device-manager, 2014.

[44] Control 4, 4Sight Anywhere Access, www.control4.com, 2014.

[45] Axiros Axess, http://axiros.com/offerings/competenceintr-069andbeyond/axessacs-tr-069-autoconfiguration-server.html, 2014.

[46] Dimark Server, www.dimark.com/tr-069_acs.html, 2014.

[47] ThinkSpeak IoT Server, https://thingspeak.com/, 2014.

[48] Secure communication for smart IoT objects: Protocol stacks, use cases and practical examples, R. Bonetto et al, IEEE Symposium on WoWMoM, 2012.

[49] Security architecture elements for IoT enabled automation networks, K. Fischer, J. Gesner, IEEE ETFA Conference, 2012.

[50] A Lightweight Multicast Authentication Mechanism for Small Scale IoT Applications, Xuanxia Yao et al, IEEE Sensors Journal, Vol 13. 2013.

[51] Standard for Software Component Testing, BCS SIGIST, 2001.

[52] Architecture and measured characteristics of a cloud based internet of things, G. Fox, R. Hartman, International Conference on Collaboration Technologies and Systems (CTS), 2012.

[53] IoT architecture to enable intercommunication through REST API and UPnP using IP, ZigBee and arduino, IEEE Conference on WiMob, 2013.

[54] Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, Arthur Watson, Thomas McCabe, 1996.

[55] A proposed index of usability: A method for comparing the relative usability of different software systems, Han Lin et al, IEEE Behaviour and Information Technology, Vol 16, 1997.

[56] Usability evaluation for enterprise SOA APIs, J. Beaton et al, 2nd international workshop on Systems development in SOA environments, 2008.