УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Немања Игњатов

Реализација окружења за аутоматско конфигурисање уређаја коришћењем TR-069 протокола заснованог на скрипт језику Луа

МАСТЕР РАД

Нови Сад, 2014

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**:		**Немања Игњатов**
Ментор, **МН**:		**др Милан Бјелица**
Наслов рада, **НР**:		**Реализација окружења за аутоматско конфигурисање уређаја коришћењем TP-069 протокола заснованог на скрипт језику Луа**
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)		**7/68/27/0/13/29/0/0**
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кqучне речи, **ПО**:		**аутономни системи,аутоматско-конфигурисање уређаја,TP-069,Луа скрипт језик**
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		**У раду је реализовано решење за подршку аутоматском конфигурисању великог броја уређаја у мрежи. Имплементирани систем омогућава дефинисање конфигурационих скриптова током рада система и примену конфигурације на појединачне уређаје или групу уређаја груписаних по одређеном критеријуму. Решење се ослања на функционалности омогућене TP-069 прокотолом и скрипт језиком Луа. Реализовани систем је независан од типа уређаја на који се примењује конфигурација. Доприноси овог рада су: евалуација коришћења Луа скрипт језика у системима за аутоматску конфигурацију уређаја, као и дефинисање модела догађаја као иницијатора извршавања конфигурационог скрипта.**
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**:		**Nemanja Ignjatov**	
Mentor, **MN**:		**Milan Bjelica, PhD**	
Title, **TI**:		**Implementation of framework for auto-configuration using TR-069 protocol, based on Lua skript language**	
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)		**7/68/27/0/13/29/0/0**	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		**Autonomic systems, auto-configuration,TR-069, Lua script language**	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		**This paper describes implementation of a framework for the auto-configuration based on TR-069 protocol and Lua script language. System described in this paper enables definition of configuration scripts during the system runtime and ability to apply defined configuration to particular devices or to the group of devices created using some predefined criteria. This solution is based on the functionalities provided by TR-069 protocol and Lua script language. Implemented system is agnostic of the type of device which is being configured using Lua script. Contributions of this paper are: evaluation of usage of the Lua script language in auto-configuration systems, and definition of the event-driven model, which is being used as an iniciator of the configuration script execution.**	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	**Ištvan Papp,PhD**	
,	Member:	**Rastislav Sthruharik, PhD**	Menthor's sign
	Member, Mentor:	**Milan Bjelica,PhD**	

Zahvalnost

Zahvaljujem se porodici i prijateljima na podršci tokom školovanja i tokom izrade ovog rada.

Zahvaljujem se kolegama iz tima na stvaranju pozitivne atmosfere tokom izrade ovog rada.

Zahvaljujem se mentoru doc. dr Milanu Bjelici na pomoći prilikom osmišljavanja rešenja.

Zahvaljujem se kolegama Saši Radovanoviću na pomoći pri izradi rešenja i Mići Ćetkoviću na utrošenom vremenu na savete i sugestije.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

ACS – *Auto-Configuration Server* – Poslužilac za automatsku konfiguraciju

API - *Application Programming Interface,* Sprežni pristupni sloj

ASN.1 – *Abstract Syntax Notation One* – Skup pravila za prezentacije, enkodovanje, dekodovanje i prenos podataka u mreži

CHOP – *Configure, Heal, Optimize and Protect* – Funkcionalnosti autonomnih sistema

CPE – *Customer Premises Equipment* – Krajnji uređaj u mreži

CRUD - *Create Read Update Delete* – operacije nad strukturama podataka

DDoS – *Distributed Denial of Service* – Tip napada računara preko mreže

EJB – *Enterprise JavaBeans* - Biblioteka za razvoj web aplikacija

FIFO - *First In First Out* – struktura podataka tipa red

formata podataka u elektronskoj formi.

GWT – *Google Web Toolkit* – Skup alata za razvoj web aplikacija

HTTP - *HyperText Transfer Protocol* – Protokol za prenos hiperteksta

HTTPMU - *HyperText Transfer Protocol Multicast* – Protokol za prenos hiperteksta između više uređaja

HTTPU - *HyperText Transfer Protocol Unicast* – Protokol za prenos hiperteksta između dva uređaja

IoT – *Internet of Things* – Internet stvari

ISO – *International Organization for Standardiozation* – Međunarodna organizacija za standardizaciju

JSON – *JavaScript Object Notation* – Standardni skup pravila za definisanje podataka u elektronskoj formi

MIB – *Management Information Base* – Baza podataka upravljačke jedinice kod SNMP prtokola

NVRAM – *Non-volatile Random Access Memory* – Operativna memorija nepromenljivog sadržaja
objekata podataka

OSI – *Open System Interconnection* – Referencntni model za otvoreno povezivanje sistema

REST – *Reprensational State Tranfer* – Skup pravila za prenos informacija korišćenjem HTTP protokola

RPC - *Remote Procedure Call* – poziv udaljene metode

SMI – *Structure of Management Information* – Podaci o krajnjem uređaju kod SNMP protokola

SNMP – *Simple Network Management Protocol* - Mrežni protokol

SOAP – *Single Object Access Protocol* - Komunikacioni protokol za razmenu

SQL – *Structured Query Language* – Programski jezik za pristup bazi podataka

SSDP – *Simple Service Discovery Protocol* – Mrežni protokol za oglašavanje i otkrivanje funkcionalnosti uređaja

SSL – *Secure Socket Layer* – Sigurnosni protokol

STB – *Set-Top Box* – Digitalni prijemnik televizijskog signala

TCP/IP – *Transport Control Protocol / Internet Protocol* – Skup protokola za upravljanje prenosom podataka na Internetu

TLS – *Transport Layer Security* – Sigurnosni protokol

TR069 – *Technical Report 069* – Mrežni protokol za upravljanje i nadzor krajnjih uređaja

UDP – *User Datagram Protocol* – Mrežni protokol transportnog sloja ISO/OSI protokol steka

UPnP – *Universal Plug and Play* – Mrežni protokol

WSDL – *Web Services Description Language* – Jezik za opis sprega u web servisu

XML - *Extensible Markup Language* – Standardni skup pravila za definisanje

1. Uvod

U današnje vreme broj korisničkih uređaja povezanih na Internet ubrzano raste. Računarski sistemi koji uključuju ove uređaje postaju sve kompleksniji, stoga upravljanje krajnjim uređajem postaje otežano. Predviđanja pokazuju da će broj korisničkih uređaja rasti stopom od 38 procenata godišnje i da složenost prosečnog uređaja raste. Trenutno, funkcionisanje računarskih sistema sa uređajima sa ovakvom složenošću nadzire i njima upravlja veliki broj stručnih ljudi iz oblasti informatike, ali broj potrebnih stručnjaka u ovoj oblasti već premašuje broj postojećih [1]. Stoga udaljeno upravljanje, nadzor, povećanje kvaliteta usluga i oporavak uređaja posle greške u radu predstavljaju jedne od osnovnih ciljeva svetskih kompanija iz oblasti distribucije televizijskog signala i multimedijalnih sadržaja, kao i omogućavanje pristupa Internetu krajnjim korisnicima. Ključnu ulogu u ispunjenju gore navedenih ciljeva igra operater računarskog sistema, zaposleni u korisničkoj podršci, koji preduzima korektivne mere na zahteve korisnika. Takođe, ispunjenje ovih ciljeva omogućava veće zadovoljstvo korisnika, što uzrokuje i proširenje tržišta. Zbog rasta kompleksnosti sistema, uloga korisničke podrške postaje umanjena i sklona greškama usled nedostatka alata koji bi im omogućili lakši, precizniji i efikasniji rad.

Faktori koji utiču na rast složenosti sistema su [2]:

- Raznolikost korisničkih uređaja;
- Integracija računarskih mreža i sistema;
- Nepredvidivost u opterećenju komunikacionih puteva u sistemu.

Vremenski period u kome se predviđa rešavanje korisničkih problema, koji po *BenchmarkPortal*-u [3] iznosi 5,97 minuta, ne ostavlja puno vremena operateru da ustanovi uzrok greške u sistemu, već se poseže za radikalnim merama za oporavak uređaja, kao što je ponovno pokretanje uređaja. Sa druge strane, korisnik očekuje pružanje usluge bez kvarova u sistemu i cela kompleksnost sistema mora biti nevidljiva za korisnika. Sistemi skloni kvarovima

usled svoje složenosti često bivaju napušteni od strane korisnika ukoliko se ti kvarovi ne rešavaju brzo i efikasno, a pritom bez napora na strani korisnika.

Potražnja za ispunjenjem gore pomenutih zahteva iziskuje istraživanja u cilju stvaranja autonomnih računarskih sistema i inkorporiranje korisničkih uređaja u iste. Autonomni računarski sistemi se zasnivaju na pojmu samoupravljanja (*Self-management)* uređaja, gde se aktivnost potrebna za upravljanje sistemom zasniva na komunikaciji između uređaja u sistemu i složenim algoritmima za definisanje stanja sistema i preduzimanje daljih koraka u rešavanju potencijalnih problema. Pojam samoupravljanja uređaja sakriva kompleksnost sistema od korisnika i od operatera i u idealnom slučaju isključuje ljudski faktor u rešavanju problema u sistemu.

Razvojem autonomnih računarskih sistema bi se omogućilo da performanse korisničkih uređaja budu veće i uvek u skladu sa trenutnim uslovima, a funkcionalnosti otpornije na kvarove u sistemu.

1.1 Koncept autonomnih računarskih sistema

Koncept autonomnog računarskog sistema se oslanja na karakteristike distribuiranih računarskih resursa, koje omogućavaju automatsko adaptiranje na nepredvidive okolnosti, istovremeno sakrivajući kompleksnost sistema od stranih faktora, kao što su operateri i korisnici[4]. Potreba za razvojem ovakvih sistema javila se početkom 21. veka usled ubrzanog rasta složenosti računarskih sistema i nemogućnosti za pružanjem novih usluga korisnicima, uz istovremeno očuvanje kvaliteta postojećih usluga.

Sistem ovog tipa pravi sopstvene odluke koristeći unapred definisanu logiku i uslove za donošenje odluka. Provere i optimizacija stanja sistema se vrše konstantno, kao i prilagođavanje novim uslovima rada. Autonomni sistem se sastoji od autonomnih komponenti koje komuniciraju između sebe i na osnovu razmenjenjih informacija donose zaključke koje korektivne mere treba preduzeti radi poboljšanja rada sistema.

Samoupravljanje uređajem je osnovni pojam u definisanju autonomnog sistema i omogućava CHOP (*Configuring, Healing, Optimizing and Protecting*) skup funkcionalnosti. Samouporavljanje se odnosi na 4 stvari koje su prikazane na Slici 1:

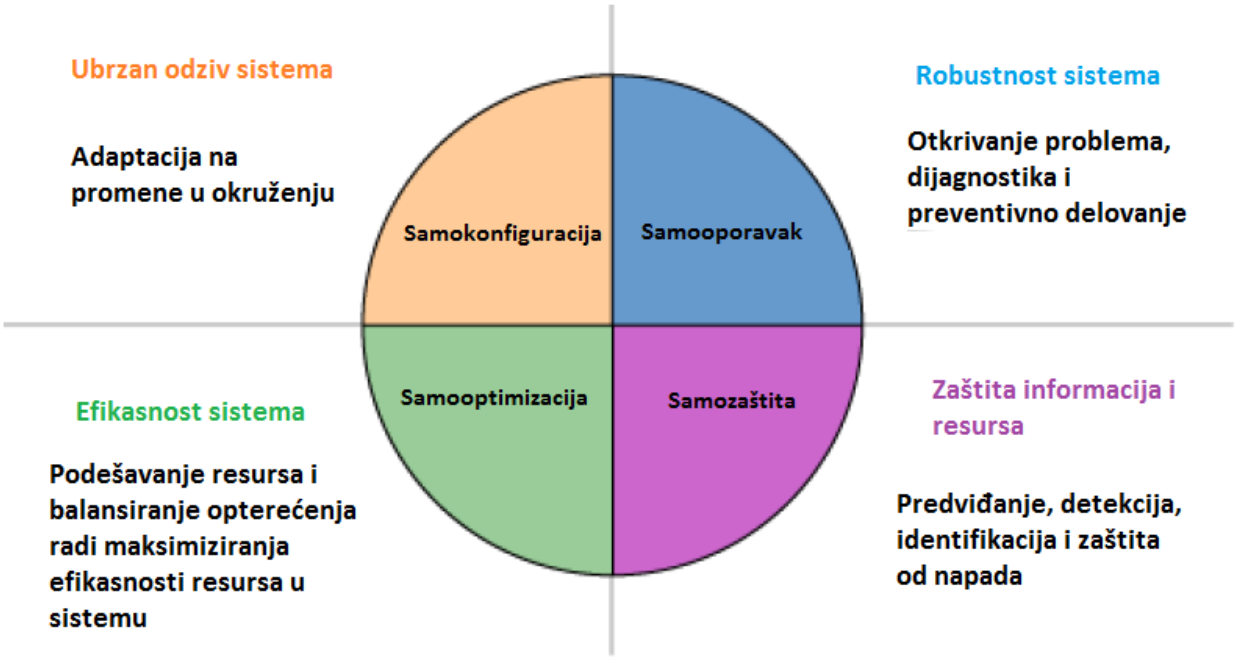
- Samokonfiguraciju;
- Samooporavak;
- Samooptimizaciju;
- Samozaštitu.

Samokonfiguracija se odnosi na sposobnost sistema da dinamički konfiguriše svoje komponente. Sistem se automatski adaptira promenama u okruženju, uz minimalnu intervenciju čoveka. Takve promene se najčešće odnose na dodavanje ili uklanjanje komponenata u sistemu[5].

Samooporavak se zasniva da detektovanju problematičnih operacija u sistemu, neočekivanog ponašanja komponenti u sistemu, dijagnostifikovanju problema i pokretanju korektivnih akcija radi povratka sistema u stabilno stanje[5].

Samooptimizacija se odnosi na sposobnost sistema da izvrši male izmene u podešavanjima resursa da bi se maksimalno iskoristile njihove mogućnosti. Zahtev prilikom samooptimizacije je da se ispune zahtevi krajnjih korisnika u vidu poboljšanja kvaliteta pruženih usluga uz očuvanje stabilnosti sistema[5].

Samozaštita je osobina sistema da predvidi, detektuje, identifikuje i zaštiti sistem od mogućih pretnji spolja. Ove pretnje mogu biti u vidu neovlašćenog pristupa sistemu, napadu štetnog softvera, DDoS napad, ili druge koje uzrokuju kvar u sistemu. Procedure zaštite sistema moraju biti tako realizovane da ukoliko se dogodi neočekivano ponašanje u sistemu, štetnost tog ponašanja bude smanjena[5].



Slika 1. Pregled osobina autonomnih sistema[5]

1.2 Cilj rada i doprinosi

Cilj ovog rada je impementacija programskog okvira koji omogućava definisanje algoritama za samoupravljanje sistemom, pravila za sprovođenje tih algoritama, kao i delove sistema na koje će se algoritmi sprovoditi. Zahtevi koje sistem mora ispuniti su:

- Mogućnost primene na različite tipove uređaja bez promene u samoj implementaciji okvira;
- Zaštita podataka koji se prenose u sistemu;
- Mogućnost upravljanja konfiguracijom uređaja;
- Mogućnost nadzora i poboljšanje perfomansi sistema;
- Upravljanje programskom podrškom uređajima u sistemu;
- Mogućnost primene korektivnih mera u slučaju neispravnog funkcionisanja sistema.

Samoupravljanje sistemom se zasniva na automatskoj konfiguraciji uređaja kao i konstantnim nadzorom parametara ispravnosti funkcionisanja uređaja. Automatska konfiguracija se odvija u dva slučaja:

- Inicijalna konfiguracija prilikom prijave uređaja u sistem;
- Rekonfiguracija u slučaju neočekivane vrednosti parametara u sistemu.

Nadzor parametara se oslanja sa nivoe notifikacije koji se odnose na brzinu i učestalost prosleđivanja vrednosti parametara od strane uređaja. Nivoi notifikacije se definišu prilikom inicijalne konfiguracije.

Za potrebe implementacije definisan je model događaja, na osnovu kog se može zaključiti u kom stanju se nalazi komponenta u sistemu i koji direktno ukazuje na moguće korektivne procedure koji bi se trebale sprovesti radi dovođenja sistema u funkcionalno stanje.

Realizacija konfiguracionih i korektivnih procedura je omogućena korišćenjem skript jezika Lua.

Važniji doprinosi ovog rada se odnose na implementaciju sistema za dinamičko konfigurisanje i upravljanje uređajima, definisanje modela događaja u sistemu, integracija interpretera za Lua skript jezik u sistem za automatsku konfiguraciju i nadzor uređaja kao i evaluacija izvodljivosti navedenih zahteva korišćenjem Lua skript jezika.

1.3 Organizacija rada

Ovaj rad je sačinjen od sedam poglavlja.

Drugo poglavlje izlaže teorijske osnove, pre svega automatske konfiguracije uređaja, korišćene protokole za konfiguraciju i nadzor rada uređaja, kao i upotrebu skript jezika za definisanje konfiguracije uređaja.

U trećem poglavlju dat je koncept rešenja za implementaciju funkcionalnosti potrebnih za realizaciju predstavljenog programskog okvira.

Četvrto poglavlje daje detaljan opis razvijenih softverskih komponenti.

U petom poglavlju su dati načini i rezultati ispitivanja i verifikacije funkcionalnosti realizovanog sistema.

Šesto poglavlje sadrži kratak pregled urađenog i moguće dalje pravce razvoja.

U sedmom poglavlju navedena je korišćena literatura.

2. Teorijske osnove

U ovom poglavlju date su teorijske osnove neophodne realizaciju programskog okvira za definisanje i izvršavanje automatske konfiguracije i kontrole uređaja. Ovo poglavlje takođe sadrži principe za automatsku konfiguraciju uređaja, korišćene protokole za tu svrhu, kao i mogućnosti za upotrebu skript jezika za definisanje konfiguracionih procedura.

2.1 Automatska konfiguracija i nadzor uređaja

Automatska konfiguracija uređaja predstavlja postupak definisanja rada i ponašanja uređaja u odnosu na njegovu ulogu u sistemu. U idealnom slučaju konfiguracija uređaja se izvršava bez potrebe za ljudskim naporom. Potražnja za automatskom konfiguracijom raste sa brojem uređaja zasnovanih na računaru, budući da velika većina njih ima pristup internetu. Jedan od uzroka potražnje je i složenost računarskih sistema, gde utrošak vremena potreban za definisanje i ispravku problema u sistemu ubrzano raste, stoga je potrebno umanjiti značak ljudskog faktora u održavanju sistema uz primenu mašinskih algoritama i procedura za upravljanje sistemom.

Po oblasti delovanja automatske konfiguracije možemo ih podeliti na:

- Lokalne;
- Globalne.

Lokalne se definišu na nivou uređaja i najčešće se nalaze zapisane u NVRAM memoriji[6] odakle bivaju učitane prilikom pokretanja i inicijalizacije uređaja. Mana lokalnih automatskih konfiguracija je što se ne menjaju vremenom, ne uzimaju u obzir promene nastale u okruženju uređaja i imaju ograničen opseg delovanja u slučaju kvara.

Globalne automatske konfiguracije izazivaju veće interesovanje u oblasti informacionih sistema i samim tim je izveden veći broj istraživanja na tu temu[7]. Većina sistema koji poseduju funkcionalnost automatske konfiguracije uređaja se zasniva na centralizovanom poslužiocu u

kome se nalazi logika i algoritmi za konfiguraciju uređaja. Važniji uslovi u ovakvim sistemima su:

- Nezavisnost sistema od tipova uređaja u njemu[8];
- Primenljivost konfiguracije za različite tipove uređaja;
- Proširivost oblasti u sistemu na koju se primenjuje konfiguracija;
- Definisanje mehanizama za objavu događaja i stanja na udaljenim uređajima[9];
- Zaštita podataka u prenosu između uređaja u sistemu;
- Mogućnost dinamičke promene konfiguracije tokom rada sistema.

Najveća prednost automatske konfiguracije na globalnom nivou je agregacija podataka sa velikog broja uređaja i mogućnost bolje identifikacije problema , stoga i nalaženje optimalnijih rešenja za tekuće događaje u sistemu[10]. U idealnom slučaju se konfiguracija i upravljanje uređajima sprovodi bez ljudskog faktora, međutim prilikom specifikacije sistema potrebno je omogućiti pristup operateru radi uvida u stanje sistema i mogućnost ručnog pokretanja određenih konfiguracija radi bolje dijagnostike rada sistema.

Nadzor uređaja se koristi pre svega radi poboljšanja pruženih usluga korisnicima. Najčešće se oblast nadzora funkcionalnosti definiše inicijalnom konfiguracijom uređaja, međutim nadzor se može vršiti manuelno od strane operatera dobavljanjem vrednosti parametara na uređaju koji su od važnosti. Stariji standardi se oslanjaju na dobavljanje vrednosti parametara u cilju saznanja o najnovijioj vrednosti tog parametra, na primer SNMP, opisan u Poglavlju 2.2.2., korišćenjem operacije *Get* i *GetNext,* dok noviji standardi, kao TR-069 opisan u Poglavlju 2.2.3., omogućavaju da uređaj automatski javlja najnovije vrednosti parametara od interesa, bez aktivnosti na strani poslužioca. Definisanje parametara od interesa se vrši prilikom konfiguracije uređaja. Neke oblasti računarskih sistema u kojima se godinama primenjuje nadzor su:

- Elektroenergetski sistemi;
- Multimedijalni sistemi[11];
- Mrežni sistemi[12];
- Sistemi u pametnim kućama[13].

2.2 Korišćeni protokoli

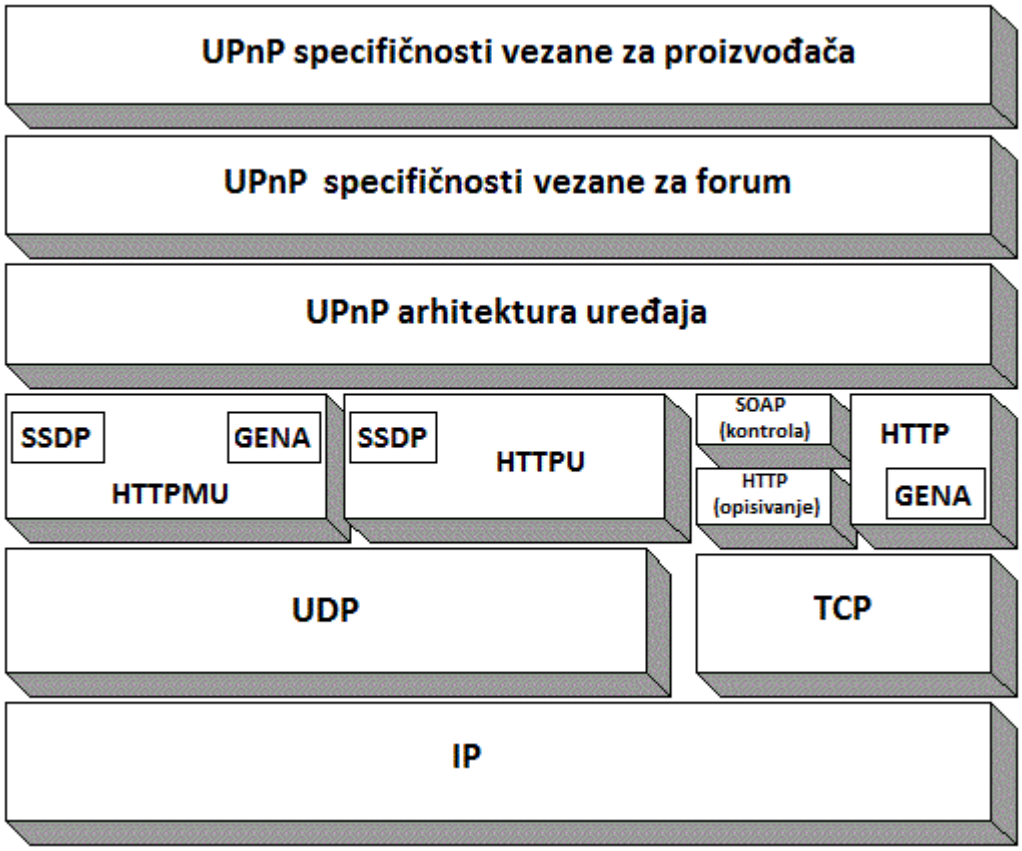
Potreba za automatskom konfiguracijom i nadzorom uređaja zahteva stvaranje standardizovanih načina za sprezanje udaljenih uređaja i centralnom poslužioca. U poslednjih 25 godina definisana je nekolicina standarda i protokola u tu svrhu. U ovom trenutku razvoj definisanje novih protokola ide u smeru rasta složenosti agenata na krajnjim uređajima u cilju rasterećenja poslužioca i smanjenja protoka podataka kroz mrežu. U narednim poglavljima opisani su najvažniji protokoli koji se koriste u sistemima za automatsku konfiguraciju uređaja.

2.2.1 Upravljanje uređajima zasnovano na UPnP

UPnP je skup standardan promovisan od strane UPnP foruma[14], definiše distribuirani računarski okvir koji obuhvata nekoliko protokola. Cilj UPnP standarda je da omogući međusobno otkrivanje uređaja u mreži, njihovo povezivanje, razmenu podržanih funkcionalnosti i podataka. U sklopu UPnP standarda definisani su protokoli koji omogućavaju:

- Upravljanje programskom podrškom uređaja;
- Upravljanje konfiguracijom uređaja;
- Dijagnostiku i održavanje funkcionalnosti uređaja;
- Nadzor performansi uređaja i protoka podataka u mreži.

UPnP protokol stek je prikazan na Slici 2.



Slika 2. UPnP protokol stek

Tri sloja na najvišem nivou protokol steka su povezana isključivo sa UPnP standardom. Sloj *UPnP arhitektura uređaja* definiše šablone za opis karakteristika datog uređaja. Sloj *UPnP specifičnosti vezane za forum* definiše pojedinosti koje se odnose na formate podataka koji se razmenjuju protokolima koji su definisani tim forumima. Sloj *UPnP specifičnosti vezane proizvođača* uređaja definiše proširenja shodno potrebama proizvođača, uz uvažavanje ograničenja određenih u sloju za arhitekturu uređaja.

Deo UPnP standarda koji se koristi za razmenju notifikacija o događajima na uređajima je GENA (*General Event Notification Architecture*). SOAP protokol se koristi za izvršavanje

kontrolnih procedura koje se sadrže u RPC porukama. SSDP protokol se koristi za otkrivanje uređaja u mreži koristeći HTTPMU ili HTTPU poruke.

Mogućnosti UPnP protokola se velike i široko primenjene u današnjim računarskim sistemima u različite svrhe. Nedostaci protokola zasnovanih na UPnP standardu su:

- Izmene u protokol steku koje vrši sam proizvođač uređaja, što se pokazalo kao potencijalno ugrožavanje sigurnosti sisitema;
- Nekorišćenje enkripcionih algoritama kao što su SSL ili TLS za zaštitu podataka koji se prenose između uređaja;
- Složenost korišćenja usled upotrebe velikog broja različitih protokola za korišćenje funkcionalnosti omogućenih UPnP standardom.

2.2.2 SNMP protokol

SNMP [15] je protokol na aplikativnom nivou korišćen za upravljanje mrežnim resursima. Osmišljen je za potrebe konfiguracije i lakšeg pronalaženja problema u radu uređaja u mreži, dok istovremeno smanjuje složenost upravljačkih mehanizama u mrežnim sistemima. Prvobitno je korišćen za upravljanje TCP/IP mrežama, dok je kasnije postao jedan od najkorišćenijih protokola za nadzor i upravljanje mrežnim sistemima. Jedna od najvećih prednosti koje je uveo SNMP protokol je interoperabilnost između uređaja različitih proizvođača, što je omogućeno ASN.1 šemom za enkodovanje poruka.

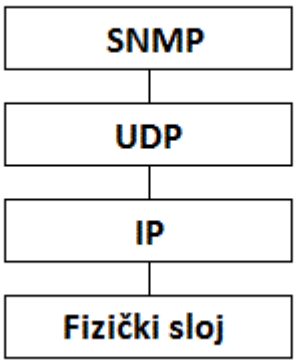
Tokom vremena definisane su tri verzije, njihove važnije karakteristike su prikazane u Tabeli 1:

SNMPV1	*SNMPV2*	*SNMPV3*
Jednostavna implementacija	Optimizacija perfomansi u odnosu na SNMPv1	Uvođenje algoritama za enkodovanje poruka
Upravljanje uređajima u mreži	Nove kontrolne poruke	Namenjen za nadzor i optimizaciju sistema
Manjak sigurnosnih mehanizama	Uvođenje sigurnosnih mehanizama	Rešeni problemi sa sigurnošću informacija
	Prisutni problemi sa sigurnošću	

Tabela 1. Prikaz važnijih karakteristika verzija SNMP standarda

Strukturu sistema zasnovanog na SNMP protokolu čine sledeće komponente:

- Struktura informacija za upravljanje (SMI - *Structure of Management Information*). Ovom strukturom se definišu tipovi podataka koji se smeju koristiti u bazi podataka.
- Upravljačka informaciona baza (MIB – *Management Information Base*) predstavlja skup informacija o uređajima u mreži. Informacije se odnose na objekte upravljanja u mreži, odnosno određene uređaje.
- SNMP agenti – softverska komponenta koju mora sadržati svaki uređaj koji podržava SNMP protokol. Svrha agenta je sprezanje sa upravljačkim strukturama u sistemu i obavljanje komunikacije sa njima.



Slika 3. SNMP protokol stek

Kao što se može videti sa Slike 3. SNMP se razlikuje u odnosu na UPnP i TR069 standard, opisan u Poglavlju 2.2.3, po upotrebi samo UDP protokola u transportnom sloju, gde se prenos vrši bez prethodnog uspostavljanja veze između entiteta u mreži. Posledica ovog pristupa može biti neispravnost prenetih podataka u slučaju sistema sa velikim brojem uređaja, gde se mrežni saobraćaj mnogostruko uvećava zbog čestog dobavljanja vrednosti parametara sa krajnjih uređaja.

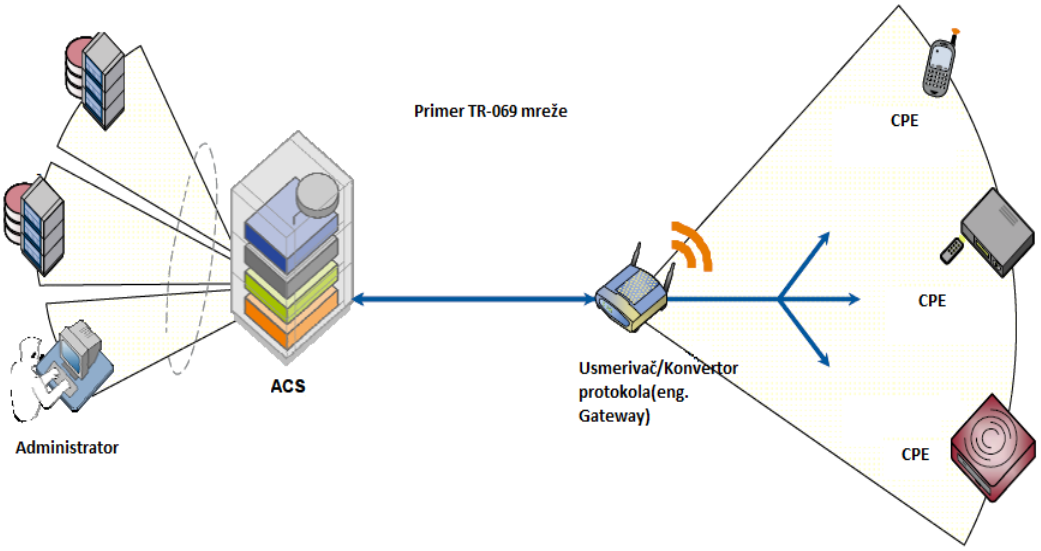
Iako je i dalje jedan od korišćenijih protokola za kontrolu i nadzor mrežnih sistema, SNMP protokol ima nedostatke koji prouzrokuju nestabilnost i ugrožavaju sigurnost današnjih informacionih sistema. Neki od nedostataka su:

- Manjak sigurnosnih mehanizama u ranijim verzjiama protokola,
- Upotreba UDP protokola u sistemima sa velikim brojem uređaja,
- Primenljivost na različite tipove uređaja pošto je SNMP prvenstveno osmišljen za mrežne uređaje kao što su: usmerivači (*Router*) , konvertori protokola (*Gateway*) , premostnici (*Bridge*) i drugi.

2.2.3 TR-069 komunikacioni protokol

TR-069 [16] (*Tehnical Report 069*) definiše protokol na aplikativnom nivou protokol steka za udaljeno rukovanje krajnjim uređajima. Korišćenjem ovog protokola veza se uvek uspostavlja između krajnjeg uređaja (CPE. *Customer Premises Equipment*) i ACS-a. Protokol se zasniva se na dvosmernoj SOAP/HTTP vezi. Protokolom je opisan samo način prenosa podataka između navedenih struktura u mreži, dok su definicije podataka koji se prenose date u modelima podataka pridruženih TR-069 standardu. Činjenice da se zasniva na HTTP protokolu i da je nezavisan od tipa uređaja donele su TR-069 protokolu široko polje primene. Neke od osnovnih funkcionalnosti koje pruža TR-069 standard su:

- Automatska konfiguracija i dinamičko omogućavanje usluga;
- Rukovanje programskom podrškom krajnjeg uređaja;
- Nagledanje statusa i performansi krajnjeg uređaja i njegova dijagnostika.



Slika 4. Primer okruženja TR-069 protokola[16]

Kao što se može videti na Slici 4. CPE uređaji mogu biti različitog tipa, a pritom ne postoje razlike u protokolu po kom se prenose podaci do ACS-a, razlikuju se samo modeli podataka za svaki od datih CPE-ova. TR-069 standard definiše niz protokola koji se moraju koristiti i na CPE i na ACS strani prenosa podataka.

CPE/ACS Korisnička aplikacija
RPC metode
SOAP
HTTP
SSL/TLS
TCP/IP

Tabela 2. Protokol stek TR-069 protokol standarda

Kao što se može videti u Tabeli 2. osnovni cilj TR-069 protokola je bezbedan prenos podataka, dok brzina prenosa i kašnjenje nije toliko bitno. Velika prednost je i što je obavezno korišćenje SSL i TLS enkripcionih algoritama, koji se trenutno najviše koriste u računarskim mrežama radi očuvanja sigurnosti prenosa informacija.

Poruke koje se prenose putem TR-069 protokola su standardizovane, gde je za svaku od njih definisana sintaksa poruke, način rukovanja, kao i rukovanje u slučaju greške. Prilikom standardizacije je korišćen SOAP protokol, koji koristi XML format.

Upravljačke poruke korišćene u TR-069 protokolu su definisane kao RPC metode, čime je omogućena jednostavna proširivost standarda. Ukoliko je potrebno dodavanje nove upravljačke poruke, potrebno je samo definisati novu RPC metodu, način rukovanja i njenu sintaksu na SOAP nivou, bez potrebe za menjanjem nižih nivoa protokol steka.

RPC metoda	TR-069 *amendment* 1	TR-069 *amendment* 2	TR-069 *amendment* 3	TR-069 *amendment* 4
GetRPCMethods	Obavezna	Obavezna	Obavezna	Obavezna
SetParameterValues	Obavezna	Obavezna	Obavezna	Obavezna
GetParameterValues	Obavezna	Obavezna	Obavezna	Obavezna
GetParameterNames	Obavezna	Obavezna	Obavezna	Obavezna
SetParameterAttributes	Obavezna	Obavezna	Obavezna	Obavezna
GetParameterAttributes	Obavezna	Obavezna	Obavezna	Obavezna
AddObject	Obavezna	Obavezna	Obavezna	Obavezna
DeleteObject	Obavezna	Obavezna	Obavezna	Obavezna
Download	Obavezna	Obavezna	Obavezna	Obavezna
Reboot	Obavezna	Obavezna	Obavezna	Obavezna
Inform	Obavezna	Obavezna	Obavezna	Obavezna
TransferComplete	Obavezna	Obavezna	Obavezna	Obavezna
Upload	Opciona	Opciona	Opciona	Opciona
FactoryReset	Opciona	Opciona	Opciona	Opciona
GetQueuedTransfers	Opciona	Opciona	Zastarela	Zastarela
ScheduleInform	Opciona	Opciona	Opciona	Opciona
SetVouchers	Opciona	Opciona	Zastarela	Zastarela
GetOptions	Opciona	Opciona	Zastarela	Zastarela
GetRPCMethods (ACS)	Opciona	Opciona	Opciona	Opciona
RequestDownload	Opciona	Opciona	Opciona	Opciona
Kicked	Opciona	Opciona	Zastarela	Zastarela
GetAllQueuedTransfers		Opciona	Opciona	Opciona
AutonomousTransferComplete		Opciona	Opciona	Opciona
DUStateChangeComplete			Opciona	Opciona
AutonomousDUStateChangeComplete			Opciona	Opciona
CancelTransfer			Opciona	Opciona
ScheduleDownload			Opciona	Opciona
ChangeDUState			Opciona	Opciona

Tabela 3. Zahtevi za podrškom RPC metoda po TR-069 verzijama standarda

Kao što se može videti u Tabeli 3.sa svakom verzijom standarda dolazi do dodavanja novih i izbacivanja zastarelih RPC metoda. Međutim, jedan od osnovnih zahteva TR-069 protokola je kompatibilnost sa prethodnim verzijama standarda, stoga je potrebno implementirati sve prethodne verzije standarda, bez obzira koja se verzija koristi.

Prednost TR-069 protokola je mogućnost praćenja primenjene konfiguracije na uređaju. Prilikom svake konfiguracije moguće je upisati vrednost identifikatora primenjenog konfiguracionog paketa koja se kasnije može pročitati radi provere konfiguracije na uređaju. Identifikator primenjenog konfiguracionog paketa zapisan je u polju Device.ManagementServer.ParameterKey[17].

2.2.3.1 Modeli podataka podržani TR-069 standardom

Kao što je već rečeno, široko polje primene TR-069 standarda je omogućeno zahvaljujući njegovoj primenljivosti na različite tipove uređaja bez potrebe za menjanjem načina prenosa poruka definisanih protokolom. Neki od modela podataka se navedeni u Tabeli 4:

Model podataka	*Tip uređaja*
TR-106	Osnovni model podataka za TR-069
TR-098	Internet konvertori prokola i usmerivači
TR-104	VoIP uređaji
TR-135	IPTV i STB uređaji
TR-140	NAS uređaji
TR-143	Model korišćen za dijagnostiku CPE
TR-157	Model komponenata datog CPE
TR-192	FAP uređaji

Tabeli 4. Modeli podataka podržani TR-069 protokolom

Modeli podataka, definisani TR-106 modelom podataka[17], su strukutirani kao stablo, u kom postoje sledeći tipovi čvorova:

- Korenski čvor;
- Objekat;
- Višestruki objekat;
- Parametar.

Korenski čvor omogućava objedinjavanje više modela podataka u jedno stablo ukoliko je to potrebno za određeni uređaj. Objekat predstavlja čvor u stablu koji ne može biti list, već se samo koristi za bolje struktuiranje podataka. Višestruki objekat je takođe objekat, s tim što se može više puta instancirati i pritom mu se pridružuje indeks pomoću kog mu se može jednoznačno pristupiti. Parametri predstavljaju listove u stablu i samo njima je pridružena vrednost. Pored vrednosti parametrima se pridružuje i niz atributa pomoću kojih se vrši konfiguracija uređaja od strane ACS-a. Atributi su:

-Nivo notifikacije (*Notification*) , koji definiše pod kojim okolnostima CPE treba da javi promenu vrednosti parametra ACS-u.

-Lista pristupa (*AccessList*), definiše listu entiteta koji smeju da promene vrednost parametra.

2.2.3.2 Komunikaciona sesija TR-069 protokola

Komunikaciona sesija definisana TR-069 standardom se uvek uspostavlja između jednog CPE-a i ACS-a. U jednom trenutku CPE sme biti u samo jednoj sesiji sa samo jednom ACS-om. CPE je HTTP klijent i uvek šalje HTTP zahteve, dok ACS uvek šalje HTTP odgovore. Pokretanje sesije mora biti pobuđeno određenim događajem koji se desio na CPE strani. CPE je obavezan da informiše ACS o događajima koji su se desili i zbog kojih je pokrenuta sesija u prvoj poruci u sesiji, *Inform* zahtevu. Događaji definisani TR-069 protokolu nalaze se u Tabeli 5:

Vrsta događaja	Značenje
0 Bootstrap	Prilikom prvog uključivanja CPE-a i prijavljanja kod ACS-a.
1 Boot	Prilikom pokretanja CPE-a.
2 Periodic	Prilikom isteka unapred definisanog intervala
3 Scheduled	Ako je ACS zakazao pokretanje sesije za određeno vreme
4 Value Change	Ukoliko se desila promena vrednosti parametra koji ACS želi da nagleda
5 Kicked	Ukoliko je došlo do promene internet stranice na strani CPE
6 Connection request	Ukoliko je ACS zatražio pokretanje sesije
7 Transfer Complete	Ukoliko se završio transfer fajla od strane CPE-a
8 Diagnostics Complete	Posle završetka dijagnstike na CPE-u
9 Request Download	Ukoliko CPE proverava da li postoji fajl koji treba da preuzme
M Reboot	Ukoliko se desilo ponovno pokretanje uređaja po zahtevu ACS-a
M ScheduleInform	Ako je ACS zakazao pokretanje sesije za određeno vreme više puta
M Download	Ukoliko je završeno preuzimanje fajla
M Upload	Ukoliko je završeno postavljanje fajla na određeni poslužilac
M <vendor-specific method>	Ukoliko je izvršena određena metoda speccifična za dati uređaj

Tabela 5. Tipovi događaja u TR-069 standardu

2.2.4 Poređenje protokola u sistemima za automatsku konfiguraciju

Funkcionalnosti koje su poželjne da protokol podržava u sistemima za automatsko upravljanje su:

- Sigurnost podataka koji se prenose;
- Jednostavna primenljivost na različite tipove uređaja;
- Jednostavnost protokola;
- Smanjeno opterećenje centralnog poslužioca;
- Kompatibilnost sa drugim protokolima korišćenim u sistemima za automatsko upravljanje.

Neke od gore navedenih osobina poseduju sva tri gore opisana protokola u Poglavljima 2.2.1, 2.2.2 i 2.2.3. UPnP je široko primenjen u krajnjim korisničkim uređajima, uglavnom isključuje upotrebu centralnog poslužioca budući da se prenos informacija obavlja između krajnjih uređaja u mreži. Budući da se već nalazi u programskoj podršci krajnjih uređaja korišćenje tog protokola bi isključilo potrebu za dodatnih resursima fizičke arhitekture, pre svega u pogledu trajne memorije. Najveći nedostaci UPnP protokola su nedostatak sigurnosnih mehanizama i kompatibilnost sa drugim protokolima u sistemima za automatsko upravljanje.

SNMP je protokol koji je jednostavan, sa malim skupom operacija i koji u novijim verzijama poseduje sigurnosne mehanizme. Takođe je kompatibilan sa drugim protokolima u sistemima za automatsko upravljanje budući da se TR-069 naziva naslednikom SNMP protokola i da se na uređajima neretko nalaze klijenti za oba protokola kao deo programske podrške. Međutim, SNMP je prevashodno osmišljen kao protokol za upravljanje mrežnim resursima. Jedan od nedostataka je korišćenje UDP protokola u transportnom sloju ISO/OSI modela čime se ugrožava validnost prenetih podataka između SNMP klijenta i SNMP poslužioca.

TR-069 poseduje većinu zahtevanih funkcionalnosti u sistemima za automatsko upravljanje. Sigurnosni mehanizmi su implementirani korišćenjem SSL i TLS enkripcionih mehanizama, primenljivost na različite tipove uređaja je realizovana korišćenje modela podataka, a smanjenje opterećenja centralnog poslužioca činjenicom da uvek TR-069 klijent inicira početak TR-069 komunikacione sesije i mogućnošću konfiguracije automatskog javljanja promene vrednosti parametra od strane TR-069 klijenta. Nedostatak predstavlja složenost protokola, naročito u novijim verzijama, prouzrokovana velikim brojem operacija. Takođe, pojedini modeli podataka su memorijski zahtevni zbog velikog broja parametara, tako da TR-069 protokol je najmanje optimalan po pitanju potrošnje memorije na krajnjem uređaju.

Zbog navedenih prednosti i mana navedenih za UPnP,SNMP i TR-069 protokol, u sklopu razvoja ovog sistema biće korišćen TR-069 protokol.

2.3 Upotreba skript jezika za definisanje konfiguracije

Algoritmi i procedure za automatsko konfigurisanje uređaja čine oslonac autonomnih sistema i oni se mogu odnositi na inicijalnu konfiguraciju, konfiguraciju tokom rada sistema usled pojave kvara u sistemu i poboljšanje performansi. Procedure se mogu definisati tokom razvoja sistema ili dinamički tokom rada sistema. Osnovne komponente konfiguracionih procedura čine:

- Kontrolna logika;
- Objekti upravljanja koji se odnose na krajnje uređaje;
- Parametri krajnjih uređaja na koje se primenjuje konfiguracija.

Istraživanja vezana za automatsku konfiguraciju uređaja definisanje konfiguracionih procedura zasnivaju na upotrebi skript jezika[9]. Konfiguracije definisane u skriptovima se mogu interpretirati na strani centralizovanog poslužioca i na strani krajnjeg uređaja, ukoliko korišćeni skript jezik može biti interepretiran na toj platformi. Interpretiranjem skriptova na centralizovanom poslužiocu se pozivaju upravljačke funkcije koje iniciraju razmenu kontrolnih poruka sa kranjim uređajem i konfigurišu ga.

Zahtevi sistemima za automatsku konfiguraciju koji se odnose na funkcionalnosti skript jezika su:

- Proširivost potrebna za pozivanje upravljačkih mehanizama;
- Jednostavnost;
- Podrška objektno orijentisanom programiranju;
- Podrška za strukuture podataka.

Proširivost skript jezika predstavlja najvažniju osobinu, zbog mogućnosti dodavanja novih funkcija u rečnik skript jezika. Korišćenjem ove mogućnosti piscu konfiguracionih skriptova je olakšan posao, jer pozivi funkcija za konfigurisanje krajnjih uređaja jednoznačno definišu operaciju koja će se primeniti na krajnjem uređaju. Podrška objektno orijentisanom programiranju zajedno sa podrškom za složene strukture podataka olakšavaju pisanje konfiguracionog skripta i omogućavaju jednostavnije modeliranje konfiguracionog scenarija. Podrška za strukture podataka je takođe važna zato što se podaci o parametrima na krajnjem uređaju ne predstavljaju kao jedna vrednost, već kao uređeni skup vrednosti unapred definisan standardom.

U današnjim konfiguracionim sistemima zasnovanim uglavnom na SNMP i TR069 standardima za definiciju konfiguracije upotrebom skriptova najviše se koriste skript jezici: JavaScript i Perl.

2.3.1 JavaScript

JavaScript je široko primenjen u računarskim sistemima. Sintaksa je slična programskom jeziku C. Podržava objektno orijentisan pristup i poseduje najširi spektar funkcionalsti u odnosu na sve druge skript jezike. Rečnik jezika je jednostavno proširiv za potrebe upravljačkih procedura u konfiguracionim sistemima. Jedna od prednosti koju pruža JavaScript je mogućnost uvoženja i pozivanje izvršavanja skripta iz drugih skriptova, što poboljšava preglednost i kontrolabilnost konfiguracionog skripta. Jedan od nedostataka JavaScript-a je kompleksnost skript jezika i veliki broj funkcionalnosti koje nisu potrebne u većini slučajeva primene konfiguracionih algoritama, a utiču na potrošnju memorije. Takođe, izvršavanje JavaScript koda u više niti nije podržano, što predstavlja jednu od većih mana ovog skript jezika.

2.3.2 Perl

Perl (*Practical Extraction and Report Language*) je široko primenjen u sistemima zasnovanim na Unix operativnim sistemima. Optimizovan je za manipulacije sa stringovima, poseduje direktnu spregu prema TCP/IP funkcionalnostima i time je omogućeno jednostavno pravljenje klijent-server aplikacija. U sistemima za automatsku konfiguraciju najčešći slučaj korišćenja Perl jezika je za izvršavanje skriptova na krajnjem uređaju, a ne na poslužiocu kao u slučaju drugih skript jezika. Jedan od razloga za široku primenu ovog jezika je podrška za objektno orijentisan pristup, međutim, preglednost koda i jednostavnost pisanja skripta je najveći nedostatak ovog skript jezika.

2.3.3 Lua

Lua skript jezik poseduje manji broj funkcionalnosti u odnosu na JavaScript i Perl, odlikuje je jednostavnost, velika brzina izvršavanja programskog koda i mali memorijski utrošak potreban za interpreter. Nedostatak je podrška za objektno orijentisan pristup i raznolikost struktura podataka kao što su redovi i liste. Jedina stuktura podataka je tabela i korišćenjem nje se nadomešta nedostatak ostalih struktura, kao i podrške objektno orijentisanom pristupu. Najveća prednost je jednostavna proširivost funkcionalnosti dodavanjem funkcija u meta-tabele koje sadrže skup ugrađenih funkcija koje Lua interpreter poseduje. Kao i JavaScript poseduje mogućnost za uvoženje i interpretiranje drugih skriptova iz skripta.

2.3.4 Poređenje skript jezika

Iako su JavaScript i Perl primenjeni u sistemima za automatsku konfiguraciju, autor smatra da ni jedan ni drugi jezik ne poseduju sve osobine nabrojane u Poglavlju 2.3. Glavni nedostaci

JavaScript-a su nemogućnost izvršavanja u više programskih niti(*multithreading)* i kompleksnost jezika zbog velikog broja funkcionalnosti. Najveća prednost je podrška za objektno orijentisan pristup što može biti korisno za potrebe modelovanja sistema koji se konfiguriše. Perl poseduje smanjen broj funkcionalnosti u odnosu na JavaScript. Široko je primenjen u sistemima za automatsku konfiguraciju ali na strani krajnjeg uređaja, a ne poslužioca. Lua ne poseduje podršku za objektno orijentisan pristup, što je veliki nedostatak. Najveće prednosti su jednostavnost i brzina izvršavanja što je po mišljenju autora najvažnije u sistemima za automatsko upravljanje u slučaju kada za veliki broj uređaja treba brzo naći rešenje za poboljšanje funkcionalnosti celog sistema.

2.4 Postojeća rešenja u oblasti

Oblast autonomnih sistema u računarstvu postoji nekoliko godina, njen razvoj je počeo početkom 21. veka. U svetu postoji nekolicina rešenja koja su kao osnovu za razvoj autonomnog sistema izabrala TR-069 prokotol i određeni skript jezik.

Motorola je razvila takav poslužilac za automatsku konfiguraciju nazvan *Motorola Edge ACS[18],* koji se zasniva na TR-069 protokolu i JavaScript jeziku. Definisanje skriptova u ovom poslužiocu je moguće za određeni uređaj ili određene tipove uređaja, dok definisanje za grupe uređaja nije. Ručno pokretanje konfiguracionog skripta je podržano, međutim zaustavljanje nije.

U svom radu [9] Rachidi je razvila sistem koji takođe koristi JavaScript za definisanje konfiguracije uređaja i zasniva se na TR-069 poslužiocu *OpenACS [19]* . Korišćenjem ovog sistema korisnik je u mogućnosti da definiše konfiguracioni skript i pridruži ga uz određene uređaje. Mehanizma za izvršavanje skriptova je realizovan unutar automatsko-konfiguracionog poslužioca, čime su obezbeđene dobre perfomanse rada sistema.

Autoru nije poznato nijedno rešenje koje uz pomoć Lua skript jezika i TR-069 protokola obezbeđuje funkcionalnosti za razvoj autonomnog sistema.

3. Koncept rešenja

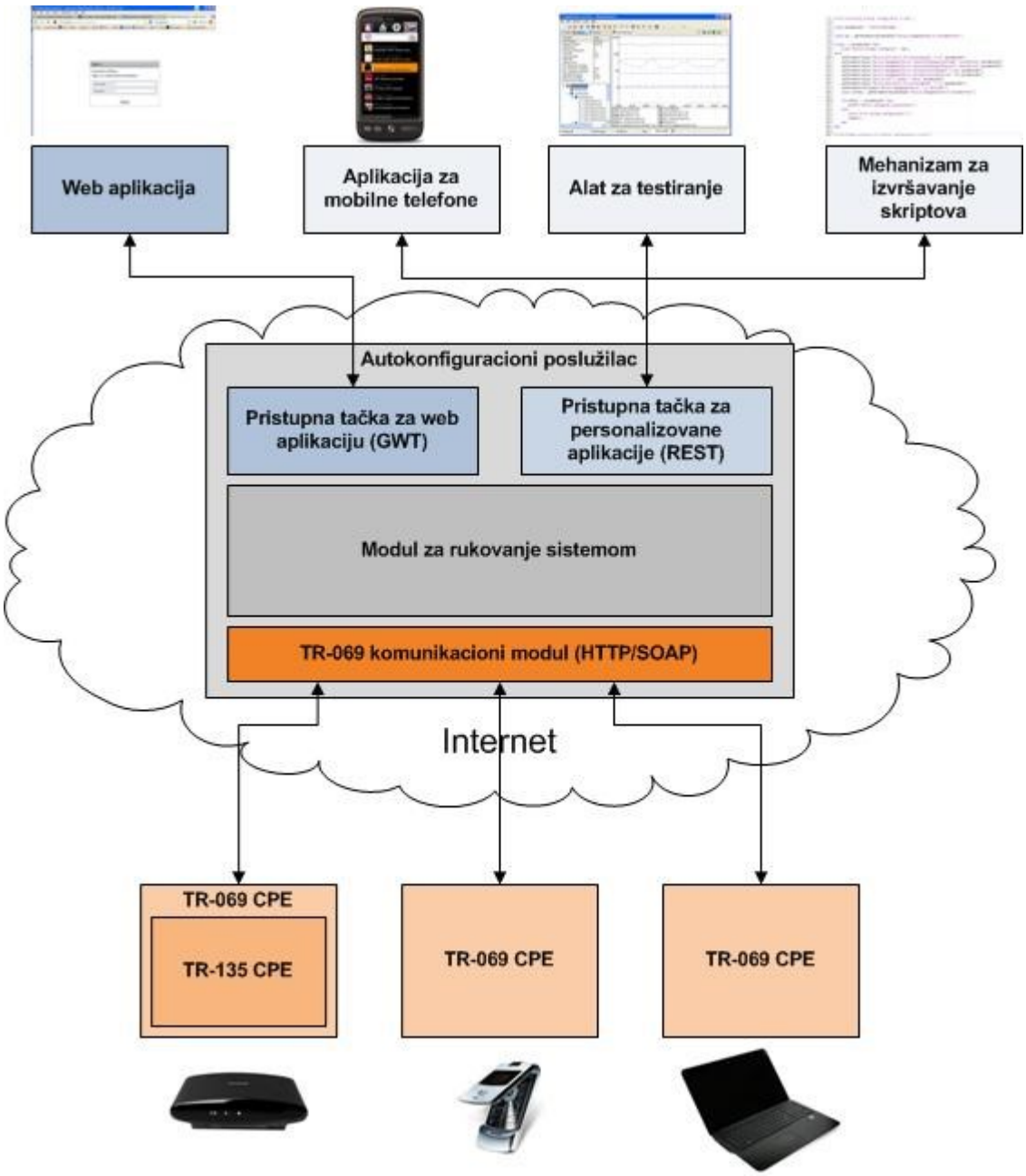
Implementacija rešenja ima za cilj nadogradnju na postojeći sistem koja omogućuje definisanje konfiguracionih skriptova tokom rada sistema, oblast primene skripta i događaj na uređaju za koji će se pokrenuti izvršavanje skripta.

Postojeći sistem nad kojim se vrši nadogradnja, *Insight*, poseduje sledeće funkcionalnost:

- Komunikaciju zasnovanu sa TR-069 protokolu,
- Biznis logiku za obradu podataka za pojedinačne uređaje ili grupe uređaja,
- Biznis logiku za obradu podataka o korisnicima uređaja,
- Sprege prema klijentskim aplikacijama zasnovane na WSDL i REST web servisima.

Korišćenjem nekih od ovih funkcionalnosti potrebno je implementirati mehanizam za konfiguracione skripte uz minimalno narušavanje perfomansi i robustnosti realizovanog sistema.

Prikaz Insight sistema za nadzor i konfiguraciju sa podrškom za izvršavanje skriptova prikazan je na Slici 5.



Slika 5. Prikaz Insight sistema sa nadogradnjom za konfiguracione skriptove

3.1 Analiza problema realizacije mehanizma za izvršavanje skriptova

Oslanjajući se na već postojeće rešenje za poslužilac za automatsku konfiguraciju zasnovano na *Enterprise JavaBeans* tehnologiji mehanizam za izvršavanje skriptova je moguće realizovati na dva načina:

- Kao modul unutar samog poslužioca;
- Kao zasebnu aplikaciju koja se spreže sa poslužiocem korišćenjem standardizovanih protokola i mehanizama.

Zahtevane funkcionalnosti mehanizma za izvršavanje skriptova su:

- Izlaganje funkcionalnosti za slanje TR-069 kontrolnih poruka u Lua skript jezik;
- Primena konfiguracionog skripta na pojedinačan uređaj ili grupu uređaja;
- Povezivanje skripta za određen događaj koji uređaj objavi poslužiocu;
- Automatska detekcija događaja i pokretanja skriptova;
- Ručno pokretanje i zaustavljanje skriptova.

Osobine TR-069 protokola, pre svega započinjanje komunikacione sesije od strane TR-069 klijenta kao i mogućnost slanja više konfiguracionih poruka tokom iste komunikacione sesije ostavljaju mogućnost za brže izvođenje konfiguracije. Iskorišćenje tih mogućnosti je moguće ukoliko funkcije izložene u Lua skript jezik imaju blokirajući i neblokirajući karakter.

Prethodno realizovani poslužilac za automatsku konfiguraciju u sistemu *Insight* se oslanja na mehanizam za slanje TR-069 konfiguracionih poruka koji ima neblokirajući karakter, pritom se odgovori dobijeni sa uređaja asinhrono prosleđuju inicijatoru slanja konfiguracionog zahteva. Stoga implementacija neblokirajućih konfiguracionih zahteva ne predstavlja problem,dok bi zahtevi sa blokirajućim karakterom zahtevali korišćenje sinhronizacionih strkutura kao što su međusobna isključivost (eng. *mutex)* , semafor (*semaphore*) i uslovna promenljiva (*condition variable*). Korišćenje sinhronizacionih struktura u EJB tehnologiji predstavlja rizik za robustnost sistema zbog zaključavanja između *SessionBean-ova* tokom transakcija[20]. Paralelno izvršavanje većeg broja skriptova takođe može ugroziti robustnost sistema usled ograničenja za kreiranje programskih niti u aplikacijama zasnovanim na EJB tehnologiji[21].

Gore navedeni razlozi predstavljaju rizik po stabilnost sistema ukoliko bi se mehanizam za izvršavanje skriptova realizovao kao modul u poslužiocu. Sa druge strane, mehanizam se može realizovati kao zasebna Java aplikacija i u tom slučaju stabilnost poslužioca nije ugrožena. Mana ovakvog pristupa predstavlja utrošak vremena na razmenu poruka izmedju aplikacije i poslužioca.

3.2 Proširenja implementiranog poslužioca

Postojeće rešenje ACS-a potrebno je proširiti radi podrške za mehanizam za konfiguraciju korišćenjem skriptova. Funkcionalnosti koje treba podržati su navedene u Poglavlju 3.1. Ispunjavanje tih funkcionalnosti izazvalo je promene u ACS-u koje se tiču:

- Biznis logike za pristup konfiguracionim skriptovima;
- Sprezanje sa modulom za TR-069 komunikaciju;
- Mehanizma za objavu događaja;
- Implementacije sprega sa aplikacijom za izvršavanje skriptova.

Navedena proširenja pojedinačno će biti opisana u narednim poglavljima.

3.2.1 Podrška za detekciju događaja sa uređaja

TR-069 protokolom definisana su pravila koja opisuju način objave događaja od strane krajnjeg uređaja. Tipovi događaja koje uređaj objavlja poslužiocu navedeni su u Poglavlju 2.2.3.2. i iskorišćeni su za mehanizam automatskog pokretanja konfiguracionog skripta. Budući da pokretanje skripta vremenski zahtevno, objava događaja se izvršava asinhrono. Implementacija asinhronog mehanizma je moguća korišćenjem *Message-Driven Bean-ova* ili korišćenjem ashinhronih funkcija. Budući da obrada događaja nije proces koji traje više od nekoliko sekundi korišćenje *Message-Driven Bean-ova* predstavlja nepotreban utrošak vremena na transakcije između *bean-ova* , korišćen je mehanizam ashinronog izvršavanja funkcija.

3.2.2 Proširenja modula za TR-069 komunikaciju

Proširenja modula se odnose na implementaciju funkcionalnosti za raspoznavanje od koga je inicirano slanje konfiguracione poruke krajnjem uređaju i funkcionalnosti za prosleđivanje rezultata izvršenja konfiguracione poruke na uređaju. Inicijator slanja konfiguracione poruke može biti:

- Mehanizam za izvršavanje konfiguracionih skriptova;
- Drugi moduli unutar poslužioca;
- Operater korišćenjem web aplikacije.

Implementirano proširenje podrazumeva razdvajanje konfiguracionih poruka tako da u slučaju da u redu čekanja postoji zahtev za krajnji uređaj koji je istog tipa kao zahtev iniciran iz skripta, njihov sadržaji neće biti spojeni u jedan zahtev, nego će biti poslati uređaju kao odvojeni. Rezultat toga je da će mehanizam za izvršavanje skriptova dobijati odgovore samo za svoje zahteve, a ne za zahteve koji su inicirani iz drugih delova sistema.

Funkcionalnost za prosleđivanje odgovora na zahteve se zasniva na čitanju podataka iz SOAP poruke dobijene od CPE-a i slanju tih podataka spoljnoj aplikaciji koja izvršava skript. Pritom se podaci pakuju u JSON poruku. Mehanizam pakovanja JSON poruka i format istih je opisan u Poglavlju 3.4.

3.2.3 Proširenja baze podataka

Baza podataka poslužioca je proširena entitetom nazvanim *Action* zaduženim za čuvanje sadržaja skripta, podacima potrebnim za definisanje oblasti primene skripte i na koji događaj će se pokrenuti izvršavanje i dodatnim podacima za bolji uvid u rad skripte:

- ime skripta;
- da li se skript trenutno izvršava;
- da li se trenutno menja funkcionalnost skripta;

- kada je poslednji put pokrenuto izvršavanje skripta.

Pregled polja entiteta *Action* nalazi se u Tabeli 6.

Ime polja	Tip podatka	Značenje
scriptName	*String*	Ime skripta, jedinstveni identifikator entiteta
devices	*Set<CPE>*	Pojedinačni uređaji na koje se skript primenjuje
groups	*Set<Group>*	Grupe uređaja na koje se skript primenjuje
filter	*List<ScriptFilter>*	Lista kriterijuma za koje primenjuje skripta
event	*String*	Događaj na koji se pokreće izvršavanje skripta
isRunning	*Boolean*	Da li se skript trenutno izvršava
isEditing	*Boolean*	Da li se skript trenutno redefniše
lastTimeRun	*Date*	Poslednje trenutak pokretanja skripta
scriptIds	*List<ActionExecutionIndex>*	Identifikatori izvršavanja skripta
scriptContent	*String*	Programski kod skripta

Tabela 6. Opis polja entiteta *Action*

Kao što se može videti iz Tabele 6. za isti skript se može definisati oblast primene na više pojedinačnih uređaja, više grupa uređaja i po predefinisanom kriterijumu u polju *filter*. Kriterijum se oslanja na polja za identifikaciju uređaja po TR-069 standardu:

- Proizvođač uređaja (*Manufacturer*),
- Klasa uređaja (*ProductClass*).

Jedan skript može istovremeno da ima više pokrenutih instanci u kojima se izvršava. Za svaki zahtev za pokretanje skripta prema aplikaciji za izvršavanje skriptova ACS-u se vraća jedinstveni niz karaktera koji označava pokrenutu instancu skripta. Korišćenjem te instance ACS može tokom rada da zaustavi izvršavanje skripta. Detaljnije objašnjenje značenja identifikatora pokrenute instance skripta dato je u Poglavlju 3.3.

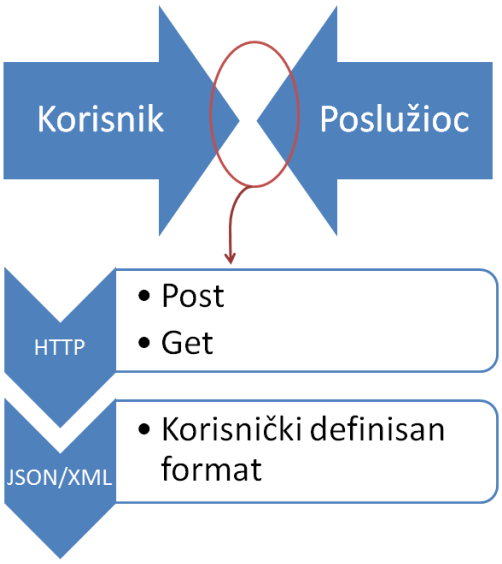
3.2.4 Proširenja REST web servisa

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 XML/SOAP format poruka ima široku primenu u komunikacionim sistemima. 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 – intenzivna 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 komunikaciji. 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 6.



Slika 6. 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 set 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.

Proširenja u REST web servisu se odnose na dodavanje sprežnih funkcija za pristup funcionalnostima konfiguracionog poslužioca. Dodate funkcije su objedinjene unutar jednog servisa i predstavljaju spregu između aplikacije za izvršavanje skripta i modula za TR-069 komunikaciju. Realizovane funkcionalnosti su:

- Dodavanje objekta u repozitorijumu;
- Uklanjanje objekta iz repozitorijuma;
- Postavljanje vrednosti parametra;
- Dobavljanje vrednosti parametra;

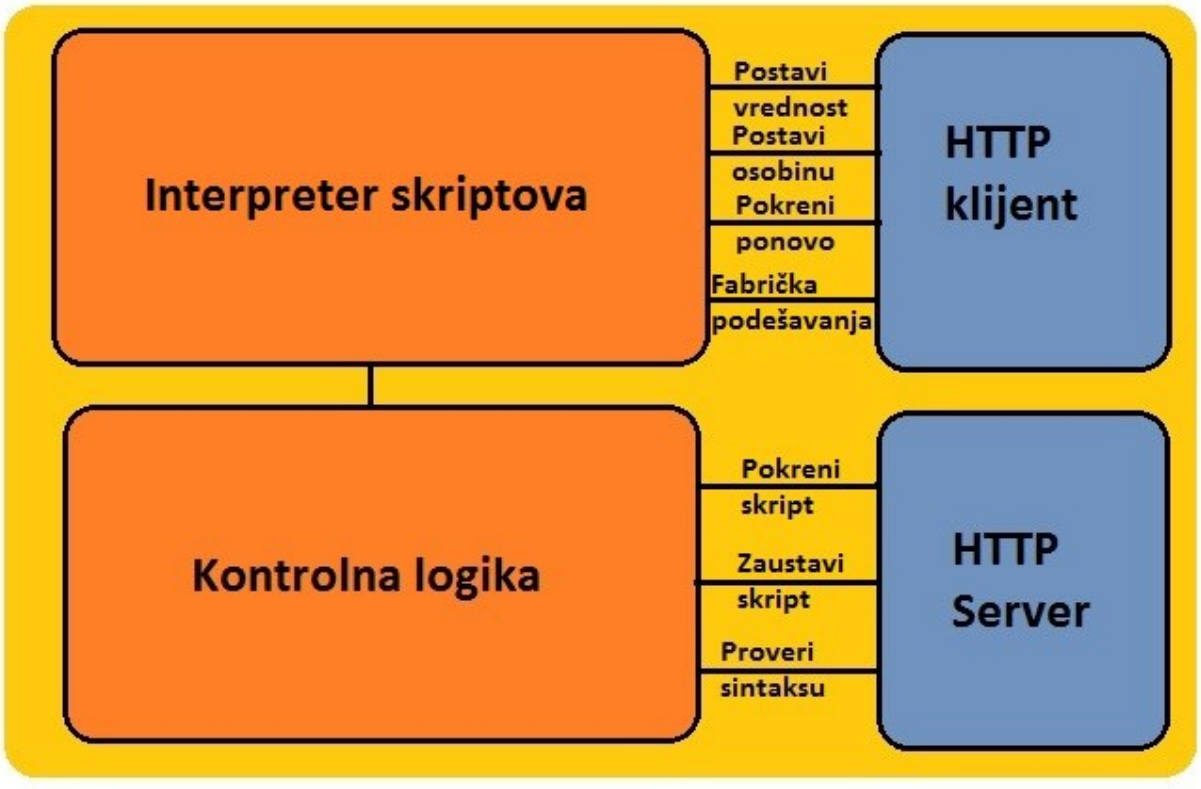
- Postavljanje osobina parametra;
- Dobavljanje osobina parametra;
- Ponovno pokretanje;
- Povratak na fabrička podešavanja;
- Izveštavanje o završetku izvršavanja konfiguracionog skripta.

3.3 Implementacija aplikacije za izvršavanje skriptova

Aplikacija za izvršavanje skriptova predstavlja središte mehanizma za podršku konfiguracionim skriptovima u sistemu *Insight*. Nadležnosti ove aplikacije su:

- Prijem i obrada zahteva za pokretanja i zaustavljanje skritpova;
- Izvršavanje skriptova;
- Slanje zahteva za konfiguraciju krajnjih uređaja;
- Prijem izveštaja o primenljenosti konfiguracije na krajnjem uređaju;
- Obaveštavanje konfiguracionog poslužioca o toku izvršavanja skripta.

Na Slici 7. nalazi se prikaz arhitekture aplikacije na najvišem nivou.



Slika 7. Prikaz arhitekture aplikacije na najvišem nivou

Kao što se može videti sa Slike 7. u aplikaciji su implementirani sledeći moduli:

- Moduli za dvosmernu HTTP komunikaciju;
  - HTTP klijent;
  - HTTP poslužilac.

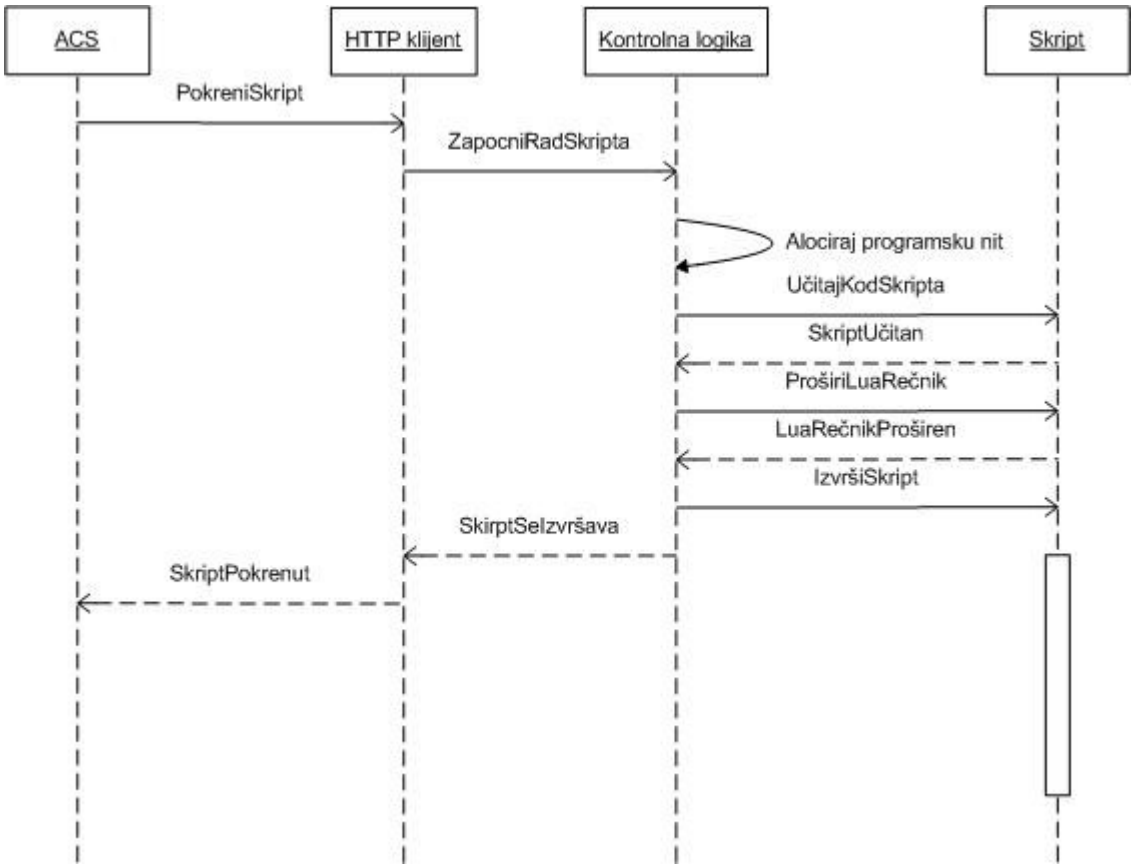
- Kontrolna logika za pokretanje i nadzor rada skripta;
- Interpreter skriptova zadužen za izvršenje skriptova i inicijaciju slanja zahteva za konfiguraciju krajnjeg uređaja.

3.3.1 Izvršavanje skriptova

Izvršavanje konfiguracionih skriptova se zasniva na funkcionalnostima *LuaJ* interpretera. Korišćene funkcionalnosti su:

- Učitavanje programskog koda skripta;
- Pokretanje i zaustavljanje rada skripta;
- Proširivanje vokabulara Lua jezika.

Budući da se svaki skript izvršava u zasebnoj niti, funkcionalnost za kreiranje,nadgledanje izvršavanja i zaustavlje rada je omogućena korišćenjen Java *Executor*-a, koji predstavlja bazen niti koje dodeljuje i zaustavlja po potrebi.



Slika 8. Primer komunikacije između modula prilikom pokretanja skripta

Kao što se može videti na Slici 8. prilikom prijema zahteva za pokretanja skripta od strane ACS-a kontrolna logika za izvršavanje skriptova proverava da li je moguće izvršenja tog skripta u zavisnosti od trenutne raspoloživosti resursa. Na raspoloživost resursa najviše utiče trenutni broj skriptova koji se izvršavaju u aplikaciji. Ukoliko postoji mogućnost za pokretanje skripta, kontrolna logika kreira zasebnu nit za novi skript i dodeljuje mu identifikator. Taj identifikator se

šalje u ACS-u u odgovoru na zahteva za pokretanje skripta i kasnije mu služi za izveštavanje o toku i završetku rada skripta. Kada je nit kreirana kontrolna logika učitava programski kod skripta u interpreter, proširuje Lua rečnik funkcijama potrebnim za konfiguraciju krajnjih uređaja i započinje izvršavanje skripta.

Proširenja Lua vokabulara vezana za konfiguraciju krajnjih uređaja su:

- Postavljanje i dobavljanje vrednosti parametra;
- Postavljanje i dobavljanje osobina parametra;
- Dodavanje i uklanjanje objekta;
- Povratak na fabrička podešavanja;
- Ponovno pokretanje uređaja.

Svaka od navedenih funkcionalnosti je realizovana kao blokirajući i neblokirajući poziv. Blokirajući pozivi znače da se izvršavanje skripta neće nastaviti sve dok se ne dobije izveštaj o primeni konfiguracionog zahteva na CPE-u, dok neblokirajući omogućavaju nastavak izvršavanja skripta odmah po prijemu odgovora sa ACS-a. Proširenje Lua rečnika je usko povezano za mehanizmom za slanje i obradu zahteva za konfiguraciju uređaja.

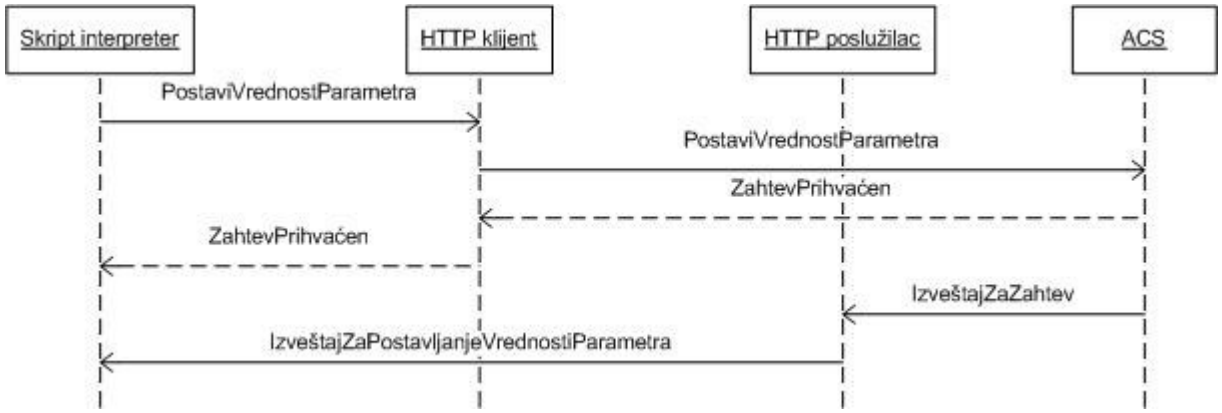
Tokom rada aplikacije kontrolna logika periodično proverava da li je završeno izvršavanje nekog skripta i ukoliko jeste šalje obaveštenje konfiguracionom poslužiocu korišćenjem HTTP klijenta i uklanja nit iz bazena niti aplikacije. Uslovi za zaključivanje da li je skript završio izvršenje su:

- Sve funkcije iz skripta su pozvane;
- Dobijeni su odgovori i izveštaji na sve poslate konfiguracione zahteve za dati CPE.

Detaljniji opis mehanizma za konfiguracione zahteve se nalazi u Poglavlju 3.4.

3.3.2 Mehanizam za slanje i obradu zahteva za konfiguraciju uređaja

Uloga mehanizma je slanje zahteva za konfiguraciju krajnjeg uređaja korišćenjem HTTP klijenta, sinhronizacija izvršavanja skripta u skladu sa odgovorima i izveštajima o primenjenosti konfiguracionih zahteva na krajnjim uređajima i prosleđivanje rezultata konfiguracionih zahteva Lua skript interpreteru. Primer izvršenja konfiguracionog zahteva prikazan je na Slici 9.



Slika 9.Primer komunikacije prilikom slanja konfiguracionog zahteva

Mehanizam za slanje i obradu zahteva za konfigurisanje se oslanja na strukture podataka za čuvanje podataka vezanih za konfiguracione zahteve. Prilikom slanja, zahtevi se čuvaju u FIFO (*First In Firsti Out*) strukturi. Smeštanje zahteva u FIFO strukturu vrši interpreter skriptova i oni se tu čuvaju dok god se ne pošalju ACS-u. Nakon slanja zahteva i prijema odgovora od ACS-a, u zavisnosti od odgovora HTTP klijent zahtev briše ili premešta u mapu (*HashMap*) u kojoj se čuvaju svi zahtevi na koje se čeka izveštaj o izvršenju na CPE-u. Ukoliko vrednost status polja ukazuje da je primena zahteva nemoguća (*RequestFailed*), on će biti obrisan,a ukoliko je zahtev prihvaćen od strane ACS-a on će biti premešten u mapu i čekaće se izveštaj. Ključ za identifikaciju zahteva je identifikator dobijen u odgovoru na zahtev od strane ACS-a. Detaljniji opis značenja statusa i identifikatora zahteva se nalazi u Poglavlju 3.4. Prilikom prijema izveštaja o primenjenosti zahteva na krajnjem uređaju rezultati će biti prosleđeni u interpreter skriptova i potom se mogu konvertovati u Lua vrednosti i koristiti za dalji tok izvršenja Lua skripta.

3.4 Sprega između aplikacije i ACS-a

Sprega između aplikacije za izvršavanje skriptova i ACS-a se zasniva na dvosmernoj HTTP komunikaciji. Poruke koje se razmenjuju su JSON formata i implementirani sigurnosti mehanizam se zasniva na HTTP autentifikaciji. Prilikom razmene poruka dešavaju se dva scenarija:

1. ACS je HTTP klijent,a aplikacija HTTP poslužilac;
2. Aplikacija je HTTP klijent, a ACS je HTTP poslužilac.

U prvom scenariju komunikacija je inicirana od strane ACS-a u sledeće svrhe:

- Zahtev za pokretanje rada skripta;
- Zahtev za zaustavljanje rada skripta;
- Slanje izveštaja o odgovoru na TR-069 kontrolni zahtev sa CPE-a.

Zahtev za pokretanje i zahtev za zaustavljanje rada skriptova se po semantici mogu odvojiti od slanja izveštaja o odgovoru na TR-069 zahtev. Zahtevi za pokretanje i zaustavljanje

se iniciraju ili od strane operatera ili od strane modula za upravljanje skriptovima, dok se izveštaji iniciraju od strane modula za TR-069 komunikaciju. Takođe, u zahtevima za pokretanje i zaustavljanje rada skripta je objekat upravljanja skript ili njegova instanca u aplikaciji za izvršavanje skriptova, dok se u izveštajima radi o TR-069 kontrolnim zahtevima iniciranih tokom rada konfiguracionog skripta.

Primeri JSON poruka vezanih za pokretanje izvršavanja skripta nalaze se na Slici 10. i Slici 11.

```
{"command":"PLAY_SCRIPT_COMMAND",
 "scriptCode":"print("FactoryReset");
 local f = factoryReset();
 print("Factory reset status "..f);
 print("AddObject");
 local a = addObject("Device.STBService.","parameterKey4234");
 sleep(10000);
 print("AddObject2");
 local b = addObject("Device.DHCPOption.","parameterKey4334");
 print(a,b);
 }}
```

Slika 10. Primer JSON poruke za pokretanje rada skripta

Kao što se može videti sa Slike 10. ACS šalje zahtev za izvršavanje određene komande u aplikaciji, tip komande je definisan u polju *command*, dok se u ostalim poljima nalaze podaci specifični za tip komande.

```
{"requestStatus":"REQUEST_OK",
 "scriptId":621977131}
```

Slika 11. Primer JSON poruke za odgovor aplikacije na zahtev za pokretanje rada skripta

Na Slici 11. je prikazan odgovor aplikacije na zahtev ACS-a za pokretanje rada skripta. Polje *requestStatus* je zajedničko za sve odgovore na zahteve ACS-a i u njemu se opisuje uspešnost izvršavanja zahteva, dok se ostala polja razlikuju po tipu zahteva. U ovom slučaju polje *skriptId* sadrži vrednost identifikatora instance pokrenutog skripta. Značenje identifikatora je detaljnije objašnjeno u Poglavlju 3.3.2.

Primeri JSON poruka vezanih za izveštavanje o odgovoru dobijenom od krajnjeg uređaja se nalaze na Slici 12. i Slici 13.

```
{"command":"REQUEST_REPORT",
 "requestId":"ddoidhpq1hptlmt6mktr4mn09v",
 "reports":
 [{"deviceId":{"manufacturer":"RT-RK","oui":"FC-FE-88",
   "productClass":"RK-2030","serialNumber":"00302148791"},
 "faultString":"Remote factory reset not supported on device.",
 "faultCode":9002}]
}
```


Slika 12. Primer izveštaja o izvršavanja kontrolne poruke na krajnjem uređaju

Na Slici 12. je prikazan izveštaj o izvršavanju kontrolne poruke za povratak uređaja na fabrička podešavanja (*FactoryReset*). Svaki izveštaj sadrži u sebi polje *requestId* koje jednoznačno identifikuje zahtev koji je prethodno poslat iz aplikacije za izvršavanje skriptova i koji nije mogao biti odmah izvršen na krajnjem uređaju zbog prirode TR-069 protokola i načina implementacije ACS-a. Polje *reports* predstavlja listu izveštaja prethodno poslatog zahteva, sa obaveznim poljem *deviceId* koje jednoznačno identifikuje uređaj na kome se primenio prethodno poslati zahtev.


```
{"requestStatus":"REPORT_PROCESSED"}
```


Slika 13. Primer odgovora na izveštaj o izvršavanju kontrolne poruke na krajnjem uređaju

U drugom scenariju poslužilac za automatsku konfiguraciju prima zahteve iz aplikacije korišćenjem REST web servisa. Funkcionalnosti čije se izvršavanje zahteva ovim scenarijom su opisane u Poglavlju 3.2.4. Sve prethodno navedene funkcionalnost osim izveštavanja o završetku izvršavanja konfiguracionog skripta se oslanjaju na TR-069 protokol i zbog osobine protokola koja se odnosi na komunikaciju iniciranu od strane TR-069 klijenta, konačni odgovor o izvršenju zahteva iniciranog iz aplikacije je nemoguće dati. Stoga se u odgovoru na zahtev aplikaciji šalje da li je zahtev moguće izvršiti ili pretpostavka kada će se zahtev izvršiti. Razlog za nemogućnost izvršenja je da se traženi uređaj ili grupa uređaja ne nalazi registrovan kod ACS-a. Pretpostavka o trenutku izvršenja zahteva na krajnjem uređaju se zasniva na uspešnosti izvršavanja zahteva za vezu sa krajnjim uređajem (*ConnectionRequest[16]*).

Na Slici 14. dat je primer zahteva od strane aplikacije.


```
{"objectPath":"Device.STBService.",
"parameterKey":"parameterKey4234",
"devId":{"manufacturer":"RT-RK","oui":"FC-FE-88",
"productClass":"RK-2030","serialNumber":"00302148791"}
}
```


Slika 14. Primer zahteva za izvršenje na krajnjem uređaju

Kao što se može videti na Slici 14. sadržaj JSON poruke sadrži neophodne podatke za kasnije kreiranje SOAP poruke prema CPE-u. Ono što se ne nalazi u JSON poruci je tip kontrolne poruke prema krajnjem uređaju. Tip poruke se definiše u kao poslednji deo u URL adresi na koju se šalje zahtev. Primer takve adrese je:

http://insight.rt-rk.com/restapi/ActionScriptServices/addObject

Na Slici 15. i Slici 16. prikazani su odgovori na zahteve poslate iz aplikacije.

{"identifier":"8e4q1mafvnpqbt0atj41b3tdtg","status":"requestWaitingConnReqFailed"}

Slika 15. Primer odgovora na zahtev koji je moguće izvršiti

{"identifier":"","status":"requestFailed"}

Slika 16.Primer odgovora na zahtev koji je nemoguće izvršiti

U primeru na Slici 15. primljeni zahtev je moguće izvršiti i zato se u odgovoru nalazi identifikator zahteva. U polju *status* se definiše brzina dospeća izveštaja na taj zahtev. Vrednosti u tom polju mogu biti:

- *requestWaitingConnReqFailed* – nije uspeo zahtev za povezivanje sa krajnjim uređajem;
- *requestWaitingConnReqSuccess* – uspeo je zahtev povezivanje sa krajnjim uređajem.

U zavisnosti od ovog polja aplikacija za izvršavanje skriptova odlučuje da li da čeka na izveštaj. Detaljan opis te funkcionalnosti se nalazi u Poglavlju 3.3.1.

4. Programsko rešenje

Programsko rešenje se zasniva na implementaciji aplikacije za izvršavanje konfiguracionih skriptova i proširenja postojećeg rešenja automatsko-konfiguracionog poslužioca zasnovanog na TR-069 protokolu.

4.1 Implementacija aplikacije

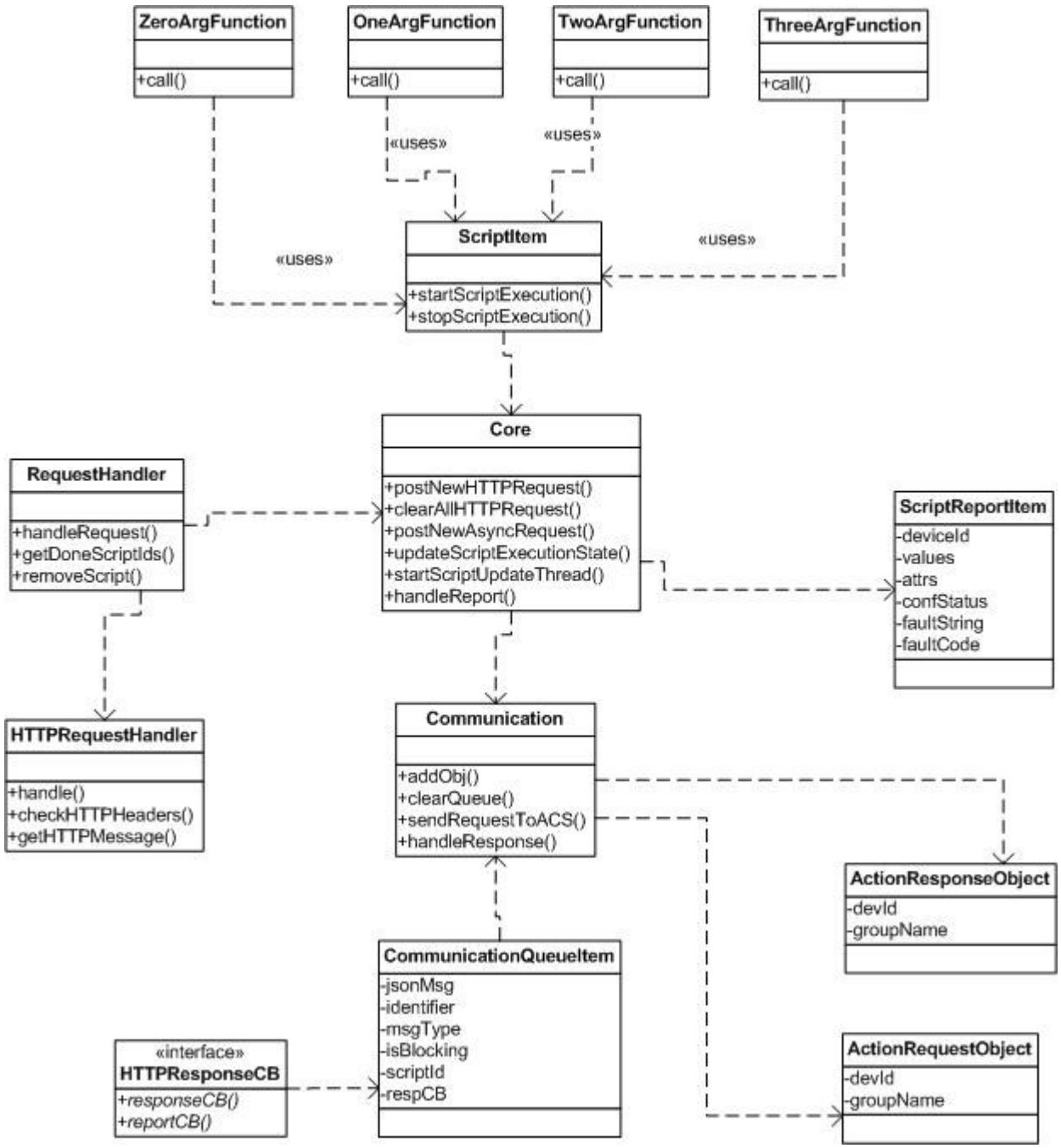
Rešenje je realizovano u programskom jeziku Java, čime je obezbeđena upotreba na različitim platformama zahvaljujući apstrakciji virtuelne mašine. Važno je napomenuti zavisnost aplikacije od biblioteka:

- *LuaJ* [22] – zadužena za interpretiranje Lua skriptova;
- *Gson* [23] – podrška za pakovanje i raspakivanje JSON poruka;
- *Sun HTTP server* [24] – podrška za HTTP poslužioca.

Aplikaciju čine moduli:

- *HTTP poslužilac*
- *Modul za rukovanje skriptovima*
- *Proširenje Lua rečnika*
- *HTTP klijent*
- *Podrška za Json poruke*

Na Slici 17. prikazane su veze između klasa unutar aplikacije za izvršavanje skriptova.



Slika 17. Prikaz dijagrama klasa u aplikaciji za izvršavanje skriptova

4.1.1 HTTP poslužilac

Funkcionalnosti HTTP poslužioca u aplikaciji za izvršavanje skriptova su implementirane u klasi *HttpRequestHandler*. Ova klasa implementira *HttpHandler* interfejs iz biblioteke *Sun HTTP server*. Funkcionalnosti HTTP poslužioca su:

- Prijem HTTP zahteva;
- Izvršavanje *Basic* autentifikacije HTTP klijenta;
- Proveru da li se radi o HTTP zahtevu sa JSON porukom u HTTP telu;
- Čitanje sadržaja HTTP tela i prosleđivanje modulu za rukovanje skriptovima;

- Odgovaranje na prethodno primljeni HTTP zahtev.

Sledi opis funkcija realizovanih unutar modula *HTTP poslužilac*.

- `public void handle (HttpExchange t)`
  - Opis: Obrada HTTP zahteva
  - Parametri:
    - *HttpExhange t* – enkapsulacija podataka potrebnih za obradu primljenog HTTP zahteva

- `private void checkHTTPHeaders (Headers headers) throws NullPointerException, IllegalAccessException,IllegalArgumentException`
  - Opis: Provera HTTP zaglavlja primljenog HTTP zahteva
  - Parametri:
    - *Headers headers* – Primljena HTTP zaglavlja
  - Izuzeci:
    - *NullPointerException* – Ukoliko je vrednost parametra *null*
    - *IllegalArgumentException* – Ukoliko HTTP zaglavlje ne ukazuje da se radi HTTP telu sa JSON porukom u njoj
    - *IllegalAccessException* – Ukoliko je došlo do greške prilikom autentifikacije

- `private String getHTTPMesage (InputStream is) throws NullPointerException`
  - Opis: Pridruživanje kriterijuma za pokretanje skripta na osnovu proizvođača i klase uređaja
  - Parametri:
    - *InputStream is* – Memorijska zona u kojoj se nalazi primljeno HTTP telo
  - Izuzeci:
    - *NullPointerException* – Ukoliko je vrednost parametra *null*

4.1.2 Modul za rukovanje skriptovima

Modul za rukovanje skriptovima predstavlja centralno mesto u aplikaciji za izvršavanje skriptova i njegove funkcionalnosti su:

- Pokretanje i nadgledanje rada skriptova;
- Proširenje Lua rečnika prilikom pokretanja skripta, detaljnije objašnjeno u Poglavlju 4.1.3;
- Iniciranje slanja kontrlnih zahteva prema ACS-u;
- Prijem izveštaja o uspešnosti primene kontrolnih zahteva i distribucija istih prema pokrenutim instancama skriptova.

Navedene funkcionalnosti su implementirane u klasama opisanim dalje u ovom poglavlju.

4.1.2.1 Klasa Core

Klasa *Core* predstavlja centralnu klasu u implementaciji aplikacije za izvršavanje skriptova. U njoj se kontrolišu aktivnosti vezane za HTTP poslužioca i klijenta, mehanizma za

pokretanje, zaustavljanje i nadzor rada skripta kao i prosleđivanje izveštaja instancama pokrenutih skriptova.

Sledi opis funkcija realizovanih unutar klase *Core*.

- `private boolean postNewHTTPrequest (CommunicationQueueItem el)`
  - Opis: Dodavanje novog HTTP zahteva na slanje HTTP klijentu
  - Parametri:
    - *CommunicationQueueItem el* – HTTP zahtev koji treba poslati
  - Povratna vrednost
    - *true* - zahtev uspešno dodat
    - *false* - zahtev nije dodat

- `public boolean clearAllHTTPRequest ()`
  - Opis: Brisanje svih HTTP zahteva u HTTP klijentu
  - Povratna vrednost
    - *true* - zahtevi uspešno obrisani
    - *false* - zahtevi nisu orbisani

- `public void postNewAsyncRequest (String identifier, CommunicationQueueItem el)`
  - Opis: Dodavanje novog zahteva na koji se čeka izveštaj od automatsko-konfiguracionog poslužiocas
  - Parametri:
    - *String identifier* – Identifikator zahtev dobijen od automatsko-konfiguracionog poslužioca
    - *CommunicationQueueItem el* – HTTP zahtev koji treba poslati

- `public void updateScriptsExecutionState ()`
  - Opis: Iniciranje slanja obaveštenja o završetku rada skripta konfiguracionom poslužiocu

- `public void startScriptUpdateThread ()`
  - Opis: Pokretanje niti koja periodično proverava koji skriptovi su završili izvršavanje

- `public void handleReport (ActionScriptControlRequest request) throws IllegalAccessException`
  - Opis: Početak obrade izveštaja o kontrolnom zahtevu i dalja distribucija instanci skripta koja čeka na izveštaj
  - Parametri:
    - *ActionScriptControlRequest request* – podaci iz izvreštaja i identifikator instance skripta kojoj treba poslediti izveštaj
  - Izuzeci:
    - *IllegalAccessException* – Ukoliko instanca skripte kojoj je namenjen izveštaj ne postoji

4.1.2.2 Klasa RequestHandler

Klasa *RequestHandler* se bavi obradom zahteva primljenih od HTTP poslužioca u aplikaciji. Zahtevi se mogu odnositi na zahteve za kontrolu skriptova, kao što su pokretanje i zaustavljanje, kao i izveštaje o izvršenje kontrolnih zahteva na krajnjem uređaju.

Sledi opis funkcija realizovanih unutar klase *RequestHandler*.

- `public ActionScriptControlResponse handleRequest (ActionScriptControlRequest req) throws NullPointerException,IllegalArgumentException`
  - Opis: Obrada zahteva dobijenog od HTTP poslužioca
  - Parametri:
    - *ActionScriptControlRequest req* – Podaci dobijeni iz tela HTTP zahteva
  - Povratna vrednost
    - *ActionScriptControlResponse* – odgovor na obrađeni zahtev
  - Izuzeci:
    - *NullPointerException* – Ukoliko je vrednost argumenta *req* jednaka *null*
    - *IllegalArgumentExeption* – Ukoliko dođe do greše prilikom obrade zahteva

- `public ArrayList<Integer> getDoneScriptsIds () throws NullPointerException`
  - Opis: Dobavljanje identifikatora svih skriptova koji su završili izvršavanje-
  - Povratna vrednost
    - *ArrayList<Integer>* - identifikator instanci skriptova koji su završili izvršavanje
  - Izuzeci:
    - *NullPointerException* – Ukoliko bazen niti u kojima se skriptovi izvršavaju nije inicijalizovan

- `public void removeScript (Integer id)`
  - Opis: Uklanjanje niti u kojoj se izvršava skript iz bazena niti
  - Parametri:
    - *Integer id* – identifikator instance skripta

4.1.2.3 Klasa ScriptItem

Namena ove klase je modelovanje eniteta instance skripta i koristi se za podršku pokretanju, zaustavljanju i nadgledanju rada skripta. Polja ove klase su:

- *String scriptPath* – ime skripta;
- *Globals globals* – okruženje interepretera za izvršavanja Lua skripta;
- *LuaValue chunk* – Učitani programski kod Lua skripta*;*
- *Future<?> execTask* – Nit u kojoj se izvršava skript;
- *String groupName* – Grupa uređaja nad kojima se primenjuje skript;
- *DeviceIdStruct deviceId* – Krajnji uređaj nad kojim se primenjuje skript.

Sledi opis funkcija realizovanih unutar klase *ScriptItem*.

- `public void startScriptExecution (ExecutorService executor) throws IllegalArgumentException`
    - Opis: Pokretanje rada konfiguracionog skripta
    - Parametri:
        - *ExecutorService executor* – bazen niti u kojem će se izvršavati skript
    - Izuzeci:
        - *IllegalArgumentException* – Ukoliko programski kod Lua skripta ne može biti interpretiran,najčešće zbog sintaksne greške
- `public void stopScriptExecution () throws IllegalArgumentException`
    - Opis: Zaustavljanje rada konfiguracionog skripta
    - Izuzeci:
        - *IllegalArgumentException* – Ukoliko skript koji se zaustavlja ne postoji

4.1.2.4 Klasa ScriptUpdateThread

Namena ove klase je provera da li je određena instanca skripta završila sa izvršavanjem i iniciranje slanja obaveštenja o završetku rada skripta ACS-u. Klasa nasleđuje klasu *Thread* i njen rad se zasniva na periodičnim upitima instancama skriptova da li su izvršili sve svoje funkcije. Sledi opis funkcije realizovane unutar klase *ScriptUpdateThread.*

- `public void run ()`
    - Opis: Periodična provera da li je skript završio izvršavanje

4.1.3 Proširenja Lua rečnika

Proširenja Lua rečnika predstavljaju klase koje nasleđuju klase iz *LuaJ* biblioteke koje omogućuju definisanje novih funkcija u Lua skript jeziku. Prilikom kreiranja nove instance skripta objekti ovih klasa se unose u meta-tabelu instance Lua interpretera uz niz karaktera koji predstavljaju ime funkcije u Lua jeziku.

Primer za to je:

Klasa *LuaAddObject* se u Lua skript jeziku predstavlja kao *addObject.*

Klase koje se nasleđuju iz *LuaJ* biblioteke su:

- *ZeroArgFunction* – funkcije u Lua skript jeziku koje se pozivaju bez argumenata;
- *OneArgFunction* - funkcije u Lua skript jeziku koje se pozivaju sa jednim argumentom;
- *TwoArgFunction* - funkcije u Lua skript jeziku koje se pozivaju sa dva argumenta;
- *ThreeArgFunction*- funkcije u Lua skript jeziku koje se pozivaju sa tri argumenta.

U Tabeli 7. nalaze se implementirane klase pomoću kojih se proširuje rečnik Lua skript jezika.

Ime klase	Broj argumenata	Ime funkcije u Lua jeziku	Značenje
LuaAddObject	2	*addObject*	Dodavanje objekta na krajnjem uređaju
LuaAddObjectBlocking	2	*addObjectAndWait*	Dodavanje objekta na krajnjem uređaju uz čekanje na izvršenje zahteva
LuaFactoryReset	0	*factoryReset*	Povratak na fabrička podešavanja
LuaFactoryResetBlocking	0	*factoryResetAndWait*	Povratak na fabrička podešavanja uz čekanje na izvršenje zahteva
LuaGetParameterAttribute	1	*getParameterAttribute*	Dobavljanje osobina parametara
LuaGetParameterAttributeBlocking	1	*getParameterAttributeAndWait*	Dobavljanje osobina parametara uz čekanje na izvršenje zahteva
LuaGetParameterValue	1	*getParameterValue*	Dobavljanje vrednosti parametara
LuaGetParameterValueBlocking	1	*getParameterValueAndWait*	Dobavljanje vrednosti parametara uz čekanje na izvršenje zahteva
LuaPrintDeviceId	0	*printDevice*	Prikaz uređaja na koji se primenljuje skript
LuaPrintGroupName	0	*printGroup*	Prikaz grupe uređaja na koje se primenuje skript
LuaRebootDevice	0	*reboot*	Ponovno pokretanje uređaja
LuaRebootDeviceBlocking	0	*rebootAndWait*	Ponovno pokretanje uređaja uz čekanje na izvršenje zahteva
LuaRemoveObject	2	*deleteObject*	Uklanjanje objekta na krajnjem uređaju
LuaRemoveObjectBlocking	2	*deleteObjectAndWait*	Uklanjanje objekta na krajnjem uređaju uz čekanje na izvršenje zahteva
LuaSetParameterAttribute	3	*setParameterAttribute*	Postavljanje osobina parametara
LuaSetParameterAttributeBlocking	3	*setParameterAttributeAndWait*	Postavljanje osobina parametara uz čekanje na izvršenje zahteva
LuaSetParameterValue	3	*setParameterValue*	Postavljanje vrednosti parametara
LuaSetParameterValueBlocking	3	*setParameterValueAndWait*	Postavljanje vrednosti parametara uz čekanje na izvršenje zahteva
LuaSleep	1	*sleep*	Pauziranje izvršavanja skripta za broj milisekundi iz argumenta

Tabela 7. Pregled klasa za proširenje rečnika Lua skript jezika

4.1.4 HTTP klijent

HTTP klijent u aplikaciji za izvršavanje skirptova se zasniva na blokirajućem redu zahteva u koji zahteve za konfiguracione poruke prema krajnjim uređajima unose pokrenute instance skriptova. Red se zasniva na običnoj FIFO strukturi, bez unošenja elemenata po prioritetu. Svaki od elemenata u redu mora da bude objekat klase koja implementira interfejs *HttpResponseCb* čime se obezbeđuje da se obrada odgovora na HTTP zahteve odvija u modulu za rukovanje skriptovima.

4.1.4.1 Klasa Communication

U klasi *Communication* implementirana je funkcionalnost za kontrolu blokirajućeg reda za HTTP zahteve i mehanizam slanja HTTP zahteva i rukovanja odgovorom. Klasa nasleđuje klasu *Thread.* Primljeni odgovor može biti obrađen kompletno u HTTP klijentu ukoliko je došlo do greške tokom obrade HTTP zahteva u ACS-u ili prosleđen modulu za rukovanje skriptovima na obradu. Sledi opis funkcija realizovanih unutar klase *Communication*.

- `public boolean addObj (CommunicationQueueItem el)`
  - Opis: Dodavanje HTTP zahteva u blokirajući red
  - Parametri:
    - *CommunicationQueueItem el* – HTTP zahtev
  - Povratna vrednost
    - *true* – Uspešno izvršenje funkcije
    - *false* - Neuspešno izvršenje funkcije
- `public boolean clearQueue ()`
  - Opis: Uklanjanje svih HTTP zahteva iz reda
  - Povratna vrednost
    - *true* – Uspešno izvršenje funkcije
    - *false* - Neuspešno izvršenje funkcije
- `public void startHTTPClient ()`

  Opis: Pokretanje HTTP klijenta
- `private void sendRequestToACS (String url, CommunicationQueueItem queueItem) throws IllegalStateException,IOException`
  - Opis: Slanje HTTP zahteva automatsko-konfiguracionom poslužiocu
  - Parametri:
    - *String url* – URL adresa automatsko-konfiguracionog poslužioca
    - *CommunicationQueueItem queue* – HTTP zahtev
  - Izuzeci:
    - *IOException* – Ukoliko dođe do greše prilikom uspostavljanja veze
    - *IllegalStateException* – Ukoliko je došlo do greške prilikom obrade HTTP zahteva na automatsko-konfiguracionom poslužiocu
- `private void handleResponse (InputStream responseBody,CommunicationQueueItem request) throws IllegalStateException`
  - Opis: Obrada HTTP odgovora
  - Parametri:
    - *InputStream responseBody* – telo HTTP odgovora
    - *CommunicationQueueItem request* – HTTP zahteve za koji je primljen odgovor
  - Izuzeci:
    - *IllegalStateException* – Ukoliko je došlo do greške prilikom obrade HTTP odgovora

4.1.4.2 Klasa CommunicationQueueItem

U ovoj klasi su enkapsulirani podaci potrebni za slanje HTTP zahteva i obradu primljenog odgovora. Pregled polja klase nalazi se u Tabeli 8.

Ime polja	Tip promenljive	Značenje
jsonMsg	*String*	Sadržaj JSON poruke u HTTP zahtevu
identifier	*String*	Identifikator zahteva dobijen u odgovoru na zahtev od poslužioca
msgType	*ACSRequestType*	Tip kontrolnog zahteva
isBlocking	*Boolean*	Indikator da li je incirani kontrolni zahtev blokirajući
scriptId	*Int*	Instanca skripte iz koje je iniciran zahtev
respCB	*HTTPResponseCb*	Objekat klase koja implementira povratne (*callback*) funkcije za obradu odgovora

Tabela 8. Prikaz polja klase *CommunicationQueueItem*

4.1.4.3 Interfejs HTTPResponseCb

Ovim interfejsom se definišu funkcije koje klase moraju implementirati da bi mogle biti korišćene za obradu odgovora na HTTP zahteve. Sledi opis funkcija koje moraju implementirati klase koje implementiraju interfejs *HTTPResponseCb*.

- `private void responseCB (String responseBody, CommunicationQueueuItem request)`
  - Opis: Funkcija za obradu HTTP odgovora
  - Parametri:
    - *String reposnseBody* – Telo HTTP odgovora
    - *CommunicationQueueItem request* – HTTP zahtev za koji je primljen odgovor

- `private void reportCB (ArrayList<ScriptReportItem> reports,CommunicationQueueItem request);`
  - Opis: Funkcija za obradu izveštaja o prethodno inicranom kontrolnom zahtevu
  - Parametri:
    - *ArrayList<ScriptReportItem> reports* – Izveštaji primljeni iz automatsko-konfiguracionog poslužioca
    - *CommunicationQueueItem request* – HTTP zahtev za koji je primljen odgovor

4.1.5 Podrška za JSON poruke

Podrška za JSON poruke se odnosi na klase uz pomoć kojih se korišćenjem *Gson* biblioteke kreiranju i raspakuju JSON poruke koje se šalju HTTP protokolom. Klase se mogu razvrstati u dve kategorije:

1. Klase za JSON poruke u HTTP zahtevima;
2. Klase za JSON poruke u HTTP odgovorima;

3. Klase JSON poruka za izveštaje o primenjenosti kontrolnih zahteva na uređaju.

Klase u prvoj kategoriji nasleđuju klasu *ActionRequestObject* koja sadrži polja vezana za uređaj (*devId*) ili grupu uređaja (*groupName*) na koje će se sadržaj JSON poruke primeniti.

Prikaz klasa iz ove kategorije sa opisom polja se nalazi u Tabeli 9.

Ime klase	Opis polja
AddObjectACSRequest	*String object* – Putanja objekta koji se dodaje *String parameterKey* – Identifikator konfiguracije
DeleteObjectACSRequest	*String object* – Putanja objekta koji se dodaje *String parameterKey* – Identifikator konfiguracije
FactoryResetACSRequest	Nema polja
GetParamAttrACSRequest	*String paramPath* – putanja parametra čije se osobine dobavljaju
GetParameterValueACSRequest	*String paramPath* – putanja parametra čija se vrednost dobavlja
RebootACSRequest	Nema polja
SetParamAttrACSRequest	*String paramPath* – putanja parametra čije se osobine postavljaju *Int notification* – nivo notifikacije za vrednost parametra *ArrayList<String> accessList* – lista entiteta koji mogu pristupiti parametru
SetParamValueACSRequest	*String paramPath* – putanja parametra čija se vrednost dobavlja *String value* – nova vrednost parametra *String parameterKey* – identifikator konfiguracije

Tabela 9. Pregled klasa za JSON poruke u HTTP zahtevima

Klase u drugoj kategoriji nasleđuju klasu *ActionResponseObject* koja sadrži polja vezana za identikator zahteva (*identifier*) dobijen od ACS-a i statusu (*status)* prethodno poslatog zahteva. Prikaz klasa iz ove kategorije sa opisom polja se nalazi u Tabeli 10.

Ime klase	Opis polja
AddObjectACSResponse	*Int addObjectStatus* – status uspešnosti dodavanja objekta
DeleteObjectACSResponse	*Int deleteObjectStatus* – status uspešnosti dodavanja objekta
FactoryResetACSResponse	Nema polja
GetParamAttributeACSResponse	*ArrayList<ParameterNameAttributes> parameters* – Lista objekata koji sadrže osobine traženog parametra
GetParameterValueACSResponse	*ArrayList<ParameterNameValue> parameters* – Lista objekata koji sadrže vrednosti traženih parametara
RebootACSResponse	Nema polja
SetParamAttributeACSResponse	Nema polja
SetParameterValueACSResponse	*Int setParamStatus* – status uspešnosti postavljanja vrednosti parametra

Tabela 10. Pregled klasa za JSON poruke u HTTP odgovorima

Kao što se može videti iz Tabele 9. i Tabele 10. polja u klasama za JSON poruke iz kategorija 1. i 2. se zasnivaju na podacima potrebnim za razmenu RPC metoda između ACS-a i CPE-a.

Trećoj kategoriji pripada klasa *ScriptReportItem* koja enkapsulira vrednosti koje su mogu dobiti kao izveštaj o primenjenosti zahteva na krajnjem uređaju

Pregled polja ove klase i opis na koje kontrolne poruke se odnose se nalazi u Tabeli 11.

Ime polja	**Značenje**	**Kontrolni zahtevi za koje se polje koristi**
DeviceIdStruct deviceId	Identifikator uređaja nad kojim je izvršen kontrolni zahtev	Svi kontrolni zahtevi
ArrayList<ParameterNameValue> values	Lista vrednosti parametara	GetParameterAttributes
ArrayList<ParameterNameAttributes> attrs	Lista osobina parametara	GetParameterValues
Int confstatus	Status uspešnosti konfiguracije	AddObject, DeleteObject, SetParameterValue
String faultString	Opis greške prilikom primene kontrolnog zahteva na uređaju	Svi kontrolni zahtevi ukoliko dođe do greške pri izvršavanju
Int faultCode	Status greške nastale prilikom primene kontrolnog zahteva na uređaju	Svi kontrolni zahtevi ukoliko dođe do greške pri izvršavanju

Tabela 11.Prikaz klase za JSON poruke za izveštaje o kontrolnim zahtevima

4.2 Proširenja na poslužiocu za automatsku konfiguraciju

Proširenja na poslužiocu za automatsku konfiguraciju se odnose na implementaciju podrške za CRUD operacije na skriptovima, objedinjenim u entitetu *Action* i odgovarajućem *SessionBean*-u za rukovanje skriptovima i povezivanje sa modulima za TR-069 komunikaciju i objavljivanje događaja sa krajnjih uređaja. Proširenja su implementirana korišćenjem istih tehnologija na kojima je implementiran ACS:

- *JBoss* aplikativni poslužilac[25];
- *Enterprise JavaBeans* za implementaciju biznis logike[26];
- *PostGreSQL* za bazu podataka[27].

4.2.1 Action entitet

Polja klase u *Action* entitetu se koriste za modelovanje konfiguracionog skripta i sprezanje sa ostatkom sistema. Detaljan opis polja i njihovo značenje nalaze se u Poglavlju 3.2.3. Druge implementirane klase na koje se oslanja *Action* entitet su:

- *ActionExecutionIndex;*
- *ScriptFilter*.

ActionExecutionIndex se koristi za čuvanje podataka o pokrenutim instancama skripta. Ova klasa sadrži polja:

- *cpeId* – Identifikator uređaja nad kojim se pokrenuti skript izvršava;
- *groupName* - Identifikator grupe uređaja nad kojim se pokrenuti skript izvršava;
- *executionIndex* – Identifikator pokrenute instance skripta dobijen iz aplikacije za izvršavanje skriptova.

ScriptFilter omogućava definisanje oblasti primene konfiguracionog skripta po određenom kriterijumu, mimo određenih uređaja ili grupa uređaja. Polja klase *ScriptFilter* su:

- *manufacturer* – proizvođač uređaja na koji se primenjuje skript;
- *productClass* – klasa uređaja na koji se primenjuje skript.

U *Action* entitetu implementirane funkcionalnosti se odnose na funkcije za postavljanje i dobavljanje vrednosti polja klase – *set* i *get* funkcije.

4.2.2 Rukovanje skriptovima

Funkcionalnosti za rukovanje skriptovima su implementirane u klasi *ActionScriptBean.* Ovaj *SessionBean* ne čuva stanje sesije sa drugim *SessionBean*-ovima (*Stateless bea*n). Sledi opis funkcija realizovanih unutar klase *ActionScriptBean*.

- `public void addNewActionScript (String filePath, String content, String event) throws NullPointerException, IllegalArgumentException`
  - Opis: dodavanje novog skripta u bazu podataka
  - Parametri:
    - *String filepath* – Identifikator skripta
    - *String content* – Programski kod skripta
    - *String event* - Događaj na koji će se skript pokrenuti
  - Izuzeci:
    - *NullPointerException* – Ukoliko neki od parametara nije prosleđen
    - *IllegalArgumentException* – Ukoliko neki od parametara nije validan,tj. ne može biti upisan u bazu podataka

- `public void addActionScriptToDevice (DeviceIdStruct cpeId, String filePath) throws NullPointerException,IllegalArgumentException`
  - Opis: Povezivanje skripta sa uređajem u bazi podataka
  - Parametri:
    - *DeviceIdStruct cpeId* – Uređaj sa kojim će se povezati skript
    - *String filePath* – Identifikator skripta
  - Izuzeci:

        - *NullPointerException* – Ukoliko neki od parametara nije prosleđen
        - *IllegalArgumentException* – Ukoliko neki od parametara nije validan,tj. ne može biti upisan u bazu podataka

- `public void addActionScriptToGroup (String groupName, String filePath) throws NullPointerException, IllegalArgumentException`
    - Opis: Povezivanje skripta sa grupom uređaja u bazi podataka
    - Parametri:
        - *String groupName* – Grupa uređaja sa kojima će skript biti povezan
        - *String filePath* – Identifikator skripta
    - Izuzeci:
        - *NullPointerException* – Ukoliko neki od parametara nije prosleđen
        - *IllegalArgumentException* – Ukoliko neki od parametara nije validan,tj. ne može biti upisan u bazu podataka

- `public void deleteActionScript (String filePath) throws NullPointerException, IllegalArgumentException`
    - Opis: uklanjanje skripta iz baze podataka uz prethodno zaustavljanje svih pokrenutih instanci skripta
    - Parametri:
        - *String filePath* – identifikator skripta koji se briše
    - Izuzeci:
        - *NullPointerException* – Ukoliko parametar nije prosleđen
        - *IllegalArgumentException* – Ukoliko skript tražen parametrom *filePath* ne može biti nađen u bazi podataka

- `public void addRuleToActionScript (String filePath, ScriptFilter rule) throws NullPointerException,IllegalArgumentException`
    - Opis: Pridruživanje kriterijuma za pokretanje skripta na osnovu proizvođača i klase uređaja
    - Parametri:
        - *String filePath* – Identifikator skripta
        - *ScriptFilter rule* - Uređena dvojka proizvodač i klasa uređaja
    - Izuzeci:
        - *NullPointerException* – Ukoliko neki od parametara nije prosleđen
        - *IllegalArgumentException* – Ukoliko skript tražen parametrom *filePath* ne može biti nađen u bazi podataka

- `public void updateActionScript (ScriptInfo scriptInfo) throws NullPointerException, IllegalStateException`
    - Opis: Izmena podataka određenog skripta u bazi podataka
    - Parametri:
        - *ScriptInfo scriptInfo* – podaci o skriptu koji trebaju biti izmenjeni
    - Izuzeci:
        - *NullPointerException* – Ukoliko neki od parametara nije prosleđen
        - *IllegalArgumentException* – Ukoliko je došlo do greške prilikom unosa podataka, do greške može prilikom upisa u bazu podataka ili prilikom zaustavljanja skriptova

- `public boolean addAndLinkNewAction (ScriptInfo scriptInfo)`
    - Opis: dodavanje skripta i povezivanje sa uređajem ili grupom uređaja u bazi podataka
    - Parametri:

        - *ScriptInfo scriptInfo* – Podaci o skriptu koji je potrebno dodati
    - Povratna vrednost
        - *true* – Uspešno izvršenje funkcije
        - *false* - Neuspešno izvršenje funkcije

- `public boolean stopAndUnlinkAllCPEActions (CPE cpe)`
    - Opis: zaustavljanje svih pokrenutih instanci skriptova za određeni uređaj i raskidanje veze u bazi podataka između uređaja i skripta. Poziva se prilikom brisanja uređaja iz baze podataka
    - Parametri:
        - *CPE cpe* – Uređaj za koji se vrši zaustavljanje skriptova i raskidanje veza
    - Povratna vrednost
        - *true* – Uspešno izvršenje funkcije
        - *false* - Neuspešno izvršenje funkcije

- `public boolean stopAndUnlinkAllGroupActions (Group group)`
    - Opis: zaustavljanje svih pokrenutih instanci skriptova za određenu grupu uređaja i raskidanje veze u bazi podataka između grupe uređaja i skripta. Poziva se prilikom brisanja grupe uređaja iz baze podataka
    - Parametri:
        - *Group group* – Grupa uređaja za koji se vrši zaustavljanje skriptova i raskidanje veza
    - Povratna vrednost
        - *true* – Uspešno izvršenje funkcije
        - *false* - Neuspešno izvršenje funkcije

- `public List<ScriptInfo> getAllScriptsInfo ()`
    - Opis: dobavljanje podataka o svim skriptovima u bazi podataka
    - Povratna vrednost – Lista objekata koji sadrže podatke o skriptovima

- `public boolean runScript (String scriptName)`
    - Opis: Pokretanje rada skripta za sve pridružene uređaja i grupe uređaja
    - Parametri:
        - *String scriptName* – identifikator skripta koji se pokreće
    - Povratna vrednost
        - *true* – Uspešno izvršenje funkcije
        - *false* - Neuspešno izvršenje funkcije

- `public boolean stopScript (String scriptName)`
    - Zaustavljanje rada skripta za sve pridružene uređaja i grupe uređaja
    - Parametri:
        - *String scriptName* – identifikator skripta koji se zaustavlja
    - Povratna vrednost
        - *true* – Uspešno izvršenje funkcije
        - *false* - Neuspešno izvršenje funkcije

- `public void publishScriptStop (String scriptName, String scriptId)`
    - Opis: Objava kraja izvršavanja skripta iz aplikacije za izvršavanje skripta
    - Parametri:
        - *String scriptName* – identifikator skripta
        - *String scriptId* – identifikator pokrenute instance skripta u aplikaciji za izvršavanje skriptova

4.2.3 Objavljivanje događaja

Objavljivanje događaja omogućava automatsko pokretanje rada skripta za koje je određeni događaj definisan. Prilikom objave događaja pronalaze se svi skriptovi koji za zadati uređaj ili grupu uređaja imaju pridružen događaj. Pritom se posle nalaženja skriptova proverava da li je nađeni skript koji treba pokrenuti već pokrenut za dati uređaj ili grupu uređaja, u cilju onemogućavanja višestrukog pokretanja istog skripta za isti uređaj ili grupu uređaja. Radi smanjenja opterećenja konfiguracionog poslužioca funkcije opisane u ovom poglavlju se izvršavaju asinhrono. Sledi opis funkcija realizovanih za potrebe funkcionalnosti objave događaja.

- `public void publishCPEEvent (DeviceIdStruct cpeId, String event)`
  - Opis: Objava događaja za krajnji uređaj u sistemu
  - Parametri:
    - *DeviceIdStruct cpeId* – identifikator krajnjeg uređaja koji objavljuje događaj
    - *String event* – Tip događaja koji se desio na krajnjem uređaju

- `public void publishGroupEvent (String groupName, String event)`
  - Opis: Objava događaja za grupu uređaja u sistemu
  - Parametri:
    - *String groupName* – identifikator grupe uređaja koji objavljuju događaj
    - *String event* – Tip događaja koji se desio na krajnjem uređaju

4.2.4 Proširenja REST servisa modulom za skriptove

Proširenja REST servisa podrškom za aplikaciju za izvršavanje konfiguracionih skriptova enkapsulirano je u klasi *ActionScriptServices*. Sve funkcije u klasi se aktiviraju ukoliko se radi o HTTP post zahtevu sa tipom zahteva *application/json* HTTP zaglavlju. U ovom poglavlju opisane su implemetirane funkcije. Detaljniji opis svakog od parametra funkcije i povratne vrednosti se može naći u Poglavlju 4.1.4. Sledi opis funkcija realizovanih za potrebe proširenja REST web servisa spregama za aplikaciju za izvršavanje skriptova.

- `public AddObjectACSResponse actionAddObject (AddObjectACSRequest request)`
  - Opis: Dodavanje objekta u repozitorijumu krajnjeg uređaja ili grupe uređaja
  - Parametri:
    - *AddObjectACSRequest request* – podaci neophodni za slanje zahteva krajnjem uređaju ili grupi uređaja
  - Povratna vrednost
    - *AddObjectACSResponse* – Indikacija da li će zahtev biti poslat ili ne i da li je uspeo zahtev za trenutno povezivanje sa kranjim uređajima

- `public DeleteObjectACSResponse actionDeleteObject (DeleteObjectACSRequest request)`
  - Opis: Uklanjanje objekta u repozitorijumu kranjeg uređaja ili grupe uređaja

  - Parametri:
    - *DeleteObjectACSRequest request* – podaci neophodni za slanje zahteva krajnjem uređaju ili grupi uređaja
  - Povratna vrednost
    - *DeleteObjectACSResponse* – Indikacija da li će zahtev biti poslat ili ne i da li je uspeo zahtev za trenutno povezivanje sa kranjim uređajima

- `public FactoryResetACSResponse actionFactoryReset (FactoryResetACSRequest request)`
  - Opis: Povratak na fabrička podešavanja krajnjeg uređaja ili grupe uređaja
  - Parametri:
    - *FactoryResetACSRequest request* – podaci neophodni za slanje zahteva krajnjem uređaju ili grupi uređaja
  - Povratna vrednost
    - *FactoryResetACSResponse* – Indikacija da li će zahtev biti poslat ili ne i da li je uspeo zahtev za trenutno povezivanje sa kranjim uređajima

- `public RebootACSResponse actionReboot (RebootACSRequest request)`
  - Opis: Ponovno pokretanje krajnjeg uređaja ili grupe uređaja
  - Parametri:
    - *RebootACSRequest request* – podaci neophodni za slanje zahteva krajnjem uređaju ili grupi uređaja
  - Povratna vrednost
    - *RebootACSResponse* – Indikacija da li će zahtev biti poslat ili ne i da li je uspeo zahtev za trenutno povezivanje sa kranjim uređajima

- `public GetParameterValueACSResponse actionGetParameterValue (GetParamValueACSRequest request)`
  - Opis: Dobavljanje vrednosti parametara sa krajnjeg uređaja ili grupe uređaja
  - Parametri:
    - *GetParamValueACSRequest request* – podaci neophodni za slanje zahteva krajnjem uređaju ili grupi uređaja
  - Povratna vrednost
    - *GetParamValueACSResponse* – Indikacija da li će zahtev biti poslat ili ne i da li je uspeo zahtev za trenutno povezivanje sa kranjim uređajima

- `public SetParameterValueACSResponse actionSetParameterValue (SetParamValueACSRequest request)`
  - Opis: Postavljanje vrednosti parametara sa krajnjeg uređaja ili grupe uređaja
  - Parametri:
    - *SetParamValueACSRequest request* – podaci neophodni za slanje zahteva krajnjem uređaju ili grupi uređaja
  - Povratna vrednost
    - *SetParamValueACSResponse* – Indikacija da li će zahtev biti poslat ili ne i da li je uspeo zahtev za trenutno povezivanje sa kranjim uređajima

- `public GetParameterAttributeACSResponse actionGetParameterAttribute (GetParamAttrACSRequest request)`
  - Opis: Dobavljanje osobina parametara sa krajnjeg uređaja ili grupe uređaja
  - Parametri:
    - *GetParameterAttrACSRequest request* – podaci neophodni za slanje zahteva krajnjem uređaju ili grupi uređaja
  - Povratna vrednost

        - *GetParameterAttributeACSResponse* – Indikacija da li će zahtev biti poslat ili ne i da li je uspeo zahtev za trenutno povezivanje sa kranjim uređajima

- `public SetParameterAttributeACSResponse actionSetParameterAttribute (SetParamAttrACSRequest request)`
    - Opis: Postavljanje osobina parametara sa krajnjeg uređaja ili grupe uređaja
    - Parametri:
        - *SetParameterAttrACSRequest request* – podaci neophodni za slanje zahteva krajnjem uređaju ili grupi uređaja
    - Povratna vrednost
        - *SetParameterAttributeACSResponse* – Indikacija da li će zahtev biti poslat ili ne i da li je uspeo zahtev za trenutno povezivanje sa kranjim uređajima

- `public String notifyScriptStop (ScriptDataObject request)`
    - Opis: Obaveštavanje o završetku izvršavanja skripta u aplikaciji za izvršavanje skriptova
    - Parametri:
        - `ScriptDataObject` *request* – podaci neophodni jednoznačno određivanje o kom skriptu i instanci tog skritpa se radi
    - Povratna vrednost
        - *String* – Tekst razumljiv čoveku (*Human-readable*) za opis uspešnosti obrade obaveštenja

4.2.5 Integracija sa TR069 komunikacionim modulom

Za potrebe integracije mehanizma za rukovanje skriptovima sa modulom za TR-069 komunikaciju implementirane su klase:

- *ScriptReportItem* – enkapsulira vrednosti iz SOAP odgovora dobijenih od TR-069 klijenata,
- *DeviceRequestItem* – opisuje zahtev za slanje kontrolne poruke krajnjem uređaju radi praćenja izvršenja zahteva i prosleđivanja izveštaja aplikaciji za izvršavanje skriptova,
- *GroupRequestItem* – enkapsulira sve zahteve ka krajnjim uređajima iz određene grupe. Koristi se za praćenje izvršavanja kontrolnih zahteva na krajnjim uređajima i obaveštavanje aplikacije kada se svi zahtevi izvrše.

Takođe, klasa *TR069Request*, koja opisuje kontrolne zahteve prema krajnjim uređajima je proširena poljem *isScriptReq*, čime je obezbeđeno da se za svaki zahtev prema krajnjem uređaju zna da li je iniciran od mehanizma za izvršavanje skriptova ili nekog drugog modula u ACS-u. Sledi opis funkcija realizovanih za potrebe inicijacije slanja izveštaja aplikaciji za izvršavanje skriptova.

- `public void sendDeviceReport (DeviceRequestItem item, String requestID)`
    - Opis: Slanje izveštaja o izvršenju kontrolnog zahteva na jednom krajnjem uređaju

- Parametri:
  - *DeviceRequestItem item* – izveštaj koji se šalje kontrolnoj aplikaciji
  - *String requestID*- identifikator kontrolnoh zahteva u aplikaciji dobijen od konfiguracionog poslužioca prilikom iniciranja zahteva

- `public void sendGroupReport (GroupRequestItem item)`
  - Opis: Slanje izveštaja o izvršenju kontrolnog zahteva nad grupom krajnjih uređaja
  - Parametri:
    - *GroupRequestItem item* – izveštaj koji se šalje kontrolnoj aplikaciji

5. Ispitivanje i evaluacija

Rešenje mehanizma izvršavanja konfiguracionih skriptova je integrisano u Insight ACS IoT poslužilac. Ispitivanje funkcionalnosti i performansi je obavljeno sa:

- Web aplikacijom;
- Ispitnim okruženjem sa 50 STB uređaja povezanih na poslužilac za automatsku konfiguraciju;
- Automatsko-konfiguracionim poslužiocem pokrenutim na *Dell E5430* laptop računaru, sa *Intel i5-3210M* procesorom i RAM memorijom od 4 GB. Poslužilac se koristi pod *Windows 7* operativnim sistemom.

Ciljevi ispitivanja:

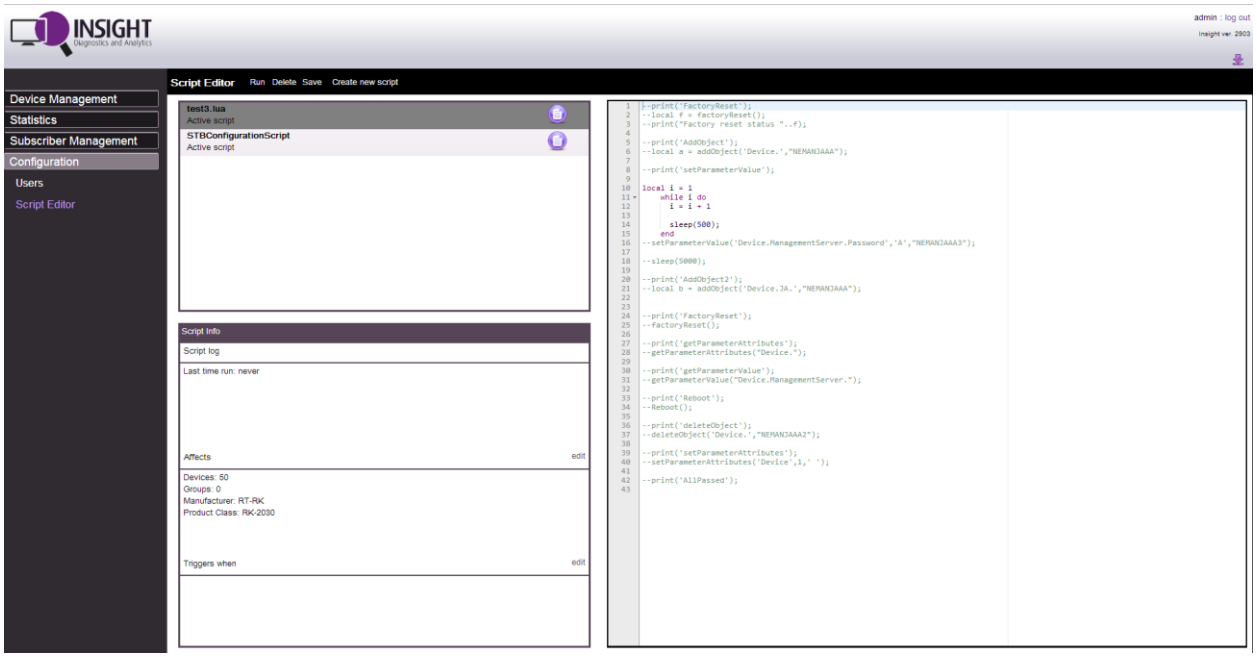
- Provera ispravnosti rada mehanizma za izvršavanje skriptova;
- Merenje perfomansi mehanizma za izvršavanje skriptova;
    - Brzina izvršavanja TR-069 konfiguracionih scenarija;
    - Poređenje sa drugim mehanizmima za sprezanje sa automatsko-konfiguracionim poslužiocem, u ovom slučaju poređenje REST web servisa sa RPC spregom;
    - Merenje potrošnje memorije u odnosu na broj konfiguracionih skriptova koji se istovremeno izvršavaju.

5.1 Provera ispravnosti mehanizma za izvršavanje skriptova

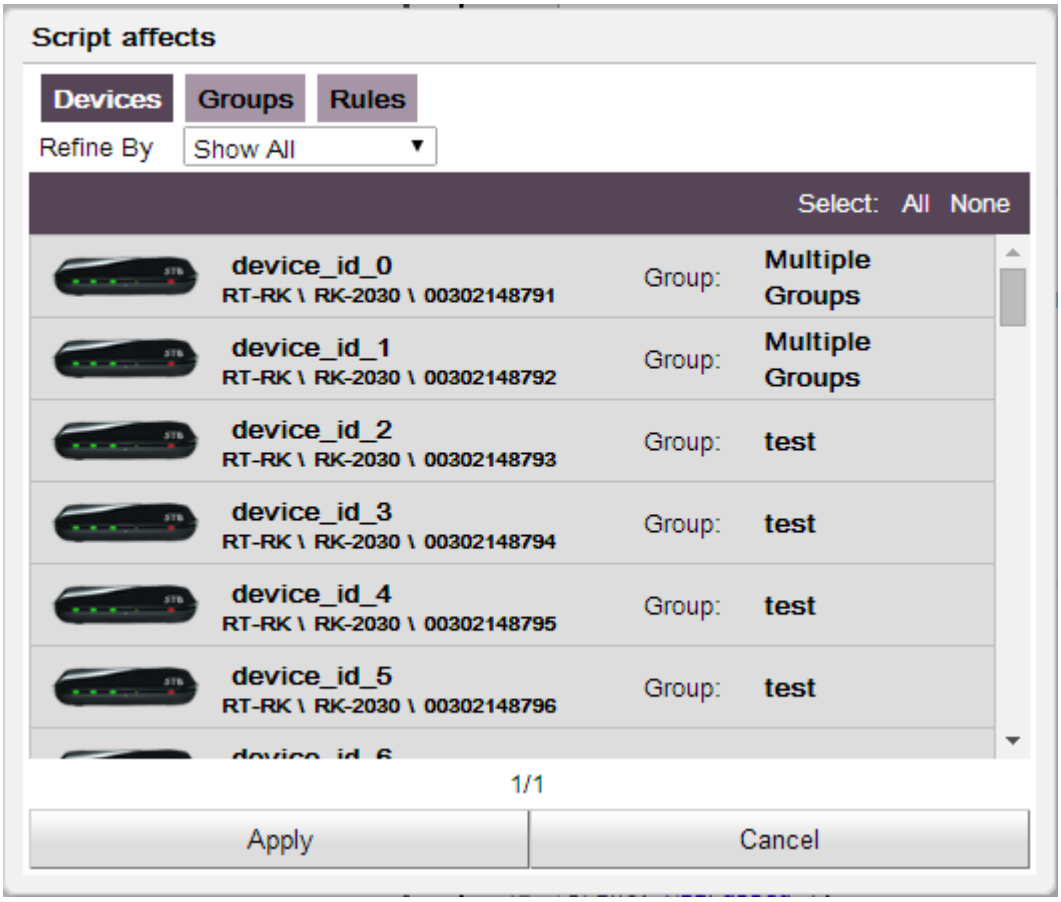
Ispitivanje funkcionalnosti poslužioca upotrebom web aplikacija i posmatranjem ponašanja uređaja tokom izvršavanja konfiguracionog skripta, prikazanog na Slici 21 . Web aplikacija je implementirana korišćenjem GWT okruženja. Skriptovi su primenjivani na različit broj pojedinačnih uređaja i grupa uređaja. Ispitane su funkcionalnosti za ručno pokretanje i

zaustavljanje skripta, kao i provera ispravnosti rada mehanizma za pokretanje skriptova prilikom objave događaja od strane krajnjeg uređaja.

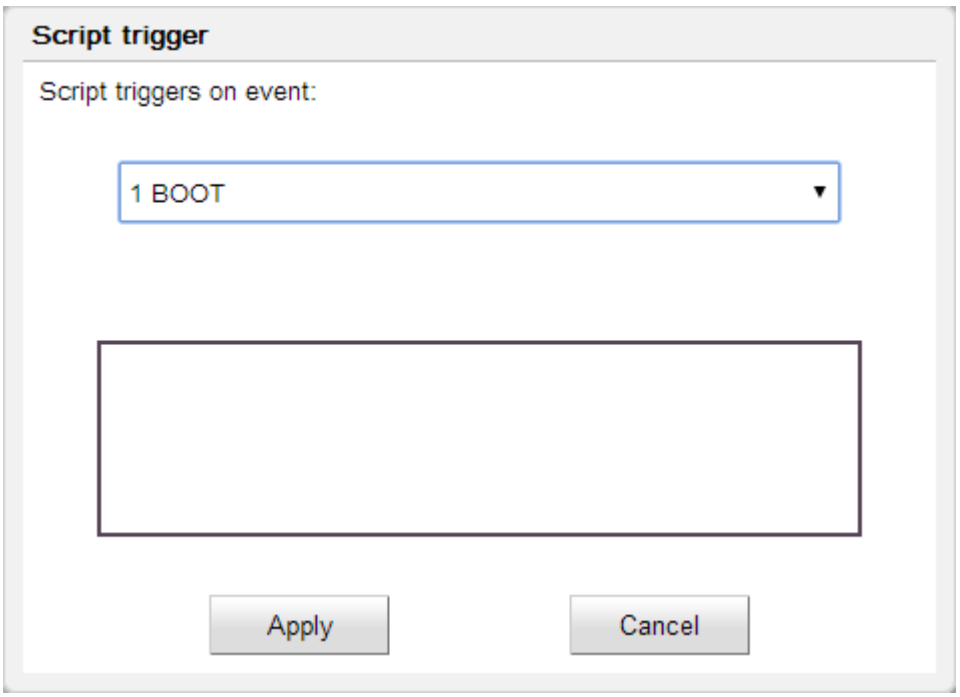
Na Slici 18, Slici 19, i Slici 20. nalazi prikaz komponenti web aplikacije pomoću koje su ispitane funkcionalnosti mehanizma za izvršavanje skriptova.



Slika 18.Prikaz web stranice za upravljanje konfiguracionim skriptovima



Slika 19. Dijalog za definisanje oblasti primene skripta



Slika 20. Dijalog za definisanje događaja na koji se skript pokreće

```
print("Executing initial configuration script");

local paramKeyRef = "InitConfPackage";

local key = getParameterValueAndWait("Device.ManagementServer.ParameterKey");

if(key == paramKeyRef) then
    print("Device already configured "..key);
else
    setParameterValue("Device.DeviceInfo.ProvisioningCode","1.23",paramKeyRef);
    setParameterValue("Device.ManagementServer.ConnectionRequestUsername","userDevice1",paramKeyRef);
    setParameterValue("Device.ManagementServer.ConnectionRequestPassword","passDevice2",paramKeyRef);
    setParameterValue("Device.ManagementServer.PeriodicInformEnable","true",paramKeyRef);
    setParameterValue("Device.ManagementServer.PeriodicInformInterval","10",paramKeyRef);
    setParameterValue("Device.Wifi.1.Enable","false",paramKeyRef);
    setParameterValue("Device.DeviceInfo.ProvisioningCode","1.23",paramKeyRef);
    setParameterAttributes("Device.ManagementServer.",2,"deviceMW");
    local confKey = getParameterValueAndWait("Device.ManagementServer.ParameterKey");

    if(confKey == paramKeyRef) then
        print ("Device configured successfully");
    else
        print("Error during configuration!!");
        reboot();
    end
end

print("Ending execution of initial configuration script");
```

Slika 21. Primer skripta za konfigurisanje krajnjeg uređaja

Ručnim isptivanjem korišćenjem web aplikacije zaključeno je da mehanizam za izvršavanje skriptova ispunjava zahtevane funkcionalnosti.

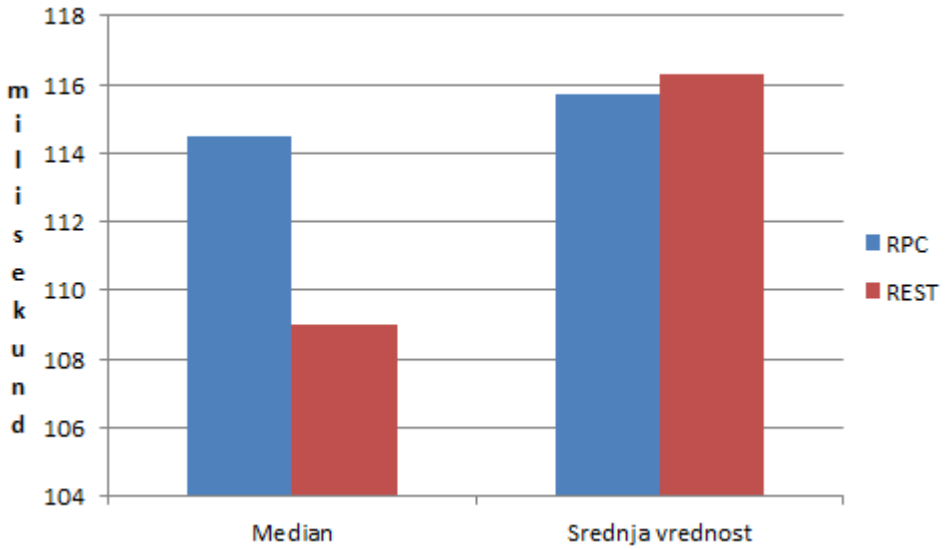
5.2 Poređenje REST web servisa u odnosu na RPC spregu

Prilikom ovog merenja sa klijentskih aplikacija poslati su HTTP zahtevi koji iniciraju izvršavanje iste funkcionalnosti u jezgru ACS-a. Cilj ispitnog slučaja je poređenje brzine odziva REST web servisa i RPC sprege. Klijentske aplikacije su:

1. Web aplikacija u slučaju RPC sprege,

2. Aplikacija za izvršavanje skriptova u slučaju REST web servisa.

Jedno merenje predstavlja vreme od iniciranja slanja HTTP zahteva do trenutka primanja HTTP odgovora. Tokom isptivanja vršeno je 10 merenja. Rezultati su prikazani na Slici 22.



Slika 22. Uporedni prikaz vremena odziva za RPC i REST

Kao što se može videti sa Slike 22. oba mehanizma imaju približno ista vremena odziva, s tim što RPC mehanizam ima veća odstupanja od srednje vrednosti brzine odziva što se može videti po rezultatima dobijenim korišćenjem *Median* algoritma za usrednjavanje. Na osnovu rezultata može se doneti zaključak da sprezanje aplikacije za izvršavanje skriptova sa ACS-om korišćenjem REST web servisa ima zadovoljavajuće performanse u odnosu na druge sprege.

5.3 Performanse mehanizma za izvršavanje skriptova

Merenje perfomansi implementiranog rešenja se zasniva na merenju vremena i potrošnje memorije za izvršenje različitih scenarija konfiguracije krajnjih uređaja pri različitim uslovima. Promenljivi uslovi u sistemu su:

- Broj uređaja u sistemu;
- Opterećenje ACS-a;
- Da li se mehanizam za izvršavanje skriptova izvršava lokalno ili distribuirano.

Za potrebe evaluacije efekta gore navedenih uslova na brzinu izvršavanja funkcionalnosti unutar implementiranog sistema izvršavani se sledeći ispitni slučajevi:

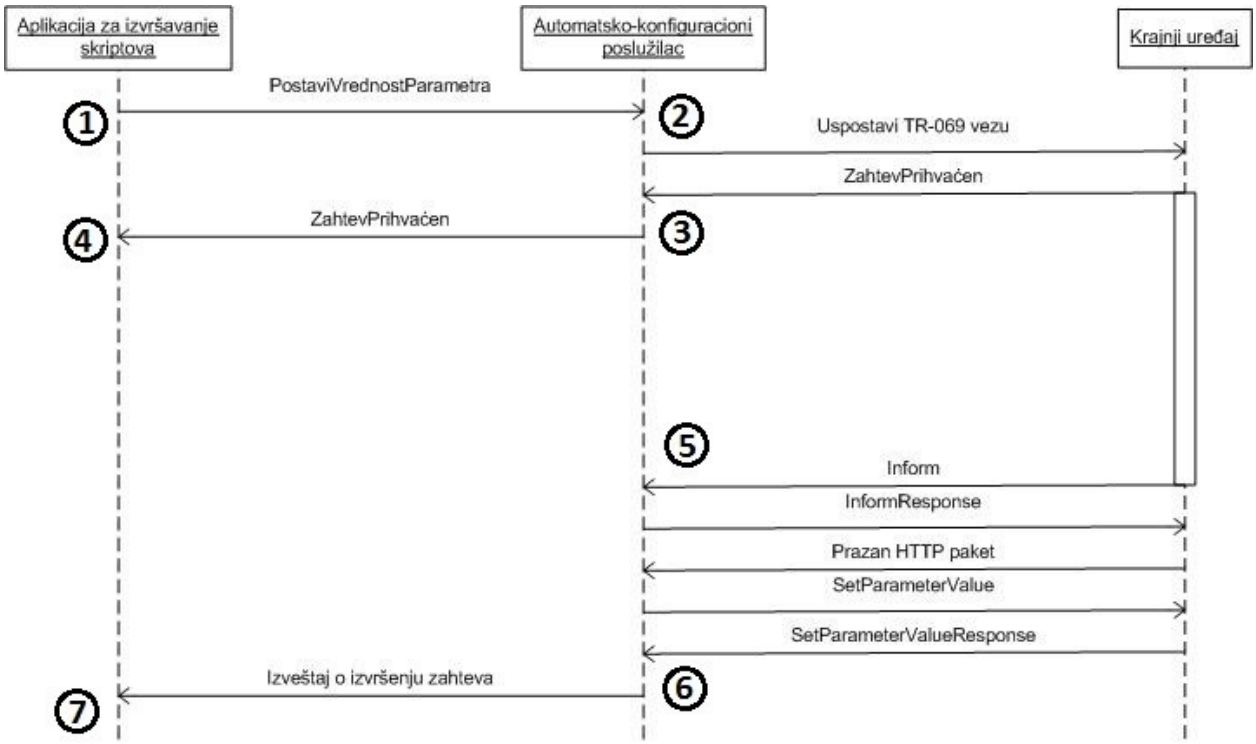
- Merenje brzine prenosa podataka između aplikacije za izvršavanje skirptova i ACS-a u odnosu na to da li se izvršavaju na istoj ili različitim platformama;
- Merenje brzine izvršavanja konfiguracionih zahteva u odnosu na broj skriptova koje se istovremeno izvršavaju.

5.3.1 Merenje brzine izvršavanja konfiguracionih scenarija

Prilikom merenja brzine izvršavanja konfiguracionih scenarija merene su brzine izvršavanja funkcionalnosti:

- Unutar aplikacije za izvršavanje skriptova;
- Prenosa informacija između aplikacije i ACS-a;
- Unutar ACS-a;
- Prenosa informacija između ACS-a i CPE-a;
- Unutar CPE-a.

Prilikom ispitivanja izvršavan je scenario u kome se na krajnji uređaj postavlja vrednost parametra: *Device.ManagementServer.Password* . Prilikom uspostavljanja veze između CPE-a i ACS-a korišćen je mehanizam zahteva za trenutno povezivanje (*ConnectionRequest*) da bi se izbeglo kašnjenje prilikom uspostavljanja TR-069 komunikacione sesije u sistemu. Merenje je vršeno u sedam tačaka tokom izvršavanja scenarija. Prikaz toka ispitnog slučaja sa mernim tačkama se nalazi na Slici 23.



Slika 23. Prikaz ispitnog scenarija za merenje brzine izvršenja zahteva

Značenje mernih tačaka sa Slike 23. su:

- Tačka 1 – Iniciranje slanja konfiguracionog zahteva;
- Tačka 2 – Vreme potrebno za prenos zahteva od aplikacije do automatsko-konfiguracionog poslužioca;

- Tačka 3 – Vreme potrebno za razmenu poruka za mehanizam zahteva za trenutno povezivanje;
- Tačka 4 – Vreme potrenno za slanje odgovora aplikaciji od trenutka prijema odgovora na zahtev za trenutno povezivanje;
- Tačka 5 – Trenutak započinjanja TR-069 komunikacione sesije;
- Tačka 6 – Trenutak prijema odgovora na zahtev za postavljanje vrednosti parametra;
- Tačka 7 – Vreme potrebno za obradu odgovora na zahtev za postavljanje parametra i prosleđivanje izveštaja aplikaciji za izvršavanje skriptova.

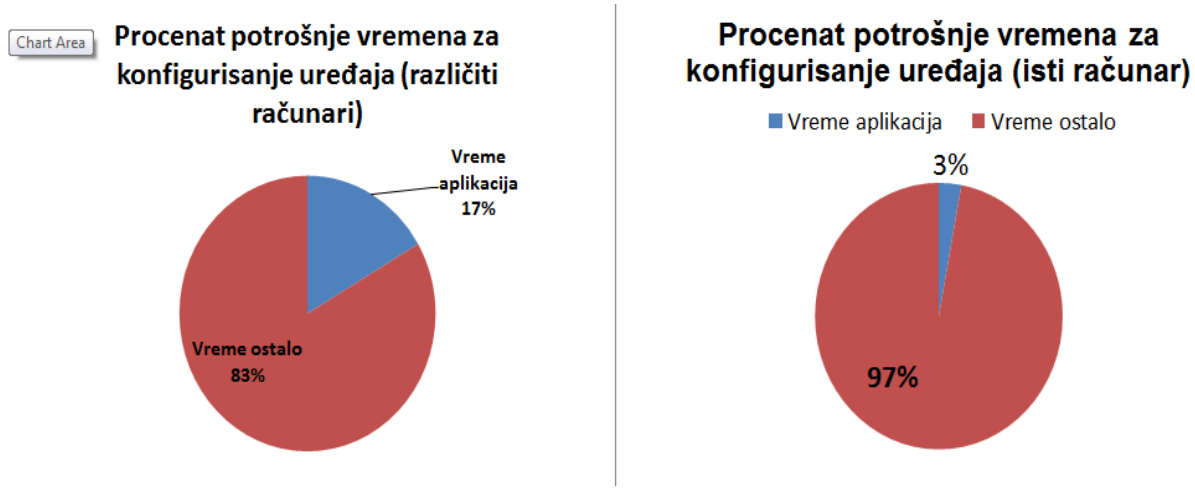
Prilikom isptivanja vršeni su ispitni slučajevi radi utvrđivanja faktora koji utiču na brzinu rada sistema. Ti faktori su stepen opterećenja ACS-a i mesto rada aplikacije u odnosu na ACS , tj. da li se izvršava na istom ili na drugom računaru.

Rezultati od važnosti su razvojeni tako da označavaju da li se odnose na aplikaciju za izvršavanje skriptova ili na ostatak *Insight* sistema. Uvedena imena su:

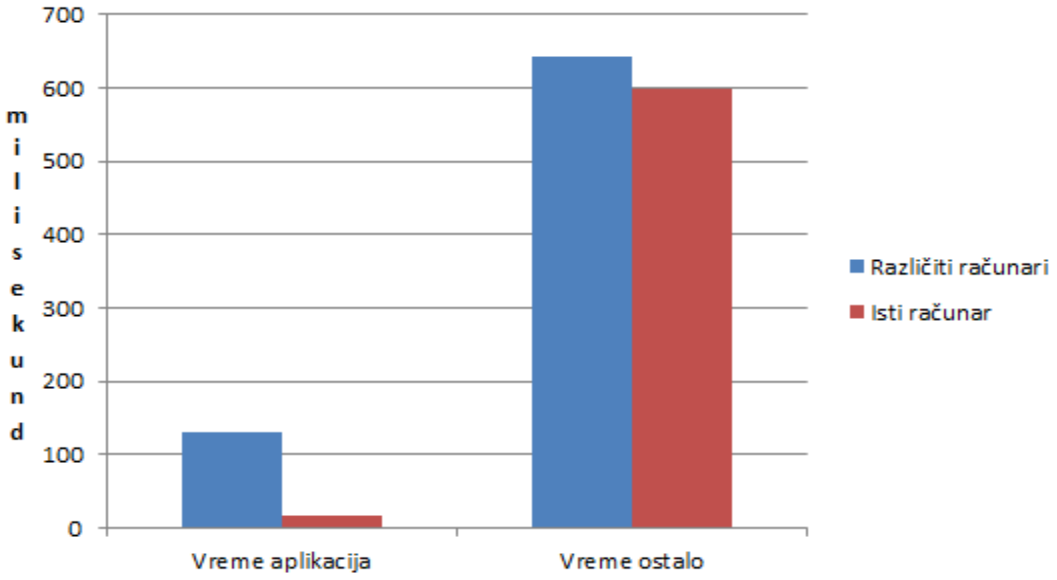
- Vreme aplikacije, vreme potrebno za razmenu podataka između ACS-a i aplikacije za izvršavanje skriptova kao i vreme potrebno za obradu primljenih podataka u aplikaciji – zbir vremena između tačaka 1 i 2; 3 i 4;6 i 7;
- Vreme ostalo – vreme potrebno za obradu podataka na ACS-u i krajnjem uređaju kao i na komunikaciju između njih - zbir vremena između tačaka 2 i 3;4 i 5;5 i 6;

5.3.1.1 Merenje brzine rada u odnosu na lokaciju izvršavanja aplikacije

Prilikom ovog merenja ispitivana je brzina odziva sistema u slučaju izvršavanja jednog konfiguracionog skripta nad jednim krajnjim uređajem. U prvom slučaju aplikacija za izvršavanje skriptova i ACS rade na istom računaru, dok u drugom slučaju rade na različitim računarima koji se nalaze u istoj lokalnoj mreži. Pre izvršenja ispitivanja sa dva računara sistemsko vreme na oba računara je sinhronizovano korišćenjem programa *NetTime[28]*. Rezultati merenja su prikazani na Slici 24 i na Slici 25.



Slika 24. Procentualni prikaz utroška vremena delova sistema tokom konfiguracije



Slika 25.Utrošak vremena delova sistema tokom konfiguracije

Kao što se može videti sa Slike 24. i Slike 25. mesto rada aplikacije i ACS-a je bitan faktor koji utiče na brzinu odziva sistema pre svega zbog utroška vremena na komunikaciju između aplikacije i ACS-a. Ukoliko rade na istom računaru utrošak vremena za komunikaciju je zanemarljiv, dok u drugom slučaju predstavlja značajan vremenski period ukoliko se gleda ceo proces konfiguracije. U slučaju izvršavanja aplikacije na drugoj platformi u odnosu na ACS, utrošak vremena koje unosi aplikacija je podnošljiv budući da je u tom slučaju obrada konfiguracionih podataka dugotrajna, pa vreme potrošeno na komunikaciju između aplikacije i ACS-a ne unosi preveliko kašnjenje. Rešenje [9] ne unosi kašnjenje u sistem zbog implementacije mehanizma za izvršavanje skriptova u sklopu ACS-a i iz tog ugla je optimalnije u odnosu na naše, međutim distribucija takvog rešenja je otežana, jer se u slučaju dugotrajne obrade podataka tokom konfiguracije opterećuje ceo ACS, a u našem slučaju to može biti druga platforma, čime se ne ugrožava brzina odziva ACS-a.

5.3.1.2 Merenje brzine rada u odnosu na stepen opterećenja ACS-a

Prilikom ovo isptivanja ACS i aplikacija za izvršavanje skriptova rade na istom računaru. Pokreće se izvršanje različitog broja skriptova istovremeno koje konfigurišu po jedan uređaj po scenariju opisanom u Poglavlju 5.2.2. Akcenat merenja je na odnosu utorška vremena koje unosi konfiguracija korišćenjem skriptova i vremena potrebnog da ACS obradi konfiguracione zahteve. Rezultati merenja prikazani su u Tabeli 12.

Broj pokrenutih skriptova	10	20	30	40	50
Vreme aplikacije	370 ms	724,5 ms	454,5 ms	609 ms	363 ms
Vreme ostalo	3387 ms	4198 ms	16533,5 ms	21425,5 ms	23192,5 ms

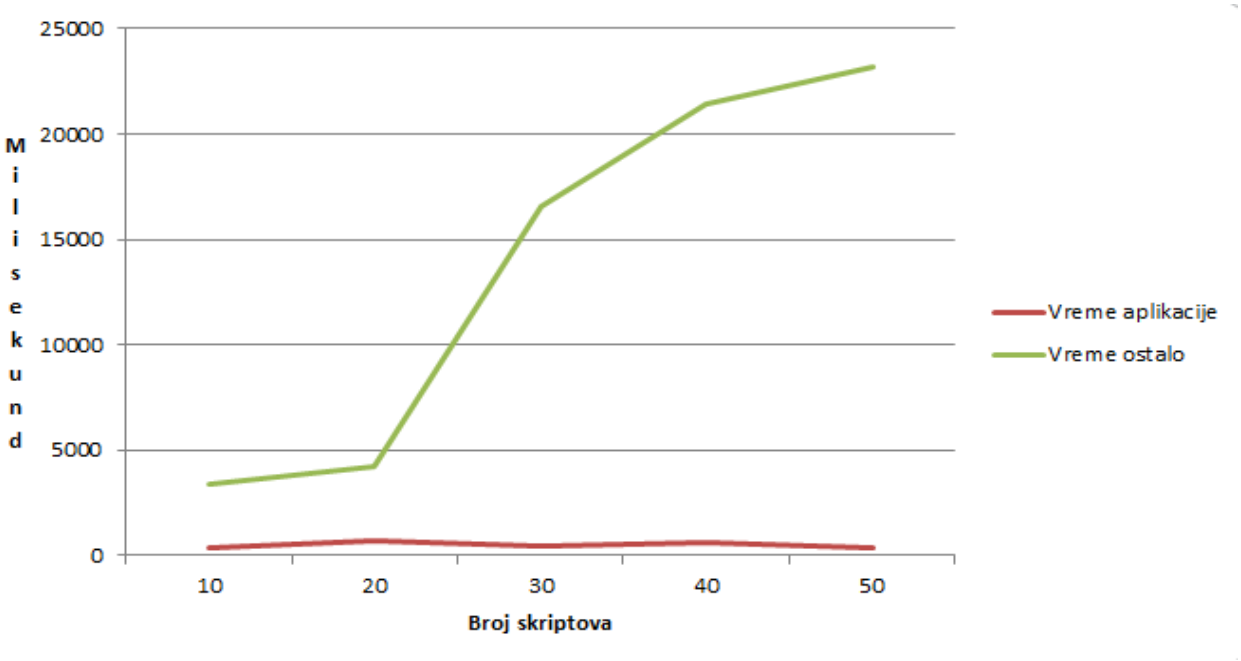
Tabela 12. Prikaz rezultata brzine primene konfiguracije u odnosu na broj skriptova

Vrednosti u Tabeli 12. predstavljaju srednje vreme potrebno za obradu jednog konfiguracionog zahteva. Srednja vremena su izračunata primenom *Median* algoritma. Kao što se može videti vreme aplikacije raste sa brojem skriptova, međutim rast nije linearan već je direktno zavistan od opterećenja ACS-a.

Vreme aplikacije naglo raste u slučaju 20 skriptova jer je poslužilac za automatsku konfiguraciju sposoban da brzo obradi taj broj skriptova i u tom slučaju aplikacija predstavlja usko grlo zbog većeg broja brzo pristiglih poruka koje treba obraditi unutar aplikacije. Problem uskog grla u aplikaciji je moguće rešiti dodavanjem struktura podataka korišćenjem kojih bi se optimizovalo vreme potrebno za obradu pristglof izveštaja. Uloga dodatih struktura bi bila da mapiraju pristigle izveštaje na instance skriptova kojima su izveštaji namenjeni, čime bi se smanjilo vreme potrebno za asocijaciju pristiglog izveštaja i skripta koji čeka na taj izveštaj. U slučaju sa 30, 40 i 50 skriptova vreme obrade na strani ACS-a je znatno veće stoga on postaje usko grlo, dok aplikacija ima dovoljno resursa da obradi pristigle poruke brzo.

Rešenje problema uskog grla na strani ACS-a se može rešiti klasterovanjem poslužioca čime bi se pristigli odgovori od CPE-ova obrađivali na više platformi. Budući da izmeren vremenski utrošak koje unosi slanje izveštaja aplikaciji ne raste sa brojem istovremeno poslatih zahteva za konfigurisanje uređaja autor smatra ovakav način implementacije mehanizma za

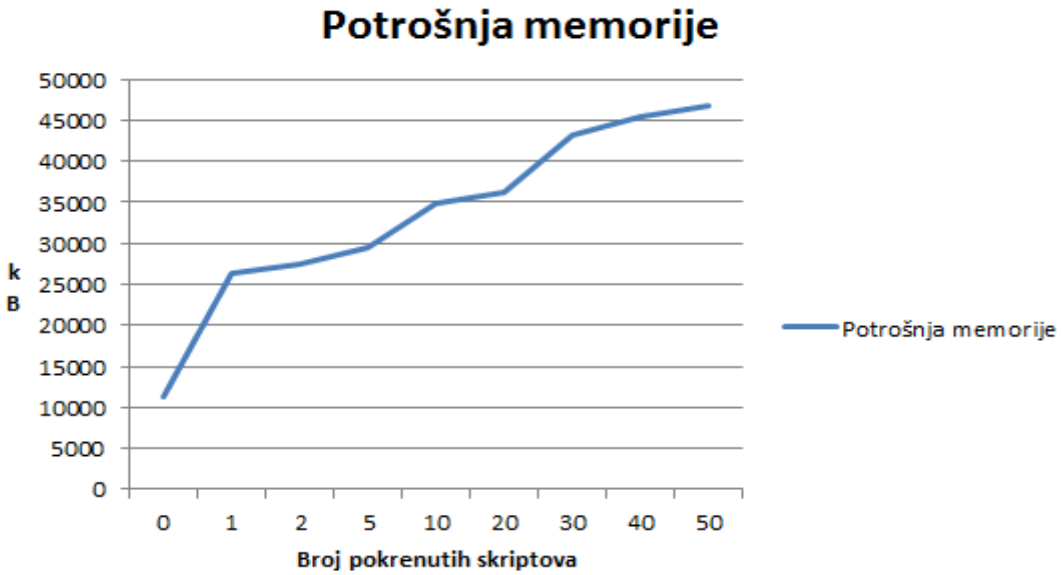
izvršavanje skriptova ne predstavlja veliko opterećenje za sistem za automatsku konfiguraciju uređaja. Grafički prikaz ovih vrednosti nalazi se na Slici 26.



Slika 26. Grafički prikaz odnosa vremena tokom konfiguracije u odnosu na broj skriptova

5.4 Potrošnja memoriije u odnosu na broj skriptova

Merenje potrošnje memorije aplikacije za izvršavanje skriptova je vršeno u odnosu na broj skriptova koji se trenutno izvršavaju. Rezultati merenja su prikazani na Slici 27.



Slika 27. Potrošnja memorije aplikacije u odnosu na broj skriptova

Kao što se može videti sa grafika na Slici 27. potrošnja memorije linearno raste u odnosu na broj skriptova koji se izvršavaju u aplikaciji. Najveći skok u potrošnji memorije se dešava tokom pokretanja prvog skripta zbog inicijalizacije okruženja za rad Lua skript interpretera.

Posle toga svaka sledeća instanca skripta zauzima približno 500 kB RAM memorije. Ova vrednost zavisi najviše od dužine programskog koda skripta koji se pokreće.

5.5 Poređenje sa drugim rešenjima iz oblasti sistema za automatsku konfiguraciju uređaja

Ispitivanja pokazuju da implementirano rešenje obezbeđuje novi skup funkcionalnosti ne narušavajući drastično perfomanse sistema. U poređenju sa rešenjima navedenim u Poglavlju 2.4. omogućuje veći skup funkcionalnosti, međutim implementacija mehanizma za izvršavanje skriptova kao zasebne aplikacije unosi utrošak vremena koji nije prisutan u drugim rešenjima. Uporedni pregled funkcionalnosti između rešenja u *Insight* sistemu sa ugrađenim mehanizmom za izvršavanje konfiguracionih skriptova i rešenja realizovanog korišćenjem *OpenACS* prikazano je u Tabeli 13.

	Insight	***OpenACS***
Korišćeni skript jezik	Lua	JavaScript
Podrška za objektno orijentisani pristup u jeziku	Moguća uz modifikacije	Prisutna
Podrška za blokirajuće konfiguracione zahteve	Prisutna	Prisutna
Podrška za neblokirajuće konfiguracione zahteve	Prisutna	Nije prisutna
Podrška za pojedinčane uređaje	Prisutna	Prisutna
Podrška za više uređaja po nekom kriterijumu	Prisutna	Prisutna
Podrška za grupe uređaja	Prisutna	Nije prisutna
Ručno pokretanje skripta i zaustavljanje skriptova	Moguće	Nije moguće
Automatsko pokretanje skripta na događaj	Moguće	Moguće
Utrošak vremena unet podrškom za skriptove	Postoji	Ne postoji

Tabela 13. Uporedni pregled rešenja za izvršavanje konfiguracionih skriptova

Kao što se može videti iz Tabele 13. implementirano rešenje poseduje veći skup funkcionalnosti u odnosu na rešenje[9]. Poboljšanja se odnose na fleksibilniji sistem za izvršavanje skriptova u smislu mogućnosti pokretanja i izvršavanja skripta nad grupom uređaja, kao i funkcionalnosti za ručno pokretanje i zaustavljanje rada skripta. Pokretanje i zaustavljanje rada skripta može biti korisno kod ispitivanja i verifikacije rada novih uređaja koji se integrišu u sistem. Korišćenjem ovih funkcionalnosti se takođe može brzo ustanoviti nivo kompatibilnosti novog uređaja sa ACS-om u pogledu zadovoljenja pravila propisanih TR-069 protokolom. Nedostatak ovog rešenja u odnosu na rešenje sa kojim se poredi predstavlja utrošak vremena unet podrškom za skiptove, kao i podrška za objektno orijentisani pristup u Lua skript jeziku.

6. Zaključak

U radu je projektovan i implementiran mehanizam za izvršavanje konfiguracionih skriptova zasnovanih na TR-069 protokolu i skript jeziku Lua. Korišćenjem ovog mehanizma može se definisati proizvoljan broj konfiguracionih skriptova, zajedno sa pravilima za njihovu primenu tokom rada sistema. Korišćenje TR-069 protokola je omogućilo nezavisnost implementiranog mehanizma od vrste uređaja na koje se primenjuje, čime je ostvarena fleksibilnost rešenja. Rešenje je organizovano kroz proširenje postojećeg rešenja za ACS i implementaciju aplikacije za izvršavanje Lua skriptova.

Proširenja u postojećem ACS-u su projektovana tako da ne zahtevaju puno vremena za obradu i ne blokiraju izvršavanje postojećih funkcionalnosti u ACS-u. Sprezanje aplikacije za izvršavanje skriptova sa ACS-om korišćenjem HTTP protokola omogućava distribuciju obrade skriptova na više računara u slučaju potrebe za dugotrajnom obradom podataka u skriptovima.

Realizacija mehanizma za izvršavanje skriptova korišćenjem Lua skript jezika je predstavljala izazov, međutim Lua se pokazala kao dobar izbor između ostalih skript jezika. Nedostatak Lua-e svakako predstavlja nedostatak podrške za objektno orijentisano programiranje. Najveće prednosti su jednostavnost, brzina izvršavanja, proširivost i mogućnost izvršavanja na raznim platformama sa organičenim resursima. Zbog svojih perfomansi Lua je jedan od najkorišćenijih skript jezika za razvoj programske podrške gde je brzina izvršavanja bitan faktor, tu spadaju pre svega računarske igre[29].Zbog svojih osobina Lua je izuzetno korišćena za sisteme koji su u razvoju i koji se često proširuju, što svakako predstavljaju sistemi za automatsku konfiguraciju uređaja. Uzimajući u obzir nedostatke i prednosti Lua skript jezika, autor zaključuje da Lua predstavlja dobro rešenje za implementaciju mehanizma za izvršavanje skriptova u sistemima za automatsku konfiguraciju uređaja.

Zahtevi ovog programskog rešenja su ispunjeni i time su otvorene nove mogućnosti za dalja unapređenja i proširenja. Nedostatak implementiranog rešenja predstavlja utrošak vremena na HTTP komunikaciju između aplikacije i ACS-a.

Fokus daljeg rada će biti integrisanje dodatnih bezbednosnih mehanizama, obezbeđivanje sigurnosnih mehanizama od prevelikog zauzimanja računarskih resursa u slučaju skriptova sa dugotrajnom obradom podataka, kao i optimizacija HTTP komunikacije između poslužioca i aplikacije za izvršavanje Lua skriptova.

7. Literatura

[1] P. Horn "Autonomic Computing:IBM's Perspective on the State of Information Technology" , 2001.

[2] Kephart J.O. Chess D.M. "The vision of autonomic computing" , 2003.

[3] BenchmarkPortal, http://www.benchmarkportal.com/

[4] D. Menasce , Kephart J.O. "Guest Editors' introduction : Autonomic Computing" 2007.

[5] IBM, The autonomic computing edge: Can you CHOP up autonomic computing? 2008. http://www.ibm.com/developerworks/autonomic/library/ac-edge4/

[6] S. Pelley,T.F. Weinsch, "Storage management in the NVRAM era", 2013.

[7] J.M. Seigneur C.D.Jense "Towards Securitz Auto-configuration for Smart Appliances",2003.

[8] J FU, S Ramamoorthy "Smart translation of generic configurations",2003.

[9] Rachidi H, Karmouch A, "A framework for self-configuring devices using TR-069", University of Ottawa, 2011.

[10] Radovanovic S, Nemet N, "Cloud-based framework for QoS monitoring and provisioning in consumer devices", 2013.

[11] MZ Bjelica, I Papp, N Teslić "Diagnostic CWMP client for set-top box devices" , 2014.

[12] Cisco autoconfiguration network , https://learningnetwork.cisco.com/docs/DOC-9872

[13] Eun-Seo Lee, Hark-Jin Lee, Kwangil Lee, and Jun-Hee Park "Automating Configuration System and Protocol for Next-Generation Home Appliances" ,2013

[14] UpnP forum web stranica, http://www.upnp.org/

[15] SNMP protokol, RFC 1157, http://www.ietf.org/rfc/rfc1157.txt

[16] TR-069 protocol amadment 1, 2006. http://www.broadband-forum.org/technical/download/TR-069_Amendment-1.pdf

[17] TR-106 data model , http://www.broadband-forum.org/technical/download/TR-106_Amendment-2.pdf

[18] Motorola Edge ACS , http://www.motorolasolutions.com/web/Business/_Documents/static%20files/EDGE%20Data%20Sheet%20365-095-15888%20x.1.pdf?pLibItem=1

[19] OpenACS open-source project, http://sourceforge.net/projects/openacs/

[20] Transactions in EJB applications, http://docs.oracle.com/cd/E15051_01/wls/docs103/jta/trxejb.html

[21] EJB restrictions , http://www.oracle.com/technetwork/java/restrictions-142267.html

[22] LuaJ, http://luaj.org/luaj/README.html

[23] Gson, https://code.google.com/p/google-gson/

[24] Sun HTTP server , http://docs.oracle.com/javase/7/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/HttpServer.html

[25] JBoss official web page, http://www.jboss.org/

[26] Enterprise JavaBeans tehnology, http://www.oracle.com/technetwork/java/javaee/ejb/index.html

[27] PostgreSQL official page, http://www.postgresql.org/

[28] Net Time official page, http://technet.microsoft.com/en-us/library/bb490716.aspx

[29] Where Lua is used, https://sites.google.com/site/marbux/home/where-lua-is-used