УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

ЗБОРНИК РАДОВА
ФАКУЛТЕТА ТЕХНИЧКИХ НАУКА

Едиција: Техничке науке - зборници
Година: XXXV
Број: 11/2020

Нови Сад

Едиција: „Техничке науке – Зборници“
Година: XXXV *Свеска*: 11

Издавач: Факултет техничких наука Нови Сад
Главни и одговорни уредник: проф. др Раде Дорословачки, декан Факултета техничких Наука у Новом Саду

Штампа: ФТН – Графички центар ГРИД, Трг Доситеја Обрадовића 6, Нови Сад

CIP-Каталогизација у публикацији
Библиотека Матице српске, Нови Сад

378.9(497.113)(082)
62

ЗБОРНИК радова Факултета техничких наука / главни и одговорни уредник Раде Дорословачки. – Год. 7, бр. 9 (1974)-1990/1991, бр.21/22 ; Год. 23, бр 1 (2008)-. – Нови Сад : Факултет техничких наука, 1974-1991; 2008-. – илустр. ; 30 цм. –(Едиција: Техничке науке – зборници)

Месечно

ISSN 0350-428X

COBISS.SR-ID 58627591

ПРЕДГОВОР

Поштовани читаоци,

Пред вама је једанаеста овогодишња свеска часописа „Зборник радова Факултета техничких наука“.

Часопис је покренут давне 1960. године, одмах по оснивању Машинског факултета у Новом Саду, као „Зборник радова Машинског факултета“, а први број је одштампан 1965. године. Након осам публикованих бројева у шест година, пратећи прерастање Машинског факултета у Факултет техничких наука, часопис мења назив у „Зборник радова Факултета техничких наука“ и 1974. године излази као број 9 (VII година). У том периоду у часопису се објављују научни и стручни радови, резултати истраживања професора, сарадника и студената ФТН-а, али и аутора ван ФТН-а, тако да часопис постаје значајно место презентације најновијих научних резултата и достигнућа. Од броја 17 (1986. год.), часопис почиње да излази искључиво на енглеском језику и добија поднаслов «Publications of the School of Engineering». Једна од последица нарастања материјалних проблема и несрећних догађаја на нашим просторима јесте и привремени прекид континуитета објављивања часописа двобројем/двогодишњаком 21/22, 1990/1991. год.

Друштво у коме живимо базирано је на знању. Оно претпоставља реорганизацију наставног процеса и увођење читавог низа нових струка, као и квалитетну организацију научног рада. Значајне промене у структури високог образовања, везане за имплементацију Болоњске декларације, усвајање нове и активне улоге студената у процесу образовања и њихово све шире укључивање у стручне и истраживачке пројекте, као и покретање нових мастер и докторских студија, доносе потребу да ови, веома значајни и вредни резултати, постану доступни академској и широј јавности. Оживљавање „Зборника радова Факултета техничких наука“, као јединственог форума за презентацију научних и стручних достигнућа, пре свега студената, обезбеђује услове за доступност ових резултата.

Због тога је Наставно-научно веће ФТН-а одлучило да, од новембра 2008. год. у облику пилот пројекта, а од фебруара 2009. год. као сталну активност, уведе презентацију најважнијих резултата свих мастер радова студената ФТН-а у облику кратког рада у „Зборнику радова Факултета техничких наука“.

Поред студената мастер студија, часопис је отворен и за студенте докторских студија, као и за прилоге аутора са ФТН или ван ФТН-а.

Зборник излази у два облика – електронском на web сајту ФТН-а (www.ftn.uns.ac.rs) и штампаном, који је пред вама. Обе верзије публикују се сваки месец, у оквиру промоције дипломираних мастера.

У овом броју штампани су радови студената мастер студија, сада већ мастера, који су радове бранили у периоду од 22.07.2020. до 30.09.2020. год., а који се промовишу 19.12.2020. год. То су оригинални прилози студената са главним резултатима њихових мастер радова.

Известан број кандидата објавили су радове на некој од домаћих научних конференција или у неком од часописа. Њихови радови нису штампани у Зборнику радова.

Велик број дипломираних инжењера–мастера у овом периоду био је разлог што су радови поводом ове промоције подељени у две свеске.

У овој свесци, са редним бројем 11. објављени су радови из области:

- машинства и
- електротехнике и рачунарства.

У свесци са редним бројем 12. објављени су радови из области:

- грађевинарства,
- саобраћаја,
- графичког инжењерства и дизајна,
- архитектуре,
- инжењерског менаџмента,
- инжењерства заштите на раду и заштите животне средине,
- математике у техници,
- геодезије и геоматике,
- управљања ризиком од катастрофалних догађаја и пожара,
- инжењерства информационих система и
- анимације у инжењерству.

Уредништво се нада да ће и професори и сарадници ФТН-а и других институција наћи интерес да публикују своје резултате истраживања у облику регуларних радова у овом часопису. Ти радови ће бити објављивани на енглеском језику због пуне међународне видљивости и проходности презентованих резултата.

У плану је да часопис, својим редовним изласком и високим квалитетом, привуче пажњу и постане довољно препознатљив и цитиран да може да стане раме-уз-раме са водећим часописима и заслужи своје место на СЦИ листи, чиме ће значајно допринети да се оствари мото Факултета техничких наука:

„Високо место у друштву најбољих“

Уредништво

SADRŽAJ

STRANA

Radovi iz oblasti: Mašinstvo

Radovi iz oblasti: Elektrotehnika i računarstvo

STRANA

STRANA

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.791
DOI: https://doi.org/10.24867/10AM01Radovic

ELEKTROLUČNO ZAVARIVANJE U AKTIVNOM ZAŠTITNOM GASU PODRŽANO ULTRAZVUČNIM VIBRACIJAMA

ELECTRIC ARC IN SHIELDING GAS WELDING SUPPORTED BY ULTRASONIC VIBRATIONS

Damjan Radović, Sebastian Baloš, *Fakultet tehničkih nauka, Novi Sad*

Oblast – MAŠINSTVO

Kratak sadržaj – *U ovom radu izvršeno je eksperimentalno zavarivanje MAG postupkom zavarivanja podržano ultrazvukom. Uzorci zavareni različitim parametrima su ispitani metalografski i ispitivana je mikrotvrdoća. Pokazano je da ultrazvučna obrada utiče na eliminaciju stubaste strukture, karakteristične za elektrolučne postupke zavarivanja. Pored toga, javlja se smanjenje sadržaja Vidmanštetenovog i alotriomorfnog ferita, a povećanje sadržaja igličastog ferita. Kao posledica, javlja se povećanje mikrotvrdoće u metalu šava. Pored toga, smanjuje se razugljeničenje na površini, što sve ukazuje da je brzina kristalizacije uz prisustvo ultrazvuka povećana u odnosu na konvencionalno MAG zavarivanje.*

Ključne reči: *MAG zavarivanje, Ultrazvučna obrada, Mikrostruktura, Mikrotvrdoća*

Abstract – *In this work, experimental MAG welding supported by ultrasonic vibrations was studied. Metallographic examination and microhardness were done on the obtained specimens. It was shown that no columnar structure occurs, the typical feature of the arc welding weldments. Also, Widmanstaetten and allotriomorphic ferrite content is decreased, while acicular ferrite content increases. As a consequence, microhardness in the weld metal increases. Finally, decarburization on the surface is decreased, which all indicates an increased crystallization rate with ultrasonic vibrations applied, compared to conventional MAG welding.*

***Keywords*:** *MAG welding, Ultrasonic treatment, Microstructure, Microhardness*

1. UVOD

Zavarivanje omogućava postizanje materijalnog kontinuiteta između elemenata koji se zavaruju, pri čemu zavareni spoj treba da ima što veću sličnost osobina sa osobinama osnovnog materijala.

Spojevi nastali zavarivanjem spadaju u grupu neraskidivih spojeva čija se neraskidivost ostvaruje pomoću atomsko-molekularnih sila koje vladaju između elementarnih čestica zavarenih materijala [1].

Kod MAG (Metal Active Gas) postupka zavarivanja električni luk se uspostavlja između osnovnog materijala i žice za zavarivanje koja je ujedno i dodatni materijal. Najčešće se koristi jednosmerna struja obrnute polarnosti. Nakon uspostavljanja električnog luka sledi dodavanje žice za zavarivanje u električni luk, žica se topi i dolazi do formiranja zavarenog spoja [2]. U zavisnosti od vrste zaštitnog gasa elektrolučno zavarivanje topljivom elektrodom se skraćeno obeležava kao MAG (Metal Aktivni Gas) ili MIG (Metal Inertni Gas). Kod MAG postupka se kao zaštita koristi CO2 (ugljen-dioksid) ili mešavina gasova koja se ponaša kao aktivni gas, a kod MIG postupka Ar, He (argon, helijum) ili mešavina gasova koja se ponaša kao inertni gas [3, 4].

Jedan od aktuelnih pravaca razvoja je vibraciona obrada, koja se može primenjivati u toku zavarivanja i nakon zavarivanja [5]. Vibraciona obrada u toku postupka zavarivanja se primenjuje sa ciljem poboljšanja mehaničkih osobina zavarenog spoja. Brojnim eksperimentima se pokušava formirati baza znanja o prednostima i nedostacima primene vibracija u toku zavarivanja, sa ciljem da se izbegnu greške zavarivanja [6].

2. EKSPERIMENTALNI RAD

Tokom eksperimentalnog istraživanja korišten je konstrukcioni čelik S275JR debljine 3 mm, aparat za zavarivanje DynaMig 450 sa dodatni materijalom u vidu pune žice SG3 prečnika 1 mm, zaštitni gas C18 (mešavina 82 % argona i ugljen-dioksida 18 %) i ultrazvučni generator MPI2000.

Tabela 1. *Hemijski sastav osnovnog materijala (%)*

C	**Si**	**Mn**	**S**	**Cr**	**P**
0,11	0,16	0,52	0,039	<0,002	0,011
Al	**Cu**	**Mo**	**Ni**	**Ti**	**Fe**
0,006	0,35	0,012	0,083	<0,002	ostatak

Na radnu ploču koja prenosi vibracije su postavljena dva lima dimenzija 150x40x3mm, koja su pričvršćena pomoću vijaka. Limovi su postavljeni sučeono i koristi se I oblik žljeba dok se rastojanje između limova menjalo tokom eksperimenta. Tokom prethodnih eksperimenata došlo se do zaključka da je najpovoljniji tehnološki postupak u kojem se nakon uspostavljanja električnog luka uključuje vibracioni uređaj i pre prekidanja električnog luka se gasi vibracioni uređaj. Odnosno, uspostavljanje i prekidanje električnog luka nije pod uticajem vibracija dok se formiranje šava između ova dva procesa odvija pod uticajem vibracija. U tabeli 2 nalaze se tehnološki parametri koji su primenjeni prilikom formiranja uzoraka.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Sebastian Baloš, vanr. prof.

Iz zavarenih ploča su isečeni gore navedeni uzorci za makroskopska i mikroskopska ispitivanja. Uzorci za makro i mikro ispitivanja 1, 2 i 3 su izabrani tako da prikazuju najčešće greške koje su primećene tokom eksperimenta, dok su uzorci 0 i 4 dobijeni bez grešaka.

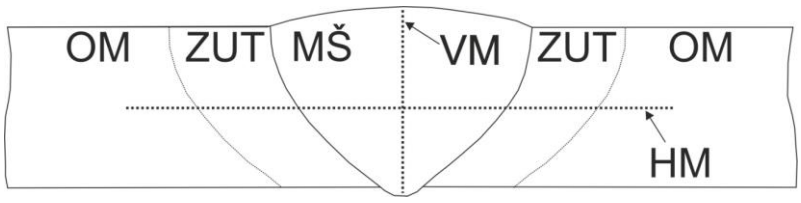
Tabela 2. *Tehnološki parametri spajanja uzoraka*

Uzorak 0	Ultrazvučni uređaj	Položaj	Frekvencija [Hz]	Jačina struje [A]	Snaga [kW]
		-	-	-	-
	Aparat za zavarivanje	Napon luka [V]	Jačina struje[A]	Brzina dodavanja žice [m/min]	Zazor [mm]
		18	120	4,5	1,5
Uzorak 1	Ultrazvučni uređaj	Položaj	Frekvencija [Hz]	Jačina struje [A]	Snaga [kW]
		vertikalni	20 060	2,2	1,2
	Aparat za zavarivanje	Napon luka [V]	Jačina struje[A]	Brzina dodavanja žice [m/min]	Zazor [mm]
		18	120	4,5	1,5
Uzorak 2	Ultrazvučni uređaj	Položaj	Frekvencija [Hz]	Jačina struje [A]	Snaga [kW]
		vetrikalni	20 060	2,2	1,2
	Aparat za zavarivanje	Napon luka [V]	Jačina struje[A]	Brzina dodavanja žice [m/min]	Zazor [mm]
		17,5	100	4,5	1,5
Uzorak 3	Ultrazvučni uređaj	Položaj	Frekvencija [Hz]	Jačina struje [A]	Snaga [kW]
		Vertikalni	20 060	2,2	1,2
	Aparat za zavarivanje	Napon luka [V]	Jačina struje[A]	Brzina dodavanja žice [m/min]	Zazor [mm]
		17,5	100	4,5	1,2
Uzorak 4	Ultrazvučni uređaj	Položaj	Frekvencija [Hz]	Jačina struje [A]	Snaga [kW]
		horizontalni	20 082	0,8	0,5
	Aparat za zavarivanje	Napon luka [V]	Jačina struje[A]	Brzina dodavanja žice [m/min]	Zazor [mm]
		18	120	4,5	1,2

Ispitivanje mikrostrukture izvršeno je na svetlosnom mikroskopu Leitz Orthoplan, a analizirane su mikrostruktura osnovnog materijala, ZUT-a i mikrostruktura metala šava.

Izvršena je standardna priprema uzoraka za metalografska ispitivanja na opremi Struers: isecanje, zatapanje uzoraka u autopolimerizirajući akrilat, brušenje vodobrusnim papirima granulacije P150-P2500, poliranje uzoraka tkaninama za poliranje dijamantskim suspenzijama 6 μm, 3 μm i 1 μm i nagrizanje Nitalom (3% HNO_3 i 97% etanol).

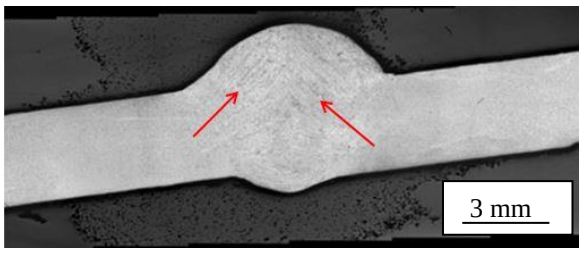
Ispitivanje mikrotvrdoće izvršeno je na poprečnom preseku zavarenog spoja po Vikersu pri opterećenju od 0.2 kg na uzorcima za analizu mikrostrukture po završetku mikrostrukturne analize, prema šemi 1, sa rastojanjem između otisaka od 0,15 mm.



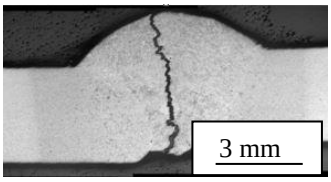
Slika 1. *Merenje tvrdoće u horizontalnom pravcu (HM) i vertikalnom pravcu (VM)*

3. REZULTATI I DISKUSIJA

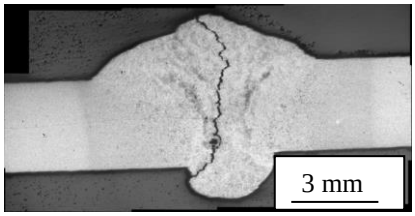
Makro prikaz zavarenih spojeva dat je na slikama 2-6. Na slici 2 je prikazan nulti uzorak na kojem se primećuju stubasti kristali, koji su rasli od graničnih kristala osnovnog materijala i sučeljavaju se na sredini spoja tj. primetna je usmerena stubičasta struktura. Ovaj uzorak nije nastao pod uticajem ultrazvučne obrade tako da se koristi kao referntni uzorak. Na makroskopskim prikazima uzoraka 1, 2 i 3 su vidljive prsline i nepravilnosti oblika šava. Na slici 6, gde je prikazan uzorak 4, se ne primećuju predhodno navedene greške. U odnosu na nulti uzorak uzorci zavareni pod uticajem ultrazvučne obrade nemaju izraženu stubastu mikrostrukturu.



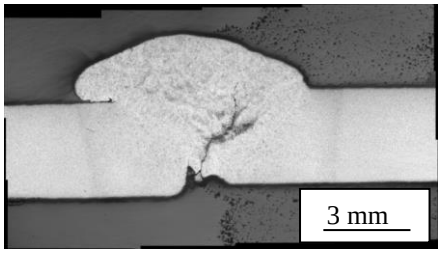
Slika 2. *Makroskopski prikaz nultog uzorka*



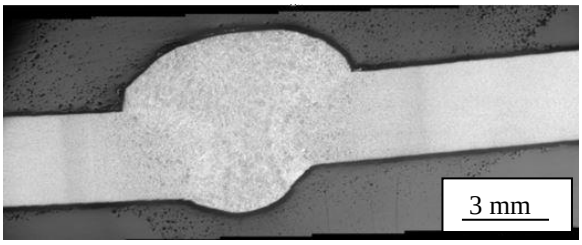
Slika 3. *Makroskopski prikaz prvog uzorka*



Slika 4. *Makroskopski prikaz drugog uzorka*



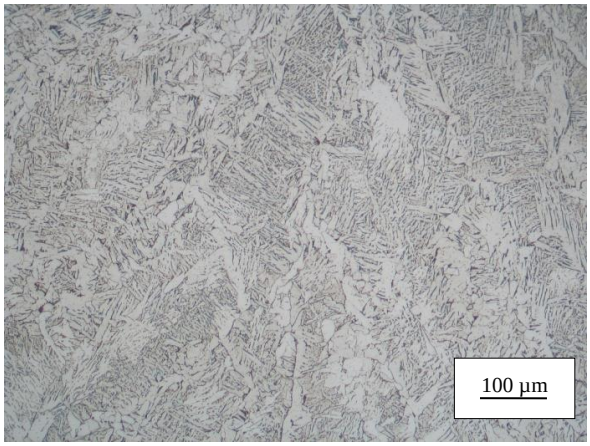
Slika 5. *Makroskopski prikaz trećeg uzorka*



Slika 6. *Makroskopski prikaz četvrtog uzorka*

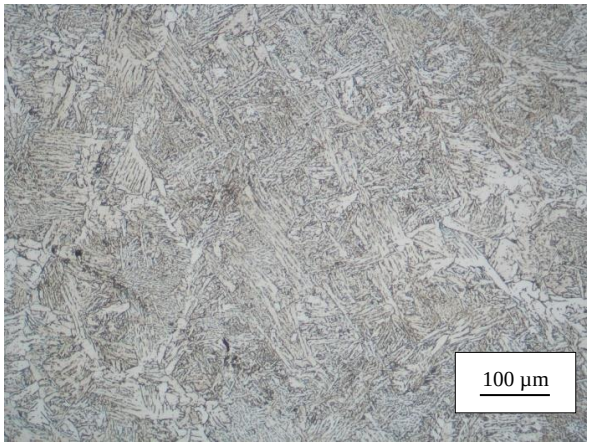
Na slikama 7 i 8 prikazana je mikrostruktura središnjeg dela metala šava nultog i četvrtog uzorka. Na nultom uzorku koji je prikazan na slici 10 primetan je igličasti ferit, Vidmanštetenov ferit i alotriomorfni ferit. Na ovoj slici se takođe primećuje stubičasta struktura kristala. Na uzorcima nastalim pod uticajem ultrazvučne obrade prisutan je acikularni igličasti ferit i znatno manje količine alotriomorfnog i Vidmanštetenovog ferita. Takođe na ovim uzorcima se ne primećuje usmerenost kristala kao na nultom uzorku.

Rezultati mikrotvrdoće su prikazani na dijagramima prikazanim na slikama 9 i 10. Vidi se da su vrednosti mikrotvrdoće veći kod uzorka 4 u odnosu na uzorak 0.

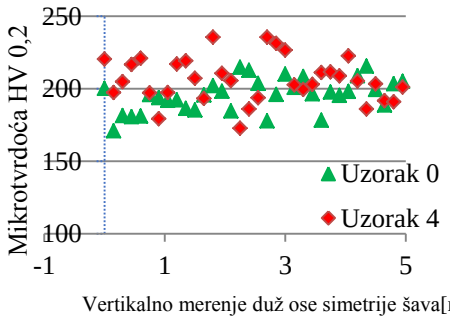


Slika 7. *Mikrostruktura središnjeg dela metala šava nultog uzorka*

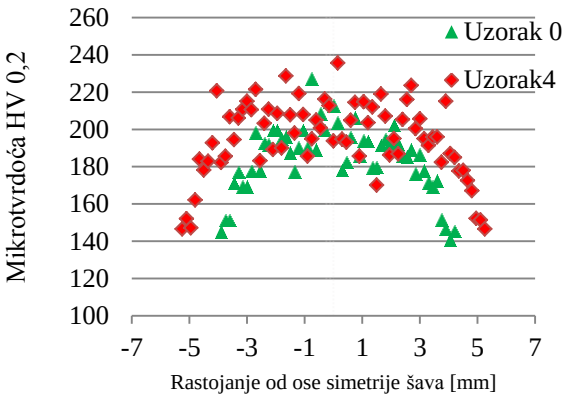
Povećanje mikrovrdoće je direktno povezano sa promenom mikrostrukture. Naime, povećanje sadržaja igličastog ferita na račun smanjenja Vidmanštetenovog i alotriomorfnog ferita kod uzorka 0 ima značajan uticaj na povećanje mikrotvrdoće, s obzirom na znatno manju veličinu zrna kod igličastog ferita. Pored toga, može se konstatovati da je dejstvo ultrazvuka povezano i sa eliminacijom usmerenosti kristala, verovatno kroz mehanizam povećanja broja centara kristalizacije, što je u skladu sa [7].



Slika 8. *Mikrostruktura središnjeg dela metala šava četvrtog uzorka*



Slika 9. *Poređenje vertikalnog merenja mikrotvrdoće uzorka 0 i uzorka 4*



Slika 10. *Poređenje horizontalnog merenja mikrotvrdoće uzorka 0 i uzorka 4*

4. ZAKLJUČAK

Na osnovu provodenog eksperimenta mogu se izvući određeni zaključci.

- Najveći uticaj na kvalitet zavara ima snaga ultrazvučnih vibracija i položaj sonotrode u odnosu na predmet obrade.
- Usled dejstva ultrazvuka, stubasta struktura se ne pojavljuje, smanjuje se sadržaj Vidmanštetenovog i alotriomorfnog ferita, a povećava se sadržaj igličastov ferita
- Tvrdoća metala šava se povećava usled dejstva ultrazvuka tokom MAG zavarivanja, što je posledica šovećanog sadržaja igličastog ferita u metalu šava. Povećana tvrdoća je indikator veće čvrstoće metala šava.

Na osnovu prethodnog, može da se generalno zaključi da je ultrazučni tretman tokom MAG zavarivanja koristan sa gledišta povećanja mehaničkih osobina, kako MAG, tako verovatno i MIG i TIG zavarivanja, sa mogućnošću upotrebe i na druge materijale, kao što su nerđajući čelici i legure obojenih metala, kao što su legure aluminijuma.

5. LITERATURA

[1] A. Blagojević, O. Pašić, *"Zavarivanje lemljenje lijepljenje",* Mostar, Mašinski Fakultet Mostar, Mašinski Fakultet Banjaluka, 1991.

[2] N. Atlija, "M*AG postupak zavarivanj*", Rijaka, Sveučilište u Rijeci, 2016.

[3] M. Klobučar, "*Teorijske i praktične osnove MIG/MAG postupka zavarivana*", Varaždin, Sveučilište Sjever Varaždin, 2016.

[4]D. Seferijan, (preveo: R. Arsenijević), "*Metalurgija Zavarivanja*", Građevinska knjiga Beograd. Beograd. 1969.

[5] B. Sabo, "*Prilog istraživanju uticaja vibracione obrade na zaostale deformacije i napone zavarenih nosača*", Novi Sad, Fakultet tehničkih nauka u Novom Sadu, 1997.

[6] M.J.Jose, S.S.Kumar, A.Sharma, "*Vibration assisted welding processes and their influence on quality of welds*", Science and technology of welding Joining. Volume 21. 2016. pages: 243-258

[7]S. Kumar , C.S. Wu, G.K. Padhy, W. Ding, "*Application of ultrasonic vibrations in welding and metal processing*", Journal of Manufacturing Processes. Volume 26. 2017. pages: 295-322

Kratka biografija:

Damjan Radović rođen je u Trebinju 1994. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Mašinstvo – Matreijali i tehnologije spajanja odbranilo je 2017. god. Kontakt: damjan.radovic.994@gmail.com

Sebastian Baloš rođen je u Somboru 1974. god. Doktorirao je na Fakultetu tehničkih nauka 2010. godine, a 2011. godine je biran u zvanje docenta. Od 2016. godine, u zvanju je vanrednog profesora u užoj naučnoj oblasti Materijali i tehnologije spajanja.

Зборник радова Факултета техничких наука, Нови Сад

УДК: 621
ДОИ: https://doi.org/10.24867/10AM02Katona

ПРОЈЕКАТ УРЕЂАЈА ЗА ИСПИТИВАЊЕ ТРАНСПОРТНИХ ТРАКА

DESIGN OF THE DEVICE FOR TESTING CONVEYOR BELTS

Мирко Катона, Никола Иланковић, Драган Живанић, *Факултет техничких наука, Нови Сад*

Област – МАШИНСТВО

Кратак садржај – *Тема рада јесте пројектовање уређаја за испитивање транспортних трака. На почетку су извршени прегледи стандарда који се односе на испитивање транспортних трака и научних радова чија је тема испитивање транспортних трака. Након тога је направљен концепт уређаја. Затим је извршен избор компонената уређаја које учествују у погону и управљању. Након тога је изведен прорачун елемената и анализа помоћу методе коначних елемената. На крају су дате смернице за безбедност и заштиту при раду.*

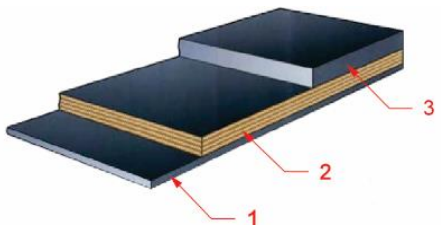
Кључне речи: *транспортна трака, узорак транспортне траке, уређај за испитивање транспортних трака.*

Abstract – *The subject of this paper is the design the testing device for conveyor belts. An overview of standards and scientific papers concerning testing devices for conveyor belts was given in the beginning. Then, the components of testing device were selected. The second part gives the calculation of the elements of testing device and their analysis with the finite element method. Finally, recommendations concerning health and safety measures were given.*

Keywords: *conveyor belt, conveyor belt sample, testing device for conveyor belts.*

1. УВОД

Трака је основни и најодговорнији елемент тракастог транспортера, а истовремено је и носећи и вучни елемент. Највише се примењује трака са текстилним влакнима као носећим слојевима који су обложени гумом као што је приказано на сл. 1. Носећи слојеви преносе силе које се јављају при кретању, а гумена облога служи као заштита од механичких оштећења и хабања [1].



Сл. 1. Транспортна трака: 1-облога са клизне стране; 2-носећа влакна; 3-облога са носеће стране

Уређајем за испитивање транспортних трака је могуће извршити четири врсте испитивања:

- Одређивање затезне чврстоће траке,
- Одређивање еластичне и трајне деформације траке;
- Одређивање отпорности према раздвајању слојева траке;
- Одређивање максималне силе коју може да издржи веза спајања траке.

2. ПРЕГЛЕД СТАНДАРДА

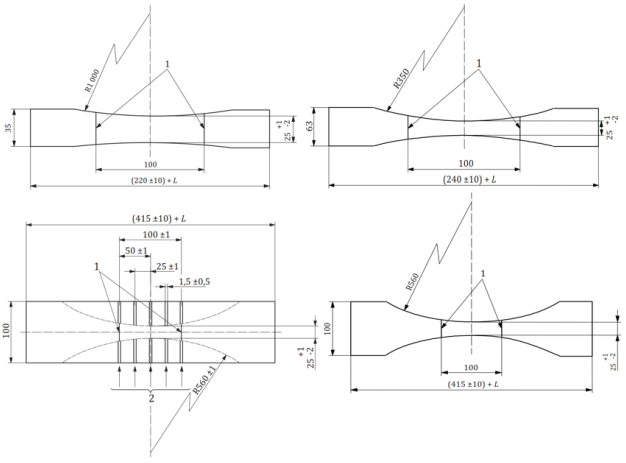
Да би се извршила набројана испитивања потребно је познавање стандарда према којима се врше.

Прегледани су следећи стандарди: DIN 22102, ISO 282, ISO 283, ISO 9856, ISO 252, ISO 1120, ISO 18573 и ISO 7500. При прегледу стандарда ISO 282 [2], утврђен је број шема узорака према табели 1.

Табела 1. Број шема узорака транспортне траке у односу на дужину траке.

Укупна дужина транспортне траке L [m]	Број шема узорака
L ≤ 500	1
500 < L ≤ 1 000	2
1 000 < L ≤ 2 000	3
2 000 < L ≤ 3 500	4
3 500 < L ≤ 5 000	5
5 000 < L ≤ 7 000	6
7 000 < L ≤ 10 000	7
Напомена: Преко 10 000 m, по једну шему узорака треба направити за сваких 5 000 m дужине траке.	

Стандард ISO 283 [3] предвиђа да је из сваке шеме узорака потребно направити по три испитна узорка чије су димензије приказане на сл. 2.



Сл. 2. Узорци транспортне траке према стандарду ISO 283

НАПОМЕНА:
Овај рад проистекао је из мастер рада чије ментор је био др Драган Живанић, ванр. проф.

Такође, наведени стандард предвиђа и дужину од 50 mm којом чељуст уређаја захвата узорак траке.
Стандард ISO 18573 [4] предвиђа атмосферске услове током испитивања. Када је у питању само контрола температуре тада је потребна температура од 23 °C ± 2 °C или 20 °C ± 2 °C, која мора бити постигнута најмање 3 сата пре испитивања.

3. ПРЕГЛЕД ДОСАДАШЊИХ ИСТРАЖИВАЊА

Најзначајније досадашње истраживање је: Издужење текстилних носећих слојева транспортне траке – Анализа оштећења, чији су аутори: *G. Fedorko, V. Molnár, P. Michalik, M. Dovica, T. Tóth, T. Kelemenová.* Из њиховог истраживања добијена је идеја за конструкцију чељусти уређаја. Такође, веома је значајно једно од њихових испитивања узорка траке чији су параметри дати у табели 2.

Табела 2. *Карактеристике траке из које је исечен узорак*

Материјал траке	EP
Назривна затезна чврстоћа	125 N/mm
Дебљина траке	8 mm
Дебљина носеће облоге	4 mm
Дебљина клизне облоге	1,5 mm
Број носећих слојева	3
Специјална особина траке	A

Резултати испитивања приказју да је до кидања узорка дошло при дејству силе од 3473 N. Напон који се том приликом јавља у узорку износи:

$$\frac{F}{S_4 \cdot L} = \frac{3457}{2,5 \cdot 25} = 55 \ \frac{\text{N}}{\text{mm}^2} = 55 \text{ MPa}$$

где су:

- F – Сила при којој долази до кидања узорка;
- S_4 - Укупна дебљина носећих слојева траке;
- L – Ширина узорка траке између реф. линија.

4. КОНЦЕПТ УРЕЂАЈА ЗА ИСПИТИВАЊЕ ТРАНСПОРТНИХ ТРАКА

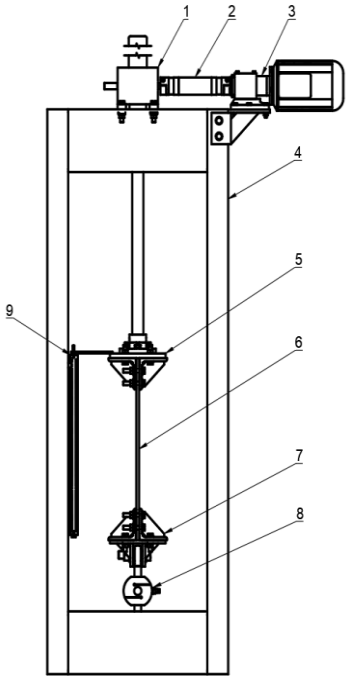
Уређај за испитивање транспортних трака треба да има могућност прихватања узорка транспортне траке помоћу чељусти које треба да прихватају узорке ширине до 100 mm, дужине 415±10 mm и дебљине до 30 mm.
Чељусти треба да имају ход 130 mm који ће омогућити кидање узорка свих димензија.
Уређај треба да затеже узорак константном брзином од 100±10 mm/min, до појаве кидања узорка, а такође треба да постоји могућност оптерећења узорка одређеном силом током дужег временског периода.
Да би дошло до кидања траке назривне затезне чврстоће 2000 N/mm потребно је да уређај на узорак траке делује силом:

$$F = 2000 \ \frac{\text{N}}{\text{mm}} \cdot 25 \text{ mm} = 50000 \text{ N}$$

где је:

- 2000 N/mm – назривна затезна чврстоћа (назривна јачина на кидање) траке према стандарду ISO 22102;
- 25 mm – ширина узорка траке између референтних линија.

На сл. 3. приказана је скица уређаја за испитивање транспортних трака.



Сл. 3. Скица уређаја за исшитивање транспортних трака

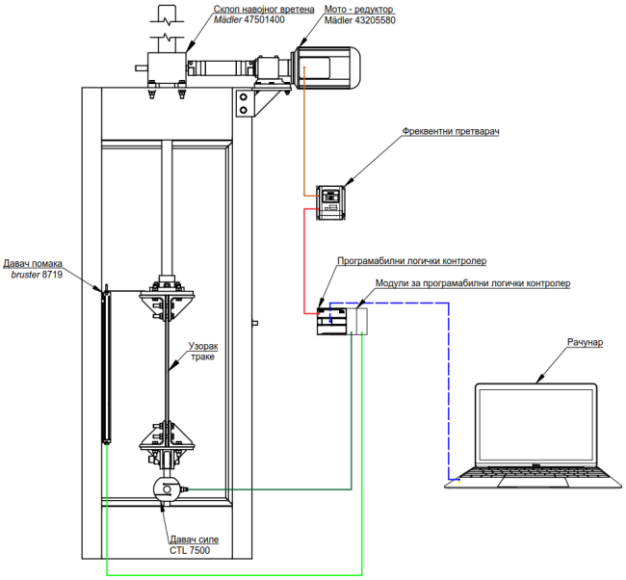
У таб. 3. приказане су компоненте означене на сл. 3.
Табела 3. Компоненте уређаја за испитивање транспортних трака

Ознака	Компонента
1	Склоп навојног вретена
2	Спојница - вратило
3	Мото - редуктор
4	Носећа конструкција
5	Горња чељуст кидалице
6	Узорак траке
7	Доња чељуст кидалице
8	Давач силе
9	Давач помака

5. ИЗБОР КОМПОНЕНТИ

У оквиру овог поглавља извршен је избор компонената које учествују у погону и управљању уређајем.

На сл. 4. приказано је повезивање компонената за управљање уређајем.



Сл. 4. Повезивање компонената за управљање уређајем

Изабране су следеће компоненте:

- Навојно вретено типа B *Mädler* 475 014 00;
- Мото-редуктор *Mädler* 432 055 80;
- Вратило-спојница *Mädler RNW* 28;
- Давач силе *Laumas CTL* 7500;
- Давач помака *burster* 8719-5500;
- Програмабилни л. контролер *Fatek FBs*–32*MA*;
- Фреквентни регулатор *Hitachi NES1*–007 *HBE*.

У табели 4. приказане су карактеристике изабраних компоненти.

Табела 4. Карактеристике изабраних компоненти

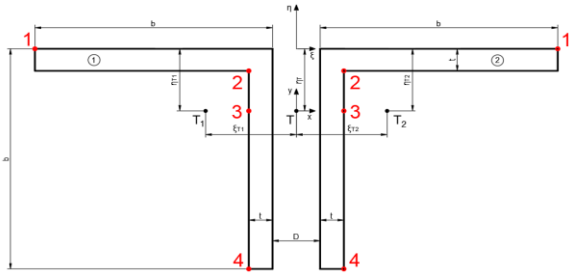
Name	Product Number	Description
Worm Gear Screw Jack NP/I; Version B	47501400	- Size: 4; - Spindle length: 1700 mm; - Trapezoidal Spindle: Tr. 40x7; - max. Stroke Force: 50 000 N; - Material: Steel C45; - Without M30 thread at the end of the spindle and without M30 Flange plate for this version; - Spindle and safety features; - Fastening Strip Set: size 4; - 2 End Switchers.
Helical Geared Motor HR/I	43205580	- P = 0.55kW; n_2 = 107 min^{-1}; M_a = 47 Nm; - Gearbox size: 30/2; Motor size 80A4; - With break, posistor and encoder if it is posible.
Connecting Shafts RNW 28	RNW 28	- RNW 28; - Length LA = 255 mm; - Boreholes d = 20 mm on both sides; - Speed: 107 min^{-1}.
Tension (compression) load cell	CTL7500	- Load capacity: 7500 kg.
Potentiometric Displacement Sensor	8719-5500	- Measuring Range: 500 mm.
PLC Fatek	FBs–32MA	- 20 points 24VDC digital input (up to 10KHz in 4 points) - 12 Points relay digital output - 1 built-in RS-232 (can be expanded up to 3) - 100~240VAC power supply
Frequency Inverter NES1-007HBE	46013075	- Number of Phases: 3; - Output Power: 0,75 kW.

6. ПРОРАЧУН УРЕЂАЈА ЗА ИСПИТИВАЊЕ ТРАНСПОРТНИХ ТРАКА

Током прорачуна извршени су докази чврстоће, прорачуни заварених и завртањских веза.

Најзначајнији прорачун јесте доказ чврстоће чељусти уређаја јер је чељуст најодговорнији део.

На сл. 5. приказана је скица L профила чељусти са обележеним тачкама у којима се рачуна напон.



Сл. 5. Скица L профила чељусти

Резултати прорачуна приказани су у табели 4.

Табела 4. *Вредности напона који се јављају у чељустима*

	σ [kN/cm^2]	τ [kN/cm^2]	σ_u [kN/cm^2]
Тачка 1	2.00	0.00	2.00
Тачка 2	1.30	1.70	3.22
Тачка 3	0.00	1.82	3.15
Тачка 4	5.07	0.00	5.07

Максимални упоредни напон јавља се у тачки 4 и износи: $\sigma_{u,4} = 5{,}07\ \frac{\text{kN}}{\text{cm}^2}$

Допуштени напон за први случај оптерећења за челик S355 од којег су L профили чељусти направљени износи: $\sigma_{dop} = 23{,}67\ \frac{\text{kN}}{\text{cm}^2}$ [5]

$$\sigma_{u,4} = 5{,}07\ \frac{\text{kN}}{\text{cm}^2} < \sigma_{dop} = 23{,}67\ \frac{\text{kN}}{\text{cm}^2}$$

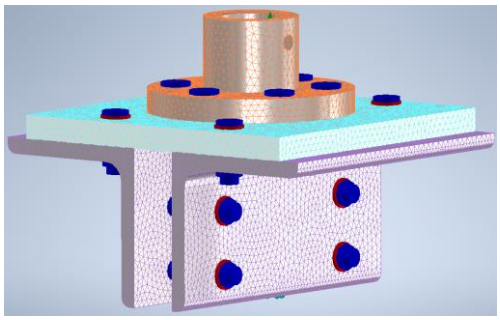
Услов доказа чврстоће је задовољен.

7. АНАЛИЗА МЕТОДОМ КОНАЧНИХ ЕЛЕМЕНАТА

За проверу напона који се приликом дејства оптерећења јавља у елементима уређаја користи се метода коначних елемената.

Софтвер коришћен за анализу је *Autodesk Nastran* који је саставни део *Autodesk Inventor-a*.

Прва анализа урађена је без укрућења (сл. 6).

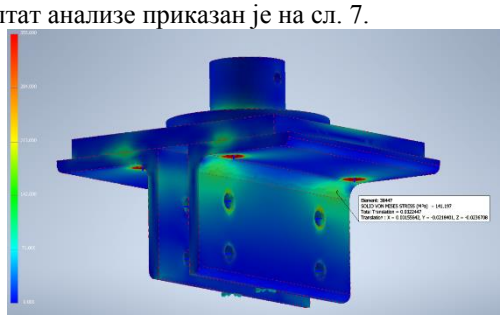


Сл. 6. Чељуст без укрућења

Ослонци су постављени на доњу површину L профила. Елементима је на овај начин онемогућено померање и ротација.

Оптерећење је у виду силе која је постављена на унутрашњој површини навртке за навојно вретено. Сила делује према горе и износи 50 kN.

Резултат анализе приказан је на сл. 7.

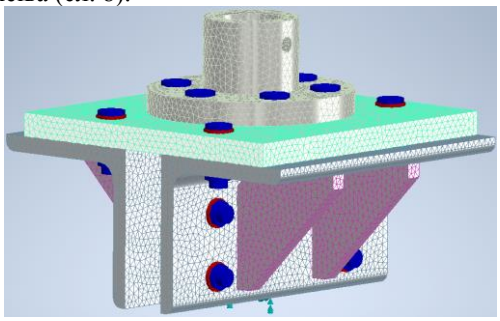


Сл. 7. Анализа чељусти без укрућења

Анализом су добијени следећи резултати:

- максимални напон у чељустима је 14,1 kN/cm^2 односно 141,2 MPa.

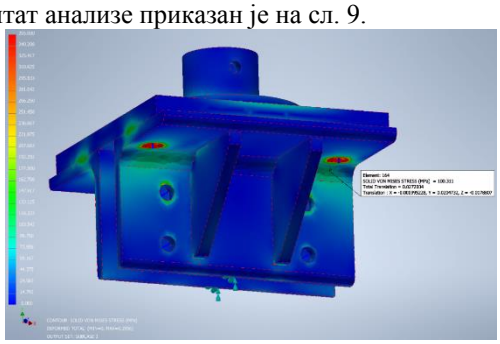
Да би се вредност напона смањила постављена су укрућења (сл. 8).



Сл. 8. Чељуст са укрућењима

Ослонци и оптерећење идентични су као и у анализи без укрућења.

Резултат анализе приказан је на сл. 9.



Сл. 9. Анализа чељусти са укрућењима

Анализом су добијени следећи резултати:

- максимални напон у чељустима је 10,03 kN/cm^2 односно 100,3 MPa.

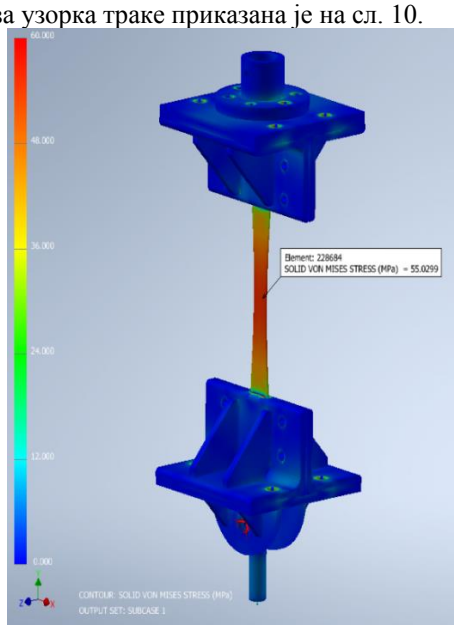
Коришћењем укрућења вредност напона смањена је за 4 kN/cm^2 односно 40 MPa.

Вредност напона добијена анализом одступа од вредности добијеном доказом чврстоће због појаве концентрације напона у појединим тачкама које у реалности не представљају проблем. Међутим у оба случаја вредности напона не прелазе дозвољену вредност напона која за материјал S355 износи 23,67 kN/cm^2 односно 236,7 MPa.

Помоћу методе коначних елемената извршена је анализа узорка траке. Узорак траке је типа А према стандарду ISO 283,

Параметри за вршење анализе подешени су према параметрима узорка траке (табела 2) коришћеном у научном раду ,,Анализа оштећења узорака транспортних трака при дејству затезног оптерећења'' Сила која делује на узорак траке је $F = 3457$ N током чијег дејства је у испитивању дошло до кидања траке.

Анализа узорка траке приказана је на сл. 10.



Сл. 10. Анализа узорка траке

Анализом су добијени следећи резултати:

- максимални напон у узорку траке је 5,503 kN/cm^2 односно 55,03 MPa.

Анализа приказује да се у узорку јавља напон који је једнак затезној чврстоћи током које је дошло до кидања узорка траке.

8. БЕЗБЕДНОСТ И ЗАШТИТА НА РАДУ СА УРЕЂАЈЕМ ЗА ИСПИТИВАЊЕ ТРАНСПОРТНИХ ТРАКА

На уређају су са обе стране постављена врата која служе за заштиту. При раду уређајем препоручује се да уређај користи један оператер. Оператер и особље који се налазе у близини су обавези да носе заштитне наочаре и заштитне ципеле. Такође, да поседују и флуоресцентни прслук да би били уочљивији у случају да се у лабораторији врше нека друга испитивања или се врши транспорт робе. Оператер је у обавези да носи и заштитне рукавице док поставља узорак у чељуст уређаја.

9. ЗАКЉУЧАК

При транспорту појединих материјала, током времена долази до промене количине влаге што доводи до промене радног оптерећења. Тада у транспортној траци долази до појаве силе која је већа од силе за коју је трака предвиђена. Дејство такве силе доводи до неправилног рада транспортера, а у екстемнијим случајевима може довести и до квара одређених елемената тракастог транспортера. Помоћу пројектованог уређаја могуће је симулирати дејство веће силе од силе добијене прорачуном и на тај начин доћи до закључака какве последице то оставља на траци.

10. ЛИТЕРАТУРА

[1] Д. Живанић: Непрекидни и аутоматизовани транспорт - скрипта, Факултет техничких наука, Нови Сад, 2019.

[2] Стандард ISO 282.

[3] Стандард ISO 283.

[4] Стандард ISO 18573

[5] З. Петковић, Д. Острић: Металне конструкције у машиноградњи 1, Машински факултет, Београд, 1996.

Кратка биографија:

Мирко Катона рођен у Руском Крстуру 1995. год. Дипломирао 2019. године на Факултету техничких наука, смер Механизација и конструкционо машинство, на којем исте године уписује мастер студије, смер Машинске конструкције, транспортни системи и логистика.

Никола Иланковић рођен у Суботици 1994. год. Завршио мастер академске студије 2018. године на Факултету техничких наука и исте године уписао докторске студије машинства. Запослен је у звању асистент мастер на Факултету техничких наука у Новом Саду.

Драган Живанић рођен у Сремској Митровици 1972. год. Докторирао је 2012. год. Од 2019. год. ради као ванредни професор на Факултету техничких наука у Новом Саду.

Зборник радова Факултета техничких наука, Нови Сад

УДК: 62
ДОИ: https://doi.org/10.24867/10AM03Bodrozic

АУТОМАТИЗОВАНО ПРОЈЕКТОВАЊЕ ТРАКАСТИХ ТРАНСПОРТЕРА СА НОСЕЋОМ КОНСТРУКЦИЈОМ ОД АЛУМИНИЈУМСКИХ ПРОФИЛА

AUTOMATED DESIGN OF BELT CONVEYORS WITH SUPPORTING STRUCTURE OF ALUMINUM PROFILES

Никола Бодрожић, Драган Живанић, Никола Иланковић, *Факултет техничких наука, Нови Сад*

Област – МАШИНСТВО

Кратак садржај – *Тема рада јесте аутоматизовано пројектовање тракастог транспортера са носећом конструкцијом од алуминијумских конструкционих профила.. На почетку су дате теоријске поставке у вези са тракастим транспортерима, обрађени су најбитнији подсклопови и дат је преглед појединих изведених решења. Након тога је дат детаљан прорачун носеће конструкције тракастог транспортера, као и прорачун потребне снаге погонског електромотора за погон транспортне траке. На крају је објашњен начин моделирања и аутоматизованог пројектовања.*

Кључне речи: *тракасти транспортер, прорачун носеће конструкције, 3D моделирање*

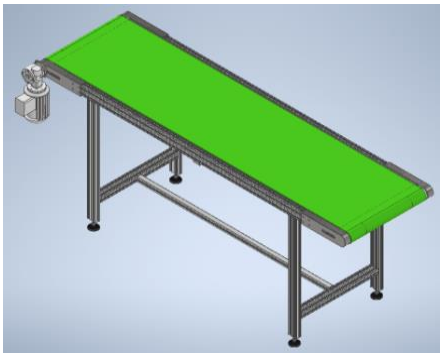
Abstract – *The subject of this paper is the automated design of the belt conveyor with supporting structure of aluminum profiles. First, theoretical basis about the belt conveyor was given. The most important elements of the conveyor were described and analyzed. The overview of existing solutions was also given. After, a detailed calculation of the conveyor construction was given and the calculation of the required drive power was presented. At the end it was described how the 3D design process was conducted.*

Keywords: *belt conveyor, calculation of the load-bearing structure, 3D design*

1. УВОД

Тракасти транспортери са носећом конструкцијом од *Alu* профила („лаки“ транспортери) су уређаји непрекидног транспорта који се најчешће користе за транспорт комадног терета или упакованих кутија у којима се налази одређен број транспортних јединица, а ретко расутог терета. Терет се најчешће транспортује у хоризонталном и мало нагнутом правцу помоћу гумене транспортне траке која представља носећи и вучни елемент. Прилагодљиви су широком спектру примене, од лаких специфичних фармацеутских производа, преко кутија и пакета у индустријама хране и пића. Такође, налазе примену у производним и процесним индустријама, аутоматизованим системима, итд.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чије ментор је био др Драган Живанић, ванр. проф.



Слика 1. *Приказ тракастог транспортера са носећом конструкцијом од Alu профила*

Основне предности оваквих транспортера су релативно мали отпори кретања, односно мала потрошња енергије, лака и једноставна конструкција и као таква заузима мало простора, висока исплативост, лако одржавање и мали експлоатациони трошкови велика поузданост у раду, савремен дизајн и бешуман рад.

Такође, поседује и недостатке као што су осетљивост на ударе у конструкцију, непогодност формирања сложених траса у хоризонталној равни као и у простору, ограничен нагиб транспортовања са стандардним тракама, осетљивост на хемијске утицаје као и на повишене температуре.

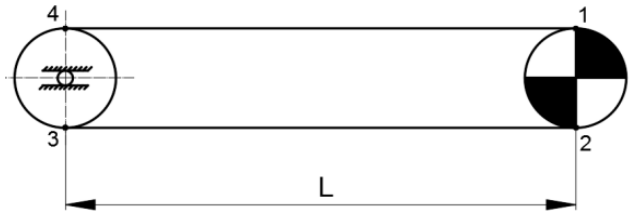
2. ОПИС ПРОБЛЕМАТИКЕ

Према пројектном задатку, за транспорт комадног терета, у аутомобилској индустрији предвиђен је тракасти транспортер са конструкцијом од алуминијумских профила. За транспортер је било потребно прорачунати и извршити избор његових карактеристичних елемената, прорачунати носивост конструкције и извршити аутоматизовано пројектовање транспортера у *3D* програму, за следеће податке:

- дужина транспортера од 1 m ÷ 6 m;
- ширина транспортне траке од 0,5 m ÷ 1 m;
- маса материјала по дужном метру од 10 kg ÷ 100 kg;
- брзина транспортне траке максимално 15 m/min односно 0,25 m/s ;
- носећа конструкција од алуминијумског профила: главни носач *light* 30 *x* 60 mm;
- погонски механизам – произвођача *MÄDLER*.

3. ПРИБЛИЖАН ПРОРАЧУН ТРАКАСТОГ ТРАНСПОРТЕРА

Силе у траци тракастог транспортера одређују се методом обиласка контуре крећући се од једне, произвољно изабране тачке, у смеру њеног кретања.



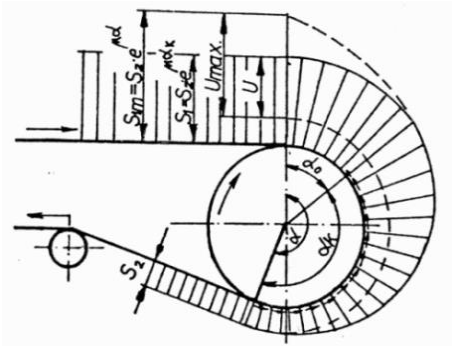
Слика 2. *Шема трасе тракастог транспортера*

Израчунавајући отпоре на појединим деловима транспортера, полазећи од силазне тачке са погонског бубња (тачка 2), добијају се силе у означеним карактеристичним тачкама у функцији S_2. На овај начин се добија систем једначина чији је број за један мањи од броја карактеристичних тачака. За добијање решења потребно је увести још један, додатни услов. Код тракастих транспортера где се трака ослања на радној страни на ваљке, овај услов се формира на основу критеријума којим је одређена минимална вредност силе на радној страни траке, чиме се обезбеђују услови да не дође до расипања материјала са траке због превеликог угиба.

Код предметних тракастих транспортера то није случај, тако да ће овај услов послужити за одређивање сила у траци за приближан прорачун, који ће послужити као основа за завршни прорачун. Приближан прорачун је детаљно одрађен у мастер раду, а након њега је урађен и прорачун потребне снаге погонског електромотора, што је дало само оквирне вредности потребних снага док се не изврши завршни прорачун тракастог транспортера.

4. ЗАВРШНИ ПРОРАЧУН ТРАКАСТОГ ТРАНСПОРТЕРА

Након урађеног приближног прорачуна потребно је одредити степен сигурности против проклизавања траке. Теорија преноса оптерећења путем трења између бубња и траке заснива се на Ојлеровом обрасцу који дефинише однос сила у наилазном и силазном краку траке у тренутку почетка њеног проклизавања по бубњу [1].



Слика 3. *Механизам преноса оптерећења*

Степен сигурности против проклизавања добија се према следећем изразу:

$$\varphi = \frac{S_2 \cdot (e^{\mu\alpha} - 1)}{S_1 - S_2} \geq 1{,}25 \qquad (1)$$

где су:

S_1– сила у наилазном краку траке;

S_2 – сила у силазном краку траке;

μ – коефицијент трења између траке и бубња;

α – обухватни угао траке по бубњу.

Како је израчунати степен сигурности против проклизавања знатно изнад дозвољене вредности, то значи да је сама транспортна трака превише оптерећена, односно превише затегнута. То може довести до смањења њеног радног века, али и стварања додатних оптерећења на вратила и лежајеве што може изазвати потребу за већим димензијама вратила, а самим тим и лежајева.

Због свега наведеног приступило се смањењу минимално дозвољене силе у траци, али тако да се и даље одржи степен сигурности у дозвољеним границама. На основу степена сигурности врши се поновно израчунавање претходно израчунатих сила у траци, односно њихова корекција. Сила у наилазном краку траке, у том случају, може се записати као збир силе у силазном краку траке и свих отпора дуж трасе, односно:

$$S_1 = S_2 + \Delta S_{2-3} + \Delta S_{3-4} + \Delta S_{4-1} \qquad (2)$$

Поступак тражења сила у карактеристичним тачкама трасе транспортера детаљно је обрађена у оквиру мастер рада.

Након корекције сила у траци приступило се одређивању стварне потребне снаге погонског електромотора према следећем изразу:

$$P = \frac{U \cdot v}{\eta} \qquad (3)$$

где су:

U – обимна сила која је једнака разлици сила у наилазној и силазној тачки са погонског бубња;

η – степен искоришћења погонског механизма;

v – брзина кретања траке транспортера.

Након одређених снага електромотора, што је детаљно урађено у оквиру мастер рада, дат је приказ потребних и стандардних снага електромотора у зависности од масе материјала по дужном метру и дужине трасе транспортера.

Табела 1. *Потребне снаге електромотора*

Снага P [kW]		Маса материјала по дужном метру					
		10 [kg]	20 [kg]	30 [kg]	40 [kg]	50 [kg]	100 [kg]
Дужина транспортера	1 [m]	0,015	0,025	0,034	0,044	0,054	0,103
	2 [m]	0,029	0,049	0,068	0,087	0,107	0,203
	3 [m]	0,044	0,073	0,102	0,130	0,159	0,303
	4 [m]	0,058	0,097	0,135	0,173	0,212	0,403
	5 [m]	0,073	0,121	0,169	0,216	0,264	0,503
	6 [m]	0,088	0,145	0,202	0,259	0,317	0,603

Табела 2. *Стандардне снаге електромотора*

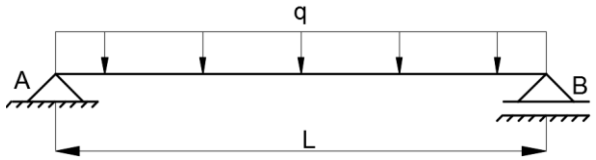
Снага $P\ [kW]$		Маса материјала по дужном метру					
		10 $[kg]$	20 $[kg]$	30 $[kg]$	40 $[kg]$	50 $[kg]$	100 $[kg]$
Дужина транспортера	1 $[m]$	0,12	0,12	0,12	0,12	0,12	0,12
	2 $[m]$	0,12	0,12	0,12	0,12	0,12	0,25
	3 $[m]$	0,12	0,12	0,12	0,25	0,25	0,37
	4 $[m]$	0,12	0,12	0,25	0,25	0,25	0,75
	5 $[m]$	0,12	0,25	0,25	0,25	0,37	0,75
	6 $[m]$	0,12	0,25	0,25	0,37	0,37	0,75

5. ПРОРАЧУН НОСЕЋЕ КОНСТРУКЦИЈЕ ТРАКАСТОГ ТРАНСПОРТЕРА

У оквиру ове тачке следи опис прорачуна целе носеће конструкције.

5.1. Прорачун главног носача носеће конструкције-доказ чврстоће

Према пројектном задатку, као главни носач тракастог транспортера користи се алуминијумски профил за који је потребно извршити прорачун носивости, односно спровести доказ чврстоће носача. Носач се моделира као проста греда која је оптерећена континуалним оптерећењем q.



Слика 4. *Модел главног носача*

Како транспортер има два подужна главна носача, за прорачун је узет један носач и на њега је редуковано оптерећење које потиче од терета. Међутим, у обзир је узет најнеповољнији случај, односно претпоставка да ће на један главни носач да се пренесе 60% укупног оптерећења. Разлог томе је могући неправилни рад траке при раду транспортера, тј. њеног закошења (шетање у подужном правцу) и неравномерне расподеле терета по ширини траке. На овај начин моделирано оптерећење иде на страну сигурности конструкције. Допуштени напони, нормални и тангенцијални се одређују на основу следећих образаца [2]:

$$\sigma_{dop} = \frac{R_{p0,2\%}}{\nu_1} \tag{4}$$

$$\tau_{dop} = \frac{R_{p0,2\%}}{\nu_1 \cdot \sqrt{3}} \tag{5}$$

где су:

$R_{p0,2\%}$ – конвенцијални напон течења;

ν_1 – степен сигурности за I случај оптерећења: оптерећење при нормалном раду без ветра.

Овде је дат табеларни приказ добијених резултата упоредних напона.

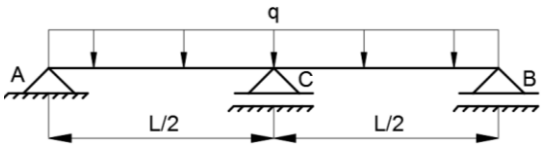
Табела 3. *Приказ резултата упоредних напона*

Напон $\sigma_u\ [MPa]$		Маса материјала по дужном метру					
		10 $[kg]$	20 $[kg]$	30 $[kg]$	40 $[kg]$	50 $[kg]$	100 $[kg]$
Дужина транспортера	1 $[m]$	1,06	2,13	3,2	4,26	5,33	10,66
	2 $[m]$	4,26	8,53	12,8	17,06	21,32	42,65
	3 $[m]$	9,6	19,2	28,8	38,4	48	96
	4 $[m]$	17	34,12	51,2	68,25	85,3	**170,61**
	5 $[m]$	26,65	53,31	80	106,63	**133,3**	**266,6**
	6 $[m]$	38,4	76,8	115,17	**153,55**	**191,95**	**383,9**

Са слике се види за које дужине трасе и масе материјала је радни напон мањи од дозвољеног, тако да је ту чврстоћа задовољена што значи да неће доћи до лома главног носача. За оне дужине трасе и масе материјала где је радни напон већи од дозвољеног напона, што значи да ће доћи до лома главног носача тј. чврстоћа није задовољена, потребно је поставити још један ослонац на средини распона греде. Усвајање већег (јачег) профила није решење, јер је по пројектном задатку захтев да се главни носач ради само од једне врсте Alu профила. Увођењем трећег ослонца добија се статички неодређен систем о ком ће бити речи у наредном поглављу.

5.2. Прорачун главног носача носеће конструкције, статички неодређен систем-доказ чврстоће

У претходном поглављу се видело да је за неке случајеве потребан још један ослонац како би чврстоћа главног носача била задовољена, односно да би радни напони били у границама дозвољеног.



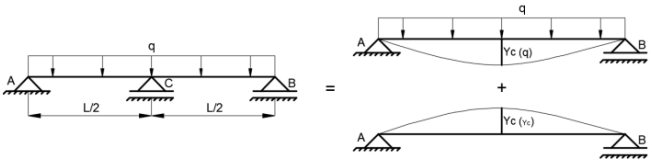
Слика 5. *Модел главног носача са три ослонца*

Приликом писања једначина равнотеже види се да је ово систем са 4 непознате реакције везе – 3 једначине равнотеже = једноструко статички неодређен систем. Према теорији из Отпорности материјала реакција ослонца која је додата (средњи ослонац) се проглашава за статички прекобројну величину. Да би овај систем једначина био решив, потребно је увести четврту (допунску) једначину, која представља геометријски услов деформације, у ком се каже да је реакција (додатог) ослонца једнака нули, односно:

$$Y_C = 0 \tag{6}$$

На основу теорије из Отпорности материјала статички неодређен систем се решава по принципу суперпозиције који је дат следећом једначином:

$$Y_C = Y_C{}^q + Y_C{}^{Yc} = 0 \tag{7}$$



Слика 6. *Принцип суперпозиције*

Комплетан поступак детаљно је урађен у оквиру мастер рада, а овде је дат приказ добијених резултата упоредних напона који нису, а који су након конструктивних измена задовољили услов да је упоредни напон мањи од дозвољеног.

Табела 4. *Упоредни напони који нису задовољили услов* $\sigma_u \leq \sigma_{dop}$

Напон $\sigma_u\ [MPa]$		Маса материјала по дужном метру		
		40 $[kg]$	50 $[kg]$	100 $[kg]$
Дужина трасе	4 $[m]$	/	/	**170,61**
	5 $[m]$	/	**133,3**	**266,6**
	6 $[m]$	**153,55**	**191,95**	**383,9**

Табела 5. *Упоредни напони који задовољавају услов* $\sigma_u \leq \sigma_{dop}$

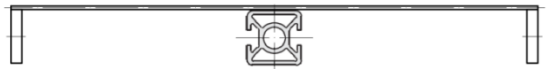
Напон σ_u $[MPa]$		Маса материјала по дужном метру		
		40 $[kg]$	50 $[kg]$	100 $[kg]$
Дужина трасе	4 $[m]$	/	/	**42,66**
	5 $[m]$	/	**33,32**	**66,65**
	6 $[m]$	**38,4**	**48**	**96**

5.3. Прорачун укрућења носеће конструкције-доказ чврстоће

Укрућење се моделира као проста греда на два ослонца, која је оптерећена континуалним оптерећењем q, као у поглављу 4.1.

Слика 7. *Приказ конструкције са укрућењем*

Са слике се види да се на транспортеру налази једно попречно укрућење које образује два поља, претпоставка је да то укрућење преузима половину оптерећења, а да се на преостала два поља преноси по четвртина укупног оптерећења. Како је укрућење сложеног попречног пресека потребно је пронаћи момент инерције тог пресека, као и његово тежиште да би се могли одредити радни напони и који делују у пресеку.



Слика 8. *Укрућење носеће конструкције сложеног попречног пресека*

Овде су дати табеларни резултати упоредних напона који делују унутар укрућења конструкције.

Табела 6. *Приказ резултата упоредних напона који делују у укрућењу конструкције*

Напон σ_u $[MPa]$		Маса материјала по дужном метру					
		10 $[kg]$	20 $[kg]$	30 $[kg]$	40 $[kg]$	50 $[kg]$	100 $[kg]$
Дужина транспортера	1 $[m]$	0,28	0,561	0,841	1,122	1,402	2,805
	2 $[m]$	0,658	1,316	1,975	2,633	3,292	6,584
	3 $[m]$	1,036	2,072	3,108	4,145	5,181	10,362
	4 $[m]$	1,414	2,828	4,242	5,656	7,070	14,141
	5 $[m]$	1,792	3,584	5,376	7,168	8,960	17,920
	6 $[m]$	2,169	4,339	6,509	8,679	10,849	21,698

6. ПРОРАЧУН ПОГОНСКОГ ВРАТИЛА ТРАКАСТОГ ТРАНСПОРТЕРА

Да би се обезбедио правилан рад и да не би дошло до пуцања вратила и нежељених хаварија код тракастог транспортера, потребно је извршити прорачун погонског вратила. Прорачун се врши за најтеже услове рада, што иде на страну сигурности вратила, односно претпоставља се да је оптерећење које изазива савијање вратила распоређено са 40-60 %. Овакав распоред оптерећења се може јавити као последица бочног померања траке услед неравномерно распоређеног терета по траци или нехомогености саме транспортне траке. Прорачун вратила се врши за случај, када је познат момент савијања (M) и момент увијања (T), који у овом случају потиче од погонског електромотора. У табели је дата препорука материјала за израду вратила, пречник вратила је рачунски добијен, $d = 25$ mm.

Табела 7. *Препорука материјала за израду вратила*

Материјал вратила		Маса материјала по дужном метру					
		10 $[kg]$	20 $[kg]$	30 $[kg]$	40 $[kg]$	50 $[kg]$	100 $[kg]$
Дужина транспортера	1 $[m]$	CF53	CF53	CF53	CF53	CF53	CF53
	2 $[m]$	CF53	CF53	CF53	CF53	CF53	CF53
	3 $[m]$	CF53	CF53	CF53	CF53	CF53	34CrNiMo6
	4 $[m]$	CF53	CF53	CF53	CF53	CF53	34CrNiMo6
	5 $[m]$	CF53	CF53	CF53	CF53	34CrNiMo6	34CrNiMo6
	6 $[m]$	CF53	CF53	CF53	34CrNiMo6	34CrNiMo6	34CrNiMo6

7. ЗАКЉУЧАК

Тракасти транспортери представљају уређаје непрекидног транспорта који се користе за транспорт различитих врста материјала више десетина година. При њиховом пројектовању тежи се повећању радног века у смислу правилног димензионисања његових делова и правилним одабиром стандардних компоненти (лежајеви, завртњеви, итд.), смањењу трошкова... У раду су дефинисане многе савремене изведбе неких техничких решења, формирана је квалитетна подлога за израду техничке документације уз поштовање правила дефинисаних правилницима и стандардима. Такође, у раду су дефинисане подлоге за аутоматизацију појединих сегмената пројеката тракастих транспортера, који би требали да постану уобичајени у инжењерској пракси.

8. ЛИТЕРАТУРА

[1] Д. Живанић: Непрекидни и аутоматизовани транспорт - скрипта, Нови Сад 2019.

[2] Зоран Петковић, Давор Острић: Маталне конструкције у машиноградњи 1, Београд 1996.

Кратка биографија:

Никола Бодрожић рођен у Сремској Митровици 1996. год. Дипломирао 2019. године на Факултету техничких наука, смер машинство, на којем исте године уписује мастер студије на усмерењу за машинске конструкције, транспортне системе и логистику.

Драган Живанић рођен у Сремској Митровици 1972. год. Докторирао је 2012. год, од 2014. ради као доцент, а од 2019. као ванредни професор на Факултету техничких наука у Новом Саду.

Никола Иланковић рођен у Суботици 1994. год. Завршио мастер академске студије 2018. године на Факултету техничких наука и исте године уписао докторске студије машинства. Запослен је у звању асистент мастер на Факултету техничких наука у Новом Саду.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 620.9:692/697
DOI: https://doi.org/10.24867/10AM05Djuric

UTICAJ KVALITETA UNUTRAŠNJE KLIME NA OSJEĆAJ UGODNOSTI KORISNIKA

THE IMPACT OF INDOOR ENVIRONMENTAL QUALITY ON THE OCCUPANTS COMFORT

Gordana Đurić, *Fakultet tehničkih nauka, Novi Sad*

Oblast- MAŠINSTVO

Kratak sadržaj – *U radu su date teorijske osnove na polju analize kvaliteta unutrašnje sredine, načini postizanja termičkog komfora, dosta je rečeno o uslovima za ostvarivanje akedvatnog kvaliteta unutrašnje sredine - KUS, kao i prnicipima mjerenja parametara KUS. Kroz četiri studije, prikazano je mjerenje parametara za određivanje termičkog komfora u unutrašnjoj sredini, kao i uticaj CO_2 na KUS i kvalitet unutrašnjeg vazduha - KUV, odnosno izvršena je sveobuhvatna analiza KUS.*

Abstract - *The paper presents the theoretical foundations in the field of analysis of the quality of the internal environment, ways to achieve thermal comfort, much has been said about the conditions for achieving adequate quality of the internal environment - IEQ, as well as the principles of measuring IEQ parameters. Through four studies, the measurement of parameters for determining the thermal comfort in the indoor environment, as well as the impact of CO2 on IEQ and indoor air quality - IAQ, ie a comprehensive analysis of IEQ was performed.*

Keywords: *Air conditioning, indoor climate quality, thermal comfort*

1. UVOD

Jedan od osnovnih zahteva projektovanja zgrade je određivanje kvaliteta unutrašnje sredine, prvenstveno čistog, svežeg vazduha i termičkih uslova ugodnosti za ljude koji borave ili rade u zgradama. Zgrade i dostupna infrastruktura imaju ogroman uticaj na kvalitet života i životnu sredinu. Svetska statistika ukazuje da se taj uticaj manifestuje kroz korišćenje preko 30% ukupne potrošnje energije, 35% prirodnih resursa, 12% vode za piće i preko 30% emisije CO_2. Potrebna količina svežeg vazduha se obezbeđuje prirodnom, mehaničkom ili kombinovano prirodno/mehaničkom ventilacijom. Prirodna ventilacija je jedan od načina da se smanji potrošnja energije u odnosu na potrošnju energije sa mehaničkom ventilacijom, a time poveća energetska efikasnost objekta. Za postizanje adekvatnog kvaliteta unutrašnje sredine neophodni su pravilni sistemi grejanja, ventilacije i klimatizacije, utvrđivanje unutrašnjih zagađivača kao što su mikroorganizmi iz vazduha, čestice i isparljiva organska jedinjenja, ispitivanje buke, utvrđivanje jačine osvetljenja, procena rizika od nekih opasnih materija, kontrola kvaliteta vode u klimatskim uređajima i optimizacija upotrebe energije.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio doc. dr Aleksandar Anđelković.

2. TEORIJSKE OSNOVE NA POLJU ANALIZE KVALITETA UNUTRAŠNJE SREDINE

KUS obuhvata uslove unutar zgrade, kao što su kvalitet vazduha, osvetljenje, buku, toplotne uslove i njihove uticaje na stanovnike. Unapređenjem kvaliteta unutrašnje sredine postiže se zaštita zdravlja ljudi, poboljšava se kvalitet života, kao i blagostanje ljudi. Pored toga, mogu se dobiti i druge koristi koje proističu iz ovih poboljšanja, poput povećane produktivnosti na poslu, poboljšanog učenja u nastavnim okruženjima i povećane vrednosti dotičnih prostora i zgrada. Kriterijumi koji se koriste u zatvorenom okruženju, kao i projektovanje i rad zgrade imaju veliki uticaj na potrošnju energije zgrade.

2.1. Termički komfor

Termički komfor igra veliku ulogu u načinu na koji doživljavamo mesta u kojima živimo i radimo. Termički komfor ili udobnost predstavlja stanje uma koje izražava zadovoljstvo termičkim okruženjem. Stanje termičkog komfora je postignuto kada je osoba zadovoljna sa temperaturom, vlažnošću i strujanjem vazduha u svojoj neposrednoj okolini i ne oseća potrebu za promenom niti jedne od navedenih veličina. Ovako definisan termički komfor predstavlja subjektivan i individualan osećaj, pa je zbog toga nemerljiva veličina. Dve vrste termičkog komfora:

1. Termički komfor sa prinudnom ventilacijom
2. Adaptivni termički komfor

2.2. Kvalitet unutrašnjeg vazduha

Zahvaljujući sistemu klimatizacije, ostvaruje se veliki broj funkcija u cilju postizanja uslova ugodnosti tokom cele godine. Osnovne funkcije su: zagrijevanje prostora u zimskom periodu i hlađenje prostora u ljetnjem periodu, održavanje relativne vlažnosti, održavanje potrebnog nivoa čistoće vazduha i ventilacija. Sve ove funkcije utiču na kvalitet vazduha u zatvorenom prostoru. Kako su se klimatizacioni sistemi tokom vremena sve više razvijali cilj je bio da se što više energija uštedi i to je dovodilo do smanjenja broja izmjena svježeg vazduha na sat, što automatski predstavlja problem lošeg kvaliteta vazduha u klimatizacionim prostorijama.

2.3. Osvetljenje

Da bi ljudi mogli efikasno i precizno obavljati vizuelne zadatke, mora biti obezbeđena odgovarajuća rasveta. Kriterijumi osvjetljenja biraju se u skladu sa zadacima i aktivnostima koje preduzimaju i moraju obezbediti komforne uslove za osobe koje borave unutar prostorije. Nivo osvetljenosti se dobija u zavisnosti od sredstava za dnevnu svetlost, električno svetlo ili kombinaciju ova dva. Dizajn prozora ne sme prouzrokovati nelagodu vida zbog odsjaja, a toplotno opterećenje od svetlostnih sistema se

uzima u obzir prilikom izračunavanja potrošnje energije u nestambenim objektima zgrade za grejanje i hlađenje. Prozori treba da budu glavni izvor svetlosti tokom dana, koji takođe pružaju kontak sa spoljnim okruženjem. Da bi se postigla veća udobnost, a smanjila potrošnja energije upotreba dnevne svetlosti je vrlo poželjna. To sve zavisi od različitih faktora, kao što su lokacija zgrade, količina dnevnog svetla leti i zimi i drugi.

2.4. Buka

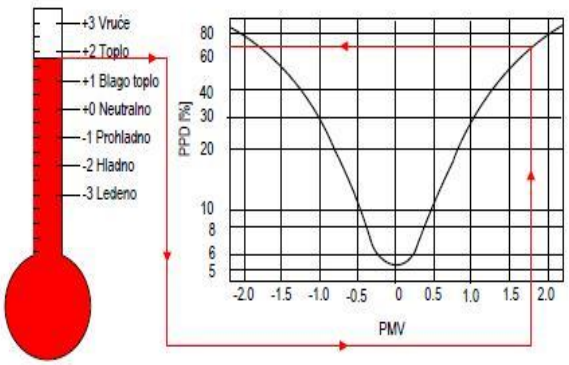
Za adekvatan kvalitet unutrašnje sredine neophodno je navesti potrebne nivoe zvuka. Buka koja nastaje od sistema za servisiranje zgrade može narušiti udobnost unutar prostorije, kao i sprečiti ili narušiti nameravanu upotrebu prostora. Buka u prostoru mora se proceniti korišćenjem A-ponderiranog ekvivalentnog nivoa zvučnog pritiska, koje je normalizovan u odnosu na vreme reverberacije, da bi se uzela u obzir apsorpcija zvuka u prostoriji. Buka ventilacionih i klimatizacionih sistema se takođe može koristiti za maskiranje ostalih izvora zvuka i poboljšati akustičnu privatnost. Takođe neophodna je i potrebna izolacija zgrade kako bi se izbegla buka iz spoljašnjih i susjednih prostorija. Ventilacija se ne oslanja na operativne prozore, ako se zgrada nalazi u prostoru sa visokom visinom spoljašnje buke u poređenju sa nivoom koji treba da se postigne u prostoru unutar zgrade.

2.5. Principi određivanja termičkog komfora

ANSI/ASHRAE Standard 55 predstavlja metode za određivanje termičkih uslova, odnosno parametara za postizanje kvaliteta unutrašnje sredine. Predviđanje udobnosti pomoću fizičkog merenja zasnovano je na dva osnovna principa:

- ✓ PMV metod (Predicted Mean Vote) – predviđeno srednje glasanje, odnosi se na prinudnu ventilaciju
- ✓ Adaptivni metod – primarno prirodna ventilacija

PMV predstavlja najistaknutiji model toplotne udobnosti i može se primeniti na klimatizovane zgrade, dok se adaptivni može primeniti samo na zgrade u kojima nisu instalirani nikakvi mehanički sistemi (slika1.).



Slika 1. *Grafički metod – predviđen procenat nezadovoljnih*

3. PRINCIPI MERENJA PARAMETARA UNUTRAŠNJE SREDINE

Standard ISO 7726 jedan u nizu međunarodnih standarda namenjenih za ispitivanje termičkog okruženja [6]. Ova serija međunarodnih standarda posebno se bavi definisanjem termina koji će se koristiti u metodama merenja, ispitivanja i tumačenja, uzimajući u obzir i standarde koji već postoje. Ovaj standard precizira minimalne karakteristike instrumenata za merenje fizičkih veličina karakterišući okruženje kao i metode za merenje fizičkih veličina ove sredine. Cilj mu nije da definiše opšti indeks komfora, već jednostavno da standardizuje proces beleženja informacija koje vode do određivanja takvih indeksa.

Određivanje ukupnih indeksa komfora zahteva znanje fizičkih veličina povezanih sa okolinom, ove fizičke veličine dele se u dve grupe, prema njihovom stepenu zavisnosti od okoline:

- osnovne fizičke veličine - karakterišu jedan od faktora okruženja nezavisno od drugog
- izvedene fizičke veličine – karakterišu grupu faktora okruženja prema karakteristikama senzora koji se koristi

4. ODREĐIVANJE TERMIČKOG KOMFORA U UNUTRAŠNJOJ SREDINI

Kao što je već rečeno parametri životne sredine koji čine termalno okruženje su: temperatura vazduha, zračenja i površine, vlažnost vazduha, brzina vazduha i lični parametri, kao što su odeća i nivo aktivnosti. Kriterijumi za prihvatljivu termalnu klimu su specificirani kao zahtevi za opšti termički i lokalni komfor.

Opšti toplotni komfor se izražava radnom temperaturom, brzinom vazduha i vlažnošću vazduha, dok lokalni komfor se predstavlja promenom srednje brzine i temperature vazduha, vertikalnim razlikama u temperaturi vazduha, asimetrijom zračenja i temperaturom poda.

Parametrima koji predstavljaju predviđeno srednje glasanje lica koja borave unutar prostorije – PMV (Predicted Mean Vote) se vrši ocena ugodnosti unutar prostorije. Takođe koristi se još jedan parametar za predstavljanje termičkog komfora, a to je predviđeni procenat nezadovoljnih osoba - PPD (Predicted Percentage Dissatisfied).

Studija koja se bavi poređenjem termičkih uslova u dve različite kancelarije (studija 03), pokazuju da u zgradama sa slobodnim kretanjem vazduha, bez mehaničkog hlađenja, prirodno provetravanim, ljudi se psihološki prilagođavaju zatvorenom prostoru sa većim temperaturama, nego što je to predviđeno PMV modelom. PMV-PPD indeksi izražavaju topli i hladni osećaj za telo u celini. Termalno nezadovoljstvo je uzrokovano neželjenim hlađenjem ili zagrijavanjem jednog određenog dela tela - lokalna neprijatnost. Lokalna toplotna nelagoda može biti uzrokovana strujanjem vazduha, velikom vertikalnom temperaturnom razlikom između glave i stopala, podom koji je previše topao ili hladan .

5. UTICAJ CO_2 NA KUV i KUS

Kako bi se pokazala veza kvaliteta vazduha u zatvorenom prostoru i koncentracije CO_2, odnosno uticaj ugljen dioksida na KUS i KUV izvršena su različita merenja i eksperimentalna ispitivanja predstavljena kroz:

- Studija uticaja CO_2 na nezadovoljstvo korisnika objekta [7].
- Studija proračuna emisije CO_2 od strane ljudi pri dve različite aktivnosti [10].

Ove studije imaju za cilj da dokažu uticaj zagađivača na kvalitet vazduha, kao što u CO_2, organska isparljiva

jedinjenja, čestice i drugi. Takođe prikazuje i nezadovoljstvo lica koja borave unutar prostorije termičkim komforom, i uticaj ugljen dioksida na ljude koji se bave različitim aktivnostima, na primer jedna osoba sedi, druga se kreće ili slično. Istraživanja su obavljena na Danskom tehničkom univerzitetu (DTU)

6. SVEOBUHVATNA ANALIZA KUS

Za ocenu KUV koristi se osam reprezentativnih parametara okruženja u zatvorenom prostoru. Najbolji način da se analizira i proceni okruženje u zatvorenom prostoru, odnosno da se objasni sveobuhvatna analiza KUS, jeste da se detaljno razjasni studija određivanja kvaliteta unutrašnje sredine u dve kancelarije. Kako bi se utvrdio KUS izvršeno je merenje osam reprezentativnih parametara u tri kancelarije i sale za sastanke na drugom spratu zgrade 402, zgrada je bez mehaničke ventilacije, kancelarije su iste veličine i orijentacije. Upravljanje prozorima i roletnama je ručno. Prostori za sastanke su ocenjivani pojedinačno, jer su različite veličine i imaju različite vrste funkcije. U tabeli 24. su navedeni parametri od koji zavisi kvalitet unutrašnjeg okruženja.
Parametri koji su već izmereni upoređuju se sa standardnim vrednostima da bi se dobila procena da li je zgrada u skladu sa odgovarajućom kodom. Na osnovu dobijenih i standardnih vrednosti klasifikovan je građevinski KUV. Proračunata predviđena srednja vrednost glasa - PMV i procenat nezadovoljnih - PPD procenjuje izgradnju KUS. Kada se odredi PMV i PPD, rezultat se analizira i komentariše s obzirom na parametre za zatvoreno okruženje. Posle izvršene ankete stanara dobijeni podaci su obrađeni, a prevalencija simptoma i žalbe na KUV predstavljene su u tabelama 1. i 2.

Tabela 1. *Prevalencija simpoma*

Simptomi	Nikad	Ponekad	Često	Svaki dan	Povezana izgradnja
Umor	29,4	64,7	5,9	0,0	0,0
Osjećaj teške glave	47,1	52,9	0,0	0,0	0,0
Glavobolja	29,4	70,6	0,0	0,0	0,0
Vrtoglavica/mučnina	29,4	58,8	1,8	0,0	5,9
Teškoća za koncentraciju	64,7	29,4	5,9	0,0	0,0
Svrab, pečenje	35,3	41,2	23,5	0,0	0,0
Nadražen, začepljen nos	70,6	23,5	5,9	0,0	0,0
Promuklo, suvo grlo	64,7	17,6	11,8	5,9	0,0
Suva, svrbežna crvena koža	100,0	0,0	0,0	0,0	0,0
Drugi	0,0	0,0	0,0	0,0	0,0

Tabela 2. *Pritužbe u zatvorenom*

Komfor	Nikad	Ponekad	Često	Svaki dan	Zbir dnevne i nedeljne prevalencije
Previsoka sobna temperatura	81,25	0	18,75	0	18,75
Variranje sobne temperature	62,50	18,75	18,75	0	18,75
Preniska sobna temperatura	50,00	18,75	18,75	6,25	25,00
Strujanje vazduha	68,75	25,00	6,25	0	6,25
Zagušen/loš vazduh	31,25	31,25	25,00	12,50	37,50
Suv vazduh	62,50	31,25	6,25	0	6,25
Buka	37,50	31,25	18,75	12,50	31,25
Prigušena svjetlost	75,00	12,50	12,50	0	12,50
Odsjaj na monitoru	62,50	25,00	12,50	0	12,50
Nedovoljno svjetla	75,00	6,25	6,25	12,50	18,75
Prašina/prljavština	56,25	31,25	0	12,50	12,50
Duvanski dim	100	0	0	0	0

7. ZAKLJUČAK

Kroz četiri studije predočena je metodologija postizanja adekvatnog KUS, na koji način koncentracije CO_2 utiču na nezadovoljstvo korisnika. Iz sprovedenih merenja tokom istraživanja na Danskom Tehničkom univerzitetu mogu se izvesti određeni zaključci. Ljudi koji rade u kancelarijama za vreme svog rada stvaraju znatne količine CO_2.

U slučaju kada nema dobre ventilacije dolazi do porasta koncentracije CO_2 preko dozvoljenih vrednosti koja utiče na koncentraciju i rad ljudi. Iz sprovedenih eksperimenata može se zaključiti da za vreme radnog vremena i boravka ljudi unutar prostorije raste temperatura vazduha, te dolazi do smanjenja vlažnosti vazduha. Suv vazduh uz minimalne količine relativne vlažnosti takođe nepovoljno deluje na ljude unutar kancelarije.

Kako se smanjuje koncentracija ugljen dioksida povećava se kvalitet unutrašnjeg vazduha, samim tim kvalitet unutrašnje sredine i smanjuje se negativan uticaj na zdravlje lica koja borave unutar prostorije. Studijama je dokazano da su u renoviranim zgradama, sa boljom izolacijom i bez mehaničke ventilacije postignuti bolji rezultati termičkog komfora, takođe potvrđeno je da je otkrivena eksponencijalna i pozitivna korelacija između CO_2 i procenta nezadovoljnih.

Jedna od studija pokazuje da nivo koncentracije ugljen dioksida zavisi od nivoa aktivnosti, pri niskom nivou aktivnosti koncentracija se smanjuje, takođe i tokom noći koncentracija CO_2 opada na spoljnu koncentraciju.

8. LITERATURA

[1] Aleksandar Anđelković, Grejanje, ventilacija I klimatizacija, Faklutet tehničkih nauka, Novi Sad, 2017.
[2] Aleksandar Anđelković, Nekonvencionalni sistemi grejanja i hlađenja, Faklutet tehničkih nauka, Novi Sad, 2017.
[3] John F. Dunlap, ANSI/ASHRAE Standard 55 – 2019, Physiology and Human Environment.
[4] The Europen Standard, EN ISO 7730 – 2005, 2007 British standard.
[5] The Europen Standard, EN ISO 16798-1 – 2019, English version.
[6] Dansk Standard, DS/EN ISO 7726
[7] Studija - Uticaja CO_2 na nezdovoljstvo korisnika objekta, Technical University of Denmark , 2013
[8] Studija - Određivanje kvaliteta unutrašnjeg okruženja u 2 kancelarije, Technical University of Denmark , 2013
[9] Studija –Poređenje termičkih uslova u novoj i staroj kancelariji, Technical University of Denmark , 2013
[10] Studija – Proračun emisije CO_2 od strane ljudi pri dvije različite aktivnosti, Technical University of Denmark , 2013
[11] Mjerenje toplinskog komfora, TURCERT, dostupno na: https://www.sertifikasyon.net/hr/hizmet/termal-konfor-olcumu/
[12] Zoran Baus, Ergonomijski Prilagodljivo Neizrazito Vođenje Procesa Pri Održavanju Toplinskog Komfora U Mjernom Laboratoriju, Doktorska Disertacija, Zagreb, 2004, Dostupno Na: Https://Manualzz.Com/Doc/15505059/Ergonomijski---Prilagodljivo-Mjernom--Laboratoriju

[13] Model predviđanja toplinkse ugodnosti prostora, Tehnički fakultet, Sveučilište u Rijeci (Faculty of Engineering, University of Rijeka) Rijeka,Republic of Croatia, dostupno na: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjK1LzdutnrAhUjlosKHRTmCv4QFjABegQICBAB&url=https%3A%2F%2Fhrcak.srce.hr%2Ffile%2F77596&usg=AOvVaw13NJYd4M1ssisiUDcmhwoE
[14] Instruction Manula, Indoor Climate Analyzer Type 1213, dostupno na: https://www.pearlhifi.com/06_Lit_Archive/15_Mfrs_Publications/10_Bruel_Kjaer/03_Instruction_Manuals/B&K%201213%20Instruction%20Manual.pdf

Kratka biografija:

Gordana Đurić, rodjena u Vlasenici, BiH RS 1995. godine.
Fakultet tehničkih nauka, Novi Sad
Mašinstvo-Energetika i procesna tehnika
Osnovne studije završila 2018. godine

Email: gordanadubocanin88@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.436
DOI: https://doi.org/10.24867/10AM06Cavic

PRIMENA EPICIKLIČNIH ZUPČASTIH PRENOSNIKA SA NEOKRUGLIM ZUPČANICIMA ZA POVEĆANJE STEPENA KORISNOSTI SUS MOTORA

APPLICATION OF EPICYCLIC GEAR TRAINS WITH NON-CIRCULAR GEARS FOR INCREASING IC ENGINE EFFICIENCY

Dijana Čavić, Milan Rackov, *Fakultet tehničkih nauka, Novi Sad*

Oblast – MEHANIZACIJA I KONSTRUKCIONO MAŠINSTVO

Kratak sadržaj – *Povećanje efikasnosti SUS motora je tema mnogih diskusija u struci, i jedan pristup koji se sreće je modifikacija termodinamičkog radnog ciklusa motora kojom se realni cilkus približava idealnom. Ovaj rad pokušava to da postigne izmenom kinematike klipnog mehanizma uvođenjem epicikličnih neokruglih zupčanika između kolenastog vratila i zamajca. Dodatak neokruglih zupčanika menja radni ciklus uvođenjem prisilnog perioda mirovanja klipa tokom ključnih momenata tokom ciklusa. Moguća rešenja oblika zupčanika izračunata su na osnovu prenosnog odnosa izraženog poznatom periodičnom funkcijom. Konačni rezultat je epiciklični prenosnik sa neokruglim zupčanicima sa primetnim uticajem na kinematiku klipnog mehanizma, što za posledicu ima povećanje efikasnosti motora.*

Ključne reči: *SUS motori, Povećanje efikasnosti, Epiciklični prenosnici sa neokruglum zupčanicima*

Abstract – *Increasing the efficiency of IC engines is a frequently discussed topic and one of the approaches taken is to modify the thermodynamic work cycle of the engine so that it resembles the ideal cycle more closely. This paper attempts to achieve that by altering the kinematics of the piston-crank mechanism by introducing an epicyclic non-circular gear train between the crank and the flywheel. The addition of the non-circular gears alters the work cycle by introducing a state of dwell during key points of the cycle. Possible gear shape solutions were calculated using a known, periodical function to express the gear transmission ratio. The final result was an epicyclic non-circular gear train that has a noticeable effect on the kinematics of the piston-crank mechanism which in turn increases the engine's efficiency.*

Keywords: *IC Engine, Increasing efficiency, Epicyclic non-circular gear train*

1. UVOD

Niz termodinamičkih procesa koji se odvijaju u cilindru SUS motora se naziva radnim ciklusom.
Usled kompleksnosti ovih procesa, radni ciklus se analizira kroz različite matematičke modele.U teoriji, najefikasniji termodinamički ciklus u kontekstu SUS motora jeste OTO ciklus, koji obuhvata izentropsko sabijanje, i izohorsko dovođenje i odvođenje toplote [1].
U SUS motorima sa konvencionalnim klipnim mehanizmom nije moguće postići izohorsko sagorevanje, što se drastično odražava na efikasnot. U idealnom scenariju, sagorevanje će početi i završiti se dok klip stoji u gornjoj mrtvoj tački. Ovim bi se postigao maksimalni termalni potencijal i eliminisao negativni rad usled predpaljenja tokom takta sabijanja. Jedan način da se ovo postigne je da se prisilno smanji brzina klipa oko gornje mrtve tačke i time pruži procesu sagorevanja više vremena [2]. Modifikacije klipnog mehanizma s ciljem povećanja efikasnosti SUS motora predmet su mnogih istraživanja [3–5].
Ovaj rad razmatra mogućnost povećanja efikasnosti SUS motora modifikovanjem klipnog mehanizma uvođenjem neokruglih zupčanika, čime se menja zakon kretanja klipa tako da klip miruje oko gornje mrtve tačke. Cilj rada je da predloži rešenje kojim se postiže tražena izmena zakona kretanja klipnog mehanizma SUS motora upotrebom neokruglih zupčanika i da analizira i prikaže uticaj modifikacije na ceo mehanizam.

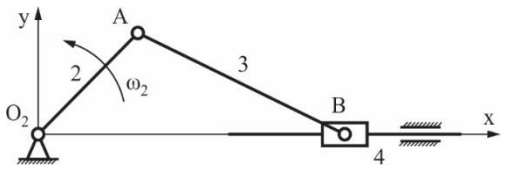
2. STANJE U OBLASTI

Neokrugi zupčanici imaju široku primenu u mašinstvu i koriste se u situacijama kada je potrebno obezbediti proizvoljno kretanje izlaznog člana mehanizma [6]. Primena neokruglih zupčanika u sklopu prese sa definisanim zakonom kretanja izlaznog člana prikazana je u [7]. Takođe u [8]. Primena neokruglih zupčanika za generisanje funkcija prikazano je u [9]. Novo rešenje mašinice za pecanje sa neokruglim zupčanicima prikazano je u [10]. Sinteza prenosnika koji omogućava period mirovanja, a obuhvata ravanski mehanizam i neokrugle zupčanike prikazana je u [11]. Sinteza mehanizma prekidnog kretanja koji sadrži neokrugle zupčanike prikazana je u [12, 13]. Iako je kinematika neokruglih zupčanika dobro istražena [14–16], oni za sad nisu našli široku primenu u industriji, usled poteškoća pri izradi. Različiti postupci narezivanja zuba obrađeni su u [17–20].

3. MODIFIKOVANI MEHANIZAM

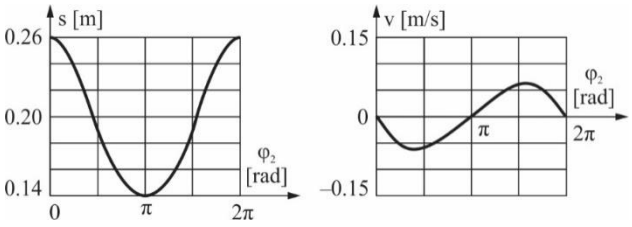
Na slici 1 prikazan je konvencionalni mehanizam, koji se sastoji iz kolenastog vratila, klipnjače i klipa. Iako je klip teorijski pogonski član, u tri od četiri takta stvarni pogon dolazi od zamajca usled akumulirane kinetičke energije. Usled toga, kolenasto vratilo će se u daljem radu smatrati pogonskim članom.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Milan Rackov, vanred. prof.



Slika 1. *Šema konvencionalnog klipnog mehanizma*

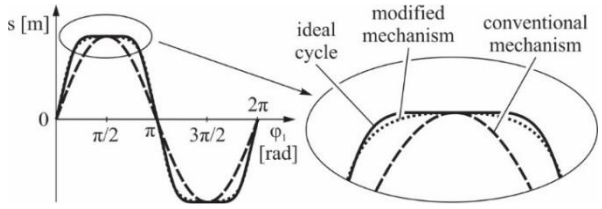
Zakoni kretanja klipa konvencionalnog klipnog mehanizma prikazani su na slici 2. Konstruktivni parametri potrebni za kinematsku analizu usvojeni su iz literature [1].



Slika 2. *Zakoni kretanja konvencionalnog klipnog mehanizma (levo -pređeni put; desno – brzina)*

3.1. Modifikacija

Na slici 2 vidi se da je brzina klipa jednaka nuli samo u trneutku prolaska kroz mrtve tačke.
Na slici 3 prikazani su zakoni kretanja klipa u slučaju idealnog ciklusa, za konvencionalni mehanizam i za modifikovani mehanizam.



Slika 3. *Različiti zakoni kretanja klipa*

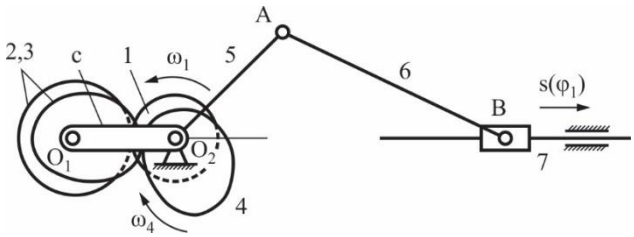
Konvencionalni klipni mehanizam radi sa približno konstantnom ulaznog brzinom usled delovanja zamajca. Uvođenjem promenljive ulazne brzine postaje moguće uticati na zakon kretanja klipa.

Ipak, pošto je klipni mehanizam samo jedan deo SUS motora, koji je opet samo deo kompleksnog pogona kretanja automobila, postoje dodatni zahtevi koje bi modifikovani mehanizam morao ispuniti. Dakle, zahtevi postavljeni pred modifikovani mehanizam su sledeći:

- Da postiže promenljivu ulaznu brzinu kolenastog vratila potrebnu za postizanje propisanog zakona kretanja klipa,
- Da se izbegnu nepotrebne konstruktivne komplikacije
- Da se izbegne zaustavljanje ostatka transmisije.

Na slici 4 prikazan je mehanizam kojim se može postići period pravog mirovanja. Zupčanik vezan za zamajac se smatra ulaznim i ima približno konstantnu brzinu.

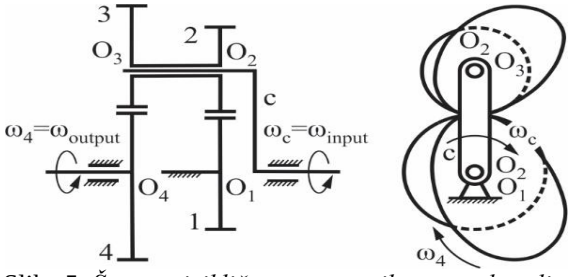
Zupčanik vezan za kolenasto vratilo ima promenljivu brzinu koja zavisi od oblika zupčanika, što teoretski znači da je oblikom zupčanika moguće postići proizvoljnu promenu brzine, a s time i proizvoljan zakon kretanja klipa.



Slika 4. *Modifikovani mehanizam sa epicikličnim neokruglim zupčanicima*

3.2. Kinematika

Slika 5 prikazuje epiciklični prenosnik sa neokruglim zupčanicima.. Jednostepeni epiciklični prenosnik sastoji se iz pet elemenata: krivaja c, centralni zupčanici 1 i 4, i planetni zupčanici 2 i 3 koji su kruto povezani. Centralni zuščanik 4 je izlazni, a krivaja c je u vezi sa zamajcem, odnosno ona je ulazni član. Da bi ovaj mehanizam imao samo jedan stepen slobode, jedan od centralnih zupčanika mora biti nepokretan, i u ovom slučaju to je zupčanik 1, što znači da je $\omega_1=0$. Zupčanici 1 i 2 su okrugli, a 3 i 4 neokrugli. Zahvaljujući promenljivom prenosnom odnosu neokruglih zupčanika, sa konstantnom ulaznom brzinom ω_c=const, postiže se promenljiva izlazna brzina koja se menja u zavisnosti od prenosnog odnosa $i_{4c}^{1}(\varphi_c)$.



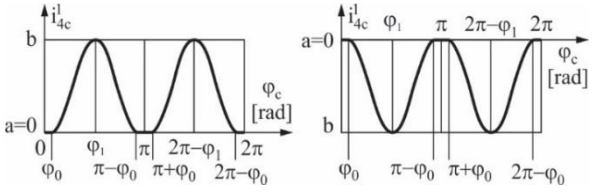
Slika 5. *Šema epicikličnog prenosnika sa neokruglim zupčanicima*

Prema Willis-u se dobija:

$$i_{4c}^{1}=\frac{\omega_4-\omega_1}{\omega_c-\omega_1}=\frac{\omega_4}{\omega_c} \tag{1}$$

$$\omega_4(\varphi_c)=i_{4c}^{1}(\varphi_c)\cdot\omega_c(\varphi_c)\ \Rightarrow\ \varphi_4(\varphi_c)=\int_0^{\varphi c} i_{4c}^{1}(\varphi_c)\,d\varphi_c \tag{2}$$

Da bi klip mirovao u gornjoj (φ_c=0) i donjoj (φ_c=π) mrtvoj tački, ugaona brzina radilice mora biti jednaka nuli u određenom intervalu oko uglova φ_c=0 i φ_c=π. Osim toga, ugona brzina radilice ne sme menjati smer. Figure 7, u skladu sa tim, prikazuje funkciju prenosnog odnosa i_{4c}^{1}.



Slika 6 *Funkcije prenosnog odnosa*

Slika 6a prikazuje prenosni odnos i_{4c}^{1} pozitivnog znaka – ugaone brzine ω_c i ω_4 su istog smera, dok je na slici 7b prenosni odnos i_{4c}^{1} negativnog znaka – ugaone brzine ω_c i ω_4 su suprotnog smera. Prema tome, opšti oblik funkcije je dat jednačinom:

$$i_{4c}^{1}(\varphi_c)=\begin{cases}\pm a,\ \varphi_c\in[0,\varphi_0)\\ \pm\frac{1}{2}(a+b)\mp\frac{1}{2}(b-a)\cos(\pi\frac{\varphi_c-\varphi_0}{\varphi_1-\varphi_0}),\ \varphi_c\in[\varphi_0,\varphi_1)\\ \pm\frac{1}{2}(a+b)\mp\frac{1}{2}(b-a)\cos(\pi\frac{\pi-\varphi_c-\varphi_0}{\varphi_1-\varphi_0}),\ \varphi_c\in[\varphi_1,\pi-\varphi_0)\\ \pm a,\ \varphi_c\in[\pi-\varphi_0,\pi)\\ i_{4c}^{1}(\varphi_c-\pi),\ \varphi_c\in[\pi,2\pi)\end{cases} \quad (3)$$

Na osnovu funkcionalnih zahteva SUS motora, za jedan obrt radilice, vratilo zamajca treba da izvrši jedan obrt, pa je:

$$\varphi_4(\varphi_c=2\pi)=2\pi\ \Rightarrow\ 2\pi=\int_0^{2\pi} i_{4c}^{1}(\varphi_c)\,d\varphi_c \quad (3)$$

Rešavanjem (4) se dobija međusobna zavisnost parametara a i b, pa je:

$$b=\frac{2-2a\frac{\varphi_0}{\varphi_1}}{1-\frac{\varphi_0}{\varphi_1}}-a \quad (4)$$

Pošto se prenosni odnos i_{4c}^{1} ne može direktno izraziti preko geometrijskih parametara mehanizma, upotrebom izraza (1) se dobija:

$$i_{4c}^{1}=1-i_{41}^{c}=1-\left(-\frac{R_3}{R_4}\right)\left(-\frac{R_1}{R_2}\right)=1-i_{43}^{c}i_{21}^{c} \quad (5)$$

Gde su i_{43}^{c} i i_{21}^{c} prenosni odnosi pri relativnom kretanju u odnosu na krivaju c.
Pri tome je:

$$i_{21}^{c}(\varphi_{1r})=-\frac{R_1}{R_2}=\frac{\omega_2-\omega_c}{\omega_1-\omega_c}=\frac{\omega_{2r}}{\omega_{1r}} \quad (6)$$

Gde je:

$$\omega_{1r}=\omega_1-\omega_c=-\omega_c \quad (7)$$

$$\omega_{2r}=-\frac{R_1}{R_2}\omega_{1r}=\frac{R_1}{R_2}\omega_c \quad (8)$$

$$\varphi_{2r}=-\frac{R_1}{R_2}\varphi_{1r}=\frac{R_1}{R_2}\varphi_c \quad (9)$$

Zupčanici 2 i 3 su kruto povezani i zbog toga je $\omega_{2r}=\omega_{3r}$ i $\varphi_{2r}=\varphi_{3r}$. Prema tome, prenosni odnos zupčastog para 3, 4 pri relativnom kretanju u odnosu na krivaju c je:

$$i_{43}^{c}(\varphi_{3r})=\left(-\frac{R_3}{R_4}\right)=\frac{\omega_4-\omega_c}{\omega_3-\omega_c}=\frac{\omega_{4r}}{\omega_{3r}} \quad (10)$$

$$\varphi_{4r}(\varphi_{3r})=\int_0^{\varphi 3r} i_{43}^{c}(\varphi_{3r})\,d\phi_{3r} \quad (11)$$

Imajući u vidu da je $\omega_{4r}=\omega_4-\omega_c$, sledi:

$$\varphi_{4r}(\varphi_c)=\varphi_4(\varphi_c)-\varphi_c \quad (12)$$

Ukoliko je $i_{4c}^{1}\geq 0$, za φ_{4r} se dobija oscilatorno promenljiva promena koja je pogodna u specifičnim situacijama [13]. Međutim, u ovom slučaju je neprihvatljiva i zbog toga se usvaja varijatna $i_{4c}^{1}\leq 0$.

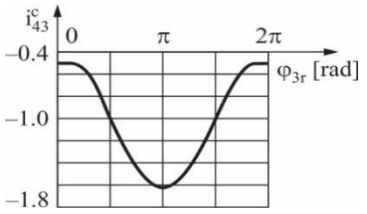
Centroide zupčanika 3 i 4 moraju biti zatvorene krive i zbog toga je:

$$\frac{T_{3r}}{n_{3r}}=\frac{T_{4r}}{n_{4r}} \quad (13)$$

where: T_{3r} and T_{4r} su periodi obrtanja neokruglih zupčanika 3 i 4 pri relativnom kretanju u odnosu na krivaju *c*, dok su n_{3r} i n_{4r} celi brojevi.
Iako je moguće više rešenja, s obzirom da je $\varphi_{4r}(\varphi_{3r})$ periodična funkcija sa periodom $\pi R_1/R_2$, tehnološki je najjednostavnije rešenje za $R_1/R_2=2$ i $n_{3r}=n_{4r}=1$. Dalje se usvaja $\varphi_1=\pi/2$ i $\varphi_0=10\pi/180$. Imajući u vidu Eq. (12), prenosni odnos i_{43}^{c} je:

$$i_{43}^{c}(\varphi_{3r})=\frac{d\varphi_{4r}}{d\varphi_{3r}} \quad (14)$$

Slika 7 prikazuje promenu prenosnog odnosa $i_{43}^{c}(\varphi_{3r})$.



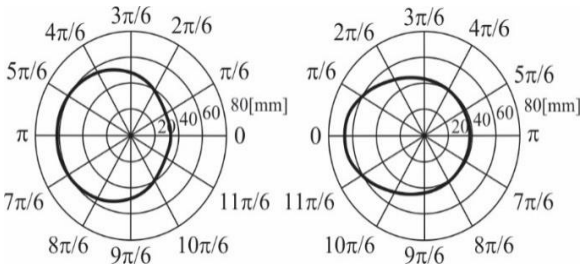
Slika 7. *Promena prenosnog odnosa* $i_{43}^{c}(\varphi_{3r})$

Jednačine centroida zupčanika 3 i 4 su:

$$R_3(\varphi_{3r})=\frac{L}{1+\left|i_{43}(\varphi_{3r})\right|}\left|i_{43}(\varphi_{3r})\right|,\ \varphi_{3r}=0...2\pi \quad (15)$$

$$R_4(\varphi_{3r})=L-r_3(\varphi_{3r}),\ \varphi_{4r}(\varphi_{3r})=\int_0^{\varphi 3r} i_{43}^{c}(\varphi_{3r})\,d\varphi_{3r},\ \varphi_{4r}=0...2\pi \quad (16)$$

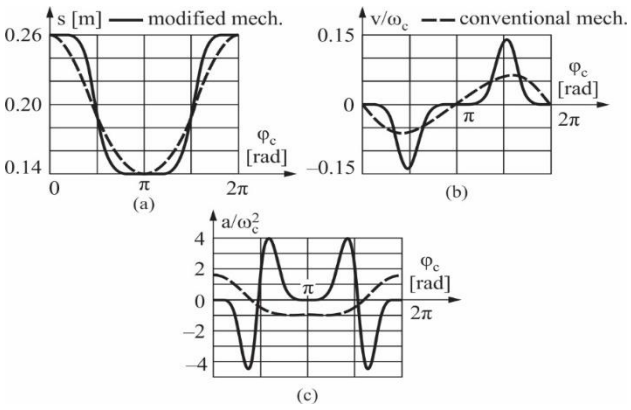
Slika 8, na osnovu Eqs. (16–17) i usvojenog medjusobnog rastojanja L=100 mm, prikazuje centroide zupčanika 3 i 4.



Slika 8 *Dobijene centroide neokruglih zupčanika*

4. SIMULACIJA KRETANJA I REZULTATI

Sprovođenjem kinematske analize modifikovanog mehanizma, dobijaju se modifikovani zakoni kretanja klipa. Položaji mrtvih tačaka i hod klipa usvojeni su na osnovu postojećih SUS motora [1], i iznose: S_{DMT}=140 mm, S_{GMT}=260 mm, h=S_{GMT}–S_{DMT}=120 mm. Slika 1a prikazuje zakon promene položaja klipa konvencionalnog I modifikovanog mehanizma. Klip modifikovanog mehanizma miruje u mrtvim tačkama, u intervalu. –10÷10° obrtanja kolenastog vratila. Slika 10b prikazje razliku u brzini klipa modifikovanog i konvencionalnog mehanizma. Ubrzanje klipa oba mehanizma prikazano je na slici 11c. Ubrzanje klipa modifikovanog mehanizma je glatka funkcija bez naglih skokova, što je povoljno. Međutim vrednosti ubrzanja su primetno veće nego kod konvencionalnog mehanizma, što implicira postojanje većih inercijalnih opterećenja, ali o tome neće biti reči u ovom radu.



Slika 9. *Zakoni kretanja klipa modifikovanog klipnog mehanizma*

5. ZAKLJUČAK

U radu je razmotrena upotreba neokruglih planetnih zupčanika radi postizanja perioda mirovanja klipa u gornjoj i donjoj mrtvoj tački. Ovakvom izmenom kinematike klipa menja se i radni ciklus motora, i tada realni radni ciklus postaje sličniji idealnom.

Mirovanje klipa u mrtvim tačkama, prvenstveno gornjoj, simulira izohorski proces dovođenja toplote prisutan u idealnom ciklusu, a kao direktnu posledicu ima povećanje efikasnosti radnog ciklusa motora. Upotrebom planetnih neokruglih zupčanika omogućen je period potpunog mirovanja uz mogućnost izrade zupčanika. Osim toga, upotrebom planetnih zupčanika je zadržana koaksijalnost između kolenastog vratila i zamajca. Ipak, nisu izbegnuti svi nedostaci.

Kinematička analiza pokazuje veće vrednosi i oštre skokove ubrzanja oko mrtvih tački, što sa sobom povlači velika inercijalna opterećenja.

Analiza dinamike mehanizma u okviru koje bi se ispitala inercijalna opterećenja je tema budućih istraživanja. Mogućnost izrade zuba zupčanika nije detaljnije ispitivana u ovom radu, ali takođe zahteva pažnju u budućnosti.

6. LITERATURA

[1] Dorić, J.Ž., Klinar, I.J.: Efficiency of a new internal combustion engine concept with variable piston motion. Therm. Sci. 18(1), 113–127 (2014).

[2] Dorić, J.Ž., Klinar, I.J.: Efficiency characteristics of a new quasi-constant volume combustion spark ignition engine. Therm. Sci. 17(1), 119–133 (2013).

[3] Pouliot, H.N., Delameter, W.R., Robinson, C.W.: A variable-displacement spark-ignition engine. SAE Technical Paper 770114(2), 446–464 (1978).

[4] ElBahloul, M.A., Aziz, E.S., Chassapis, C.: Performance study of the hypocycloid gear mechanism for internal combustion engine applications. Int. J. Engine Res. 1–17 (2019).

[5] Chironis, N.P.: Mechanisms, Linkages and Mechanical Controls. McGraw-Hill, New York (1965).

[6] Mundo, D., Danieli G.A.: Use of non-circular gears in pressing machine driving systems. IASME Trans. 1(1), 7–11 (2004).

[7] Soong, R.-C.: On the new mechanical press with a planetary gear train. Trans. Can. Soc. Mech. Eng. 40(1), 45–58 (2016).

[8] Litvin, F.L, Gonzalez-Perez, I., Fuentes, A., Hayasaka, K.: Design and investigation of gear drives with non-circular gears applied for speed variation and generation of functions. Comput. Methods Appl. Mech. Eng. 197(45–48), 3783–3802 (2008).

[9] Lovasz, E.-C., Modler, K.-H., Neumann, R., Gruescu, C.M., Perju, D., Ciupe, V., Maniu, I.: Novel design solutions for fishing reel mechanisms. Chin. J. Mech. Eng. 28(4), 726–736 (2015).

[10] Modler, K.-H., Lovasz, E.-C., Bär, G.F., Neumann, R., Perju, D., Perner, M., Mărgineanu, D.: General method for the synthesis of geared linkages with non-circular gears. Mech. Mach. Theory 44(4), 726–738 (2009).

[11] Zheng, F., Hua, L., Han, X., Li, B., Chen, D.: Synthesis of indexing mechanisms with non-circular gears. Mech. Mach. Theory 105(1), 108–128 (2016).

[12] Smelyagin, A.I., Prikhod'ko, A.A.: Structure and kinematics of a planetary converter of the rotational motion into the reciprocating rotary motion. J. Mach. Manuf. Reliab. 45(6), 500–505 (2016).

[13] Mundo, D.: Geometric design of a planetary gear train with non-circular. Mech. Mach. Theory 41(4), 456–472 (2006).

Kratka biografija:

Dijana Čavić rođena je u Novom Sadu 1996. god. Master rad na Fakultetu tehičkih nauka iz oblasti Mehanizacija i konstrukciono mašinstvo odbranila je 2020.god.

kontakt: dijana.cavic@gmail.com

Milan Rackov je rođen u Kikindi 1976. Doktorirao je 2013. na FTN i 2014. god. na MF u Bratislavi. Od 2018. je zvanju vanredni profesor. Oblast interesovanja su mašinski elementi i zupčasti prenosnici.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621
DOI: https://doi.org/10.24867/10AM07Brkic

MEHANIZAM MANIPULATORA ZA POLJOPRIVREDNE SISTEME: ANALIZA KINEMATIKE I DINAMIKE

MANIPULATOR MECHANISM FOR AGRICULTURAL SYSTEMS: A KINEMATIC AND DYNAMIC ANALYSIS

Boris Brkić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – MEHANIZACIJA I KONSTRUKCIONO MAŠINSTVO

Kratak sadržaj – *Kompleksnost ubiranja „mekih“ plodova čini automatizaciju ubiranja gotovo nemogućom. U ovom radu obrazložena je potreba za automatizacijom, dat je pregled postojećih mehanizama i prikazan je razvoj novog prostornog mehanizma. Sprovedena je kinematička, a zatim i dinamička analiza, koje su proverene upotrebom Matlab i ADAMS softvera. Na kraju su prikazani rezultati razvoja mehanizma i kasnijih analiza.*

Ključne reči: *Roboti u poljoprivredi, mehanizmi, simulacije, analiza mehanizma, prostorni mehanizmi.*

Abstract – *The complexity of harvesting ''soft“ fruits makes automation nearly impossible. This paper explains why automation is needed, reviews the existing mechanisms used in this field and shows the development of a new spatial mechanism. Kinematic analysis was carried out, followed by a dynamic analysis, which were both verified through the use of Matlab and ADAMS software. Finally, the results of the mechanism development and analysis were presented.*

Keywords: *Robots in agriculture, mechanisms, simulations, mechanism analysis, spatial mechanisms.*

1. UVOD

U dvadeset prvom veku pred poljoprivrednu proizvodnju postavljaju se mnogi zahtevi. Predviđeni rast ljudske populacije na svetskom nivou zahteva povećanje: efikasnosti proizvodnje i količine hrane. Takođe povećanje globalnih temperatura i smanjenje dostupne količine fosilnih goriva zahteva optimizaciju celog procesa. Ovaj cilj moguće je postići samo automatizacijom poljoprivredne proizvodnje.

Automatizacija procesa u poljoprivredi već je u toku širom sveta, upotreba mašina za: obradu zemljišta, setvu, zaštitu i ubiranje znatno je humanizovala rad i povećala prinose u prošlosti.

Ipak pored toga danas postoji veliki broj poslova koje i dalje obavljaju ljudi. Primer takvih poslova je ubiranje „mekih“ plodova. Ubiranje „mekih“ plodova bez njihovog oštećenja je jedan od najvećih problema koji još uvek nisu u rešeni.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Maja Čavić, vanred. prof.

Glavni razlog tome jeste raznolikost karakteristika plodova. Zbog kompleksnosti prirode problema veliki broj istraživača smatra da se ova vrsta problema može rešiti uvođenjem specijalizovanih robota.

U ovom radu biće predstavljen novi mehanizam robota sa šest stepeni slobode kretanja koji ima potencijal da bude korišćen u poljoprivrednoj proizvodnji.

Mehanizam će biti detaljno analiziran sa aspekta kinematike i dinamike: izvešće se jednačine kretanja a zatim će biti urađena detaljna kinetostatička analiza svih članova. Simulacija kretanja će se izvesti korišćenjem programa MATLAB., Formiraće se model mehanizma u programu ADAMS-u i verifikovati dobijeni rezultati.

2. STANJE U OBLASTI

Automatizacija procesa u poljoprivrednoj industriji tema je brojnih publikacija u struci. Pokazalo se da se svakom problemu mora prići sa specifičnim rešenjem koje zahteva multidisciplinarni prilaz [1].

Pre automatizacije bilo kog postupka ubiranja postavlja se veliki broj pitanja i uzima se u obzir veliki broj uticajnih faktora [1]. U gotovo svim slučajevima zbog velikog broja različitih karakteristika: plodova, biljaka i okruženja, roboti koji vrše ubiranje moraju posedovati veliki broj senzora i stepeni slobode kretanja [2].

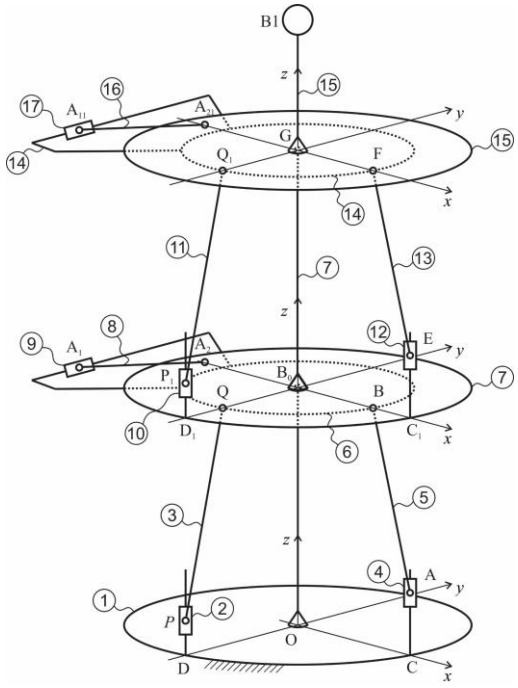
Sama kompleksnost rešenja često povlači za sobom niz nedostataka, npr. svi roboti koriste električne komponente koje su osetljive na kišu i blato, što prouzrokuje niz problema u praksi. Senzori koji su u blizini radnih organa često su najizloženiji nečistoćama i samim tim oštećenjima. Mehanička struktura mehanizama robota mora biti prilagođena težini plodova i njihovom položaju, a pri tome mora omogućavati velike brzine kretanja. Takođe struktura robota mora omogućavati postizanje određene „spretnosti“ rada mašine [3].

Pored velikog broja zahteva koji se postavljaju pred poljoprivredne robote, došlo je do naglog razvoja početkom 2000 godine. Razvijeni su roboti za ubiranje: pečuraka [4], lubenica [5], plavog patlidžana [6], pomorandži [7] i mnogih drugih biljaka. Takođe razvijena su rešenja koja opslužuju plastenike [8]. Sva rešenja su se razlikovala. Jedina konstanta koja ih povezuje jeste upotreba optičkih senzora. U postojećoj literaturi još uvek nije razvijen robot sa mehanizmom koji je obrađen u ovom radu. Ono što je specifično za mehanizam koji će se analizirati jeste upotreba sfernih zglobnih veza.

3. OPIS RADA MEHANIZMA

Na slici 1 prikazan je prostorni mehanizam, koji će biti detaljno analiziran. Član 1 predstavlja postolje sa vođicama na kojima su članovi 2 i 4 kao i kruto zavareni štap OB_0. Svi klizači (članovi: 2, 4, 9, 10, 12, 17) su pogonski i vrše translatorno kretanje po vođicama. U tački B_0, štap OB_0 sferno (sferni zglob) je vezan za član 6. Sfernim vezama u tačkama Q i B vezani su za član 6 članovi 3 i 5. Član 3 vezan je cilindričnom vezom (1 rotacija) u pravcu *x* ose za član 2. Članovi 4 i 5 sferno su vezani.

Kretanja članova 2 i 4 zadaju položaje članu 6. Član 6 na sebi nosi član 9 koji se translatorno kreće po njegovoj vođici. Članovi 6 i 7 vezani su cilindričnom vezom u tački B_0 i mogu se zakretati jedan u odnosu na drugi oko B_0G pravca. Član 8 zglobno je vezan za članove 9 i 7. Kretanje člana 9 definiše zakretanje člana 7 u odnosu na član 6.



Slika 1. *Šema prostornog mehanizma*

Član 7 ima sličnu konfiguraciju kao član 1, za njega su zavarene dve vođice po kojima se kreću članovi 10 i 12. Takođe štap B_0G zavaren je za član 7 u tački B_0. Ostatak mehanizma se ponavlja tj. veza između članova 10 i 11 je cilindrična, između 11 i 14 kao i 13 i 14 je sferna. Isto tako su veze između sledećih članova sferni zglobovi: 12 i 13, 16 i 17, 16 i 15.

Diskovi tj. članovi 14 i 15 klize jedan u odnosu na drugi kao članovi 6 i 7. Članovi 14 i 7 sferno su vezani u tački G. Položaj člana 14 u odnosu na član 7 definisan je pogonskim članovima 10 i 12, dok je rotacija člana 15 u odnosu na član 14 definisana kretanjem člana 17. Štap GB_1 zavaren je za član 15 u tački G. Svi članovi su kruti. Koordinatni sistemi: B_0xyz, $Gxyz$ čvrsto su vezani za članove 7 i 15 respektivno. Nakon svakog zakretanja ose u oznaci dobijaju po jednu zapetu (').

4. KINEMATIČKA ANALIZA

Cilj kinematičke analize je nalaženje brzine i ubrzanja tačke B_1 sa slike 1. pošto ona predstavlja hvataljku (radni organ) manipulatora.

4.1. Kretanje pogonskih članova

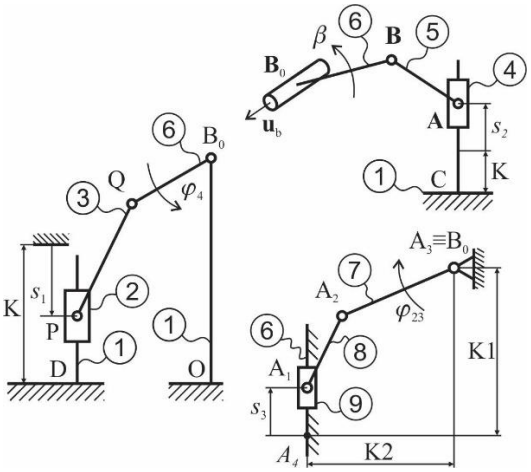
Da bi sistem tela bio mehanizam mora se kontrolisano kretati tj. mora se zadati zakon kretanja pogonskim članovima i dobiti poznato, kontrolisano kretanje izlaznog člana. U ovom slučaju koristi se polinom 3-4-5 zbog svojih povoljnih karakteristika [9]:

$$s = 10h\cdot(t/T)^3 - 15h\cdot(t/T)^4 + 6h\cdot(t/T)^5 \quad (1)$$

T predstavlja vreme trajanja kretanja, *h* je konačni pređeni put pogonskog člana, *t* je vreme. Svi pogonski članovi imaju isti zakon kretanja.

4.2. Prvi segment

Mehanizam se može podeliti na 2 segmenta. Ovi segmenti se analiziraju posebno. Prvi segment sastoji se od članova: 1, 2, 3, 4, 5, 6, 7, 8 i 9. Rešavanje zahteva razlaganje na tri mehanizma data na slici 2. Svaki mehanizam definiše jedan od tri ugla sfernog kretanja člana 7 oko tačke B_0.



Slika 2. *Tri mehanizma sa uglovima koji definišu položaj člana 7*

Pošto su dva mehanizma ravanska (sl. 2 levo i dole desno) i slična, mogu se lako rešiti pisanjem vektorskih kontura čijim projektovanjem na ose koordinatnog sistema i rešavanjem se dobijaju uglovi φ_4 i φ_{23}. Treći mehanizam je prostorni (sl. 2 gore desno). Za rešavanje prostornog mehanizma koriste se matrične metode. U ovom slučaju iz uslova da je član 5 krut dobija se traženi ugao β. Na ovaj način dobijene su jednačine koje tačno definišu položaj člana 7 u svakom trenutku vremena. Korišćenjem dobijenih uglova formiraju se matrice transformacije:

$$\left[R_{\varphi_4,x}\right]=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\varphi_4 & \sin\varphi_4\\ 0 & -\sin\varphi_4 & \cos\varphi_4\end{bmatrix},\left[R_{\beta,y'}\right]=\begin{bmatrix}\cos\beta & 0 & \sin\beta\\ 0 & 1 & 0\\ -\sin\beta & 0 & \cos\beta\end{bmatrix} \quad (2)$$

$$\left[R_{\varphi_{23}^*,z''}\right]=\begin{bmatrix}\cos\varphi_{23}^* & -\sin\varphi_{23}^* & 0\\ \sin\varphi_{23}^* & \cos\varphi_{23}^* & 0\\ 0 & 0 & 1\end{bmatrix},\quad \begin{aligned}[R1]&=\left[R_{\beta,y'}\right]\left[R_{\varphi_4,x}\right],\\ [R2]&=\left[R_{\varphi_{23}^*,z''}\right]\left[R_{\beta,y'}\right]\left[R_{\varphi_4,x}\right]\end{aligned} \quad (3)$$

gde je φ_{23}^* ugao zakretanja čl. 7 u odnosu na čl. 6.

Nakon nalaženja uglova rotacije oko odgovarajućih osa, moguće je odrediti ugaonu brzinu člana 7. Za to potrebno

je sve vektorske konture i uslove diferencirati po vremenu i rešiti. Nakon rešavanja opet se dobijaju eksplicitne jednačine, s tim da se sada određuju: $\dot{\varphi}_4$, $\dot{\varphi}_{23}$ i $\dot{\beta}$. Sada se ugaona brzina člana 7 dobija kao:

$$\vec{\omega} = \begin{bmatrix} \dot{\varphi}_4 \\ 0 \\ 0 \end{bmatrix} - \dot{\beta}\left[R_{\varphi_4,x}\right]^{-1}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \dot{\varphi}_{23}\left[R1\right]^{-1}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (4)$$

gde se matrice transformacije koriste za prebacivanje vektora iz pokretnih u nepokretni koordinatni sistem.

Ako se diferenciraju vektorske konture i uslovi još jednom po vremenu dobijaju se jednačine iz kojih se određuju: $\ddot{\varphi}_4$, $\ddot{\varphi}_{23}$ i $\ddot{\beta}$. Diferenciranjem ugaone brzine dobijaja se ugaono ubrzanje člana 7 kao:

$$\vec{\varepsilon} = \begin{bmatrix} \ddot{\varphi}_4 \\ 0 \\ 0 \end{bmatrix} - \ddot{\beta}\left[R_{\varphi_4,x}\right]^{-1}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} - \dot{\beta}\dot{\varphi}_4\left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times \left(\left[R_{\varphi_4,x}\right]^{-1}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}\right)\right) + \quad (5)$$
$$+\ddot{\varphi}_{23}\left[R1\right]^{-1}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \dot{\varphi}_{23}\left(\begin{bmatrix} \dot{\varphi}_4 \\ 0 \\ 0 \end{bmatrix} - \left[R_{\varphi_4,x}\right]^{-1}\begin{bmatrix} 0 \\ \dot{\beta} \\ 0 \end{bmatrix}\right) \times \left(\left[R1\right]^{-1}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right)$$

Brzina i ubrzanje tačke G nalaze se kao:

$$\vec{v}_G = \vec{\omega} \times \vec{r}_G^{B_0} \quad (6)$$

$$\vec{a}_G = \vec{\varepsilon} \times \vec{r}_G^{B_0} + \vec{\omega} \times \left(\vec{\omega} \times \vec{r}_G^{B_0}\right) \quad (7)$$

gde je:

$$\vec{r}_G^{B_0} = \left[R2\right]^{-1}\begin{bmatrix} 0 \\ 0 \\ \overline{B_0G} \end{bmatrix} \quad (8)$$

Ovime je završena analiza prvog segmenta.

4.3. Drugi segment

Detaljnijom analizom može se uočiti da je drugi segment isti kao i prvi. Zato se sada ceo proračun ponavlja na isti način. Dobijenim veličinama pridružuje se broj 1 u indeksima i oznakama. Dobijene jednačine definisane su za pokretni koordinatni sistem B_0xyz. Najbitnije veličine dobijaju sledeće oznake:

$$\begin{array}{l} s_1 \to s_4 \\ s_2 \to s_5 \\ s_3 \to s_6 \end{array} \quad \begin{array}{l} \vec{\omega} \to \vec{\omega}_1 \\ \vec{\varepsilon} \to \vec{\varepsilon}_1 \end{array} \quad \vec{r}_G^{B_0} \to \vec{r}_{B1}^{G} \quad (9)$$

Brzina i ubrzanje vrha mehanizma - tačke B_1 je:

$$\begin{aligned} \vec{v}_{B1} &= \vec{v}_G + \vec{\omega}_{B1} \times \vec{r}_{B1}^{G} \\ \vec{a}_{B1} &= \vec{a}_G + \vec{\varepsilon}_{B1} \times \vec{r}_{B1}^{G} + \vec{\omega}_{B1} \times \left(\vec{\omega}_{B1} \times \vec{r}_{B1}^{G}\right) \end{aligned} \quad (10)$$

gde su:

$$\begin{aligned} \vec{\omega}_{B1} &= \vec{\omega} + \left[R2_1\right]^{-1}\vec{\omega}_1 \\ \vec{\varepsilon}_{B1} &= \vec{\varepsilon} + \left[R2_1\right]^{-1}\vec{\varepsilon}_1 \end{aligned} \quad (11)$$

Time je analiza drugog segmenta i kinematička analiza celog mehanizma završena.

4. KINETOSTATIČKA ANALIZA

Cilj kinetostatičke analize je dobijanje pogonskih sila u svim pogonskim članovima kao i sila koje deluju na pojedine članove pod dejstvom opterećenja tokom kretanja mehanizma.

U tački B_1 kruto se veže kugla mase *m =10kg*, momenata inercije u svim pravcima *J*. Kugla u ovom slučaju simulira težinu ubranog ploda i hvataljke. Na ceo mehanizam deluje sila težine kugle kao i inercijalne sile koje su posledica kretanja kugle:

$$\begin{aligned} \vec{G} &= m \cdot g \\ \vec{F}^{in} &= -m \cdot \vec{a}_{B1} \\ \vec{M}^{in} &= -J \cdot \vec{\varepsilon}_{B1} \end{aligned} \quad (12)$$

Analiza počinje od vrha mehanizma i sprovodi se do svakog pogonskog čl. Korišćenjem drugog i trećeg Njutnovog zakona kao i Dalamberovog principa, za svaki član pišu se jednačine za glavni vektor svih sila i glavni moment svih sila za jednu proizvoljnu tačku. Koristi se metod dekompozicije. Pošto je u pitanju prostorni mehanizam sve jednačine se projektuju na tri ose jednog pokretnog ili nepokretnog koordinatnog sistema. Rešavanje ovih sistema radi se isključivo matričnom metodom. Pošto često nije moguće rešiti jednačine pojedinih članova posebno, rešavaju se zajedno u sistemima jednačina.

Na kraju kinetostatičke analize dobijene su reakcije veza u pokretnim i nepokretnim koordinatnim sistemima, čime se omogućava dimenzionisanje svih članova mehanizma. Takođe dobijaju se potrebne pogonske sile na svim pogonskim članovima što omogućuje dimenzionisanje, ili odabir pogonskih motora.Ova analiza prevazilazi okvire ovog rada jer se ne može svesti na par jednačina kao kinematička analiza pa neće biti prikazana.

5. SIMULACIJE

Na osnovu jednačina koje su izvedene u kinematičkoj i kinetostatičkoj analizi izvršiće se simulacija kretanja u MATLABu. Članovima mehanizma zadaju se konačne dimenzije. Takođe, radi verifikacije, formira se model mehanizma i vrše se simulacije kretanja u programu ADAMS.

Za svaki pokret mehanizma dobijaju se vrednosti: položaja, brzina i ubrzanja svih tačaka mehanizma. Isto tako dobijaju se sve sile koje deluju na sve članove mehanizma. Simulira se pet pokreta. Prvi pokret dobija se kada je mehanizam u vertikalnom položaju i zadaju se kretanja članova 9 i 17. Drugi pokret dobija se zadavanjem kretanja članova 4 i 12 tako da se donji disk zakrene za 20 stepeni, a gornji za 40 stepeni. Treći pokret počinje iz krajnjeg položaja drugog pokreta. Mehanizam se zatim kreće u vertikalni položaj koristeći iste članove a pogon. Kroz vertikalni položaj prolazi i nastavlja da se naginje na drugu stranu dok diskovi ne dobiju uglove naginjanja od -20 i -40 step. Četvrti pokret predstavlja kombinaciju prvog i drugog. Peti pokret dobija se kretanjem svih pogonskih članova istovremeno. Svaki pokret traje jednu sekundu, broj koraka u Adamsu i matlabu je 100.

6. REZULTATI I DISKUSIJA

Pri pretpostavci da su svi članovi a posebno diskovi tj. članovi 6, 7, 14 i 15 laki, rezultati dobijeni u oba programa poklapaju se za sve pokrete. Pretpostavka je opravdana jer su mase pojedinih članova, pogotovo upotrebom modernih materijala, značajno manje od mase hvataljke sa plodom. Nakon zadavanja dužina svih članova:

$$\overline{OD} = \overline{OC} = \overline{B_0D_1} = \overline{B_0C_1} = 150mm$$
$$\overline{OB_0} = \overline{B_0G} = \overline{GB1} = 500mm$$
$$\overline{QB_0} = \overline{B_0B} = \overline{Q_1G} = \overline{GF} = 100mm$$
$$\overline{PQ} = \overline{AB} = \overline{P_1Q_1} = \overline{EF} = 300mm$$
$$K = 204,196mm \quad K1 = 300mm \quad K2 = 100mm$$
$$\overline{B_0A_2} = \overline{GA_{12}} = 100mm \quad \overline{A_1A_2} = \overline{A_{11}A_{21}} = 300mm$$

I nakon zadavanja hoda za kretanje svih pogonskih članova kao:

$\underline{1.\ pokret} \rightarrow s_1 = 0,\ s_2 = 0,\ s_3 = 34,263mm,$
$s_4 = 0,\ s_5 = 0,\ s_6 = 34,263mm$
$\underline{2.\ pokret} \rightarrow s_1 = 0,\ s_2 = -33,119mm,\ s_3 = 0,$
$s_4 = 0,\ s_5 = -33,119mm,\ s_6 = 0$
$\underline{3.\ pokret} \rightarrow s_1 = 0,\ s_2 = -68,404mm,\ s_3 = 0,$
$s_4 = 0,\ s_5 = -68,404mm,\ s_6 = 0$
$\underline{4.\ pokret} \rightarrow s_1 = 0,\ s_2 = -33,119mm,\ s_3 = 34,263mm,$
$s_4 = 0,\ s_5 = -33,119mm,\ s_6 = 34,263mm$
$\underline{5.\ pokret} \rightarrow s_1 = -33,119mm,\ s_2 = 33,454mm,\ s_3 = 34,263mm,$
$s_4 = 33,119mm,\ s_5 = 33,454mm,\ s_6 = 34,263mm$

Dobija se da su:

- sve sile u razumnim granicama,

-najveća brzina tačke B1 iznosi 1991mm/s i postiže se u p 3. Pokretu,

-najveće ubrzanje iznosi 6309 mm/s^2 i postiže se takođe u 3. Pokretu.

-najveća pogonska sila pojavljuje se u čl. 4 i iznosi 1286 N, dok se sa dodavanjem masa povećava na 1434 N.

Da bi znali koliki je uticaj masa koje su zanemarene, u ADAMS-u formiran je model sa masama i momentima inercije za navedene članove. Rezultati pokazuju da dodavanjem masa i momenata inercije najviše rastu sile na donjim pogonskim članovima. Za dodate mase od po 2,5 kg na centre diskova (ćl. 6, 7, 14 i 15) uz masu od 10 kg u tački B_1 i dodavanjem momenata inercije članovima 6 i 14: $J_x = 2,812 \cdot 10^4 kg \cdot mm^2$, $J_y = 1,4 \cdot 10^4 kg \cdot mm^2$ i članovima 7 i 15 kao: $J_x = 1,25 \cdot 10^4 kg \cdot mm^2$, $J_y = 6251 kg \cdot mm^2$ uz to da je: $J = 10^6 kg \cdot mm^2$ dobija se da je najveći porast pogonske sile na članu 9 u petom pokretu. U tom slučaju sila raste za 17,4% u nekim trenutcima tokom kretanja.

7. ZAKLJUČAK

Analiziran je prostorni mehanizam manipulatora sa šest stepeni slobode kretanja. Jednačine koje opisuju kretanje manipulatora i opterećenja svih članova su izvedene što potvrđuju simulacije u programskom paketu ADAMS. Tokom analize pokazalo se da je drugi segment veoma sličan prvom, što znači da se drugi segment može nadograditi na vrh mehanizma *n* puta, što omogućava formiranje manipulatora sa 3*n* stepeni slobode kretanja. Ovako formiran mehanizam imao bi značajno veću pokretljivost i radni prostor. Jednačine koje su dobijene mogu da se i u tom slučaju koriste. Takođe, dobijene jednačine mogu da se koriste za optimizaciju mehanizma. Matrice rotacije i zakretanja značajno olakšavaju rešavanje problema gde postoji veliki broj pokretnih koordinatnih sistema. Primećeni porast pogonske sile nakon dodavanja inercijalnih karakteristika diskovima ukazuje na to da kod formiranja mehanizma dodavanjem više uzastopnih drugih segmenata dolazi do sve veće računske greške pogonske sile jer formirane jednačine ne uzimaju u obzir mase svih elemenata.

8. LITERATURA

[1] C. Schertz, G. Brown, "Basic considerations in mechaniying citrus harvest", In proc. Annual meeting of the american society of agricultural engineers, 343-346. Amherst, june (1966).

[2] N. Tillett, "Robotic manipulators in horticulture: a review", Agricultural engineering research 55(2): 89–105 (1993).

[3] B. Sivaraman and F. Burks, "Robot manipulator for citrus harvesting: configuration selection", In proc. ASABE annual international meeting, paper number: 071144, Minneapolis, June 17-20 (2007).

[4] N. Reed, J. Miles, J. Butler, M. Baldwin, R. Noble, "Automatic mushroom harvester development", Agricultural engineering research 78(1): 15–23 (2001).

[5] Y. Edan, D. Rogozin, T. Flash, E. Miles, "Robotic melon harvesting", IEEE transactions on Robotics and Automation 16(6): 831–835 (2000).

[6] S. Hayashi, K. Ganno, Y. Ishii, I. Tanaka, "Robotic Harvesting System for Eggplants", Japan Agricultural Research Quarterly: JARQ 36(3) 163–168 (2002).

[7] G. Muscato, M. Prestifilippo, N. Abbate, I. Rizzuto, "A prototype of an orange picking robot: past history, the new robot and experimental results", Industrial Robot: An International Journal 32(2): 128–138 (2005).

[8] J. Van Henten, J. Hemming, J. van Tuijl, G. Kornet, J. Meuleman, J. Bontsema, A. van Os, "An autonomous robot for harvesting cucumbers in greenhouses", Autonomous robots 13(3): 241–258 (2002).

[9] R. Norton, "Design of machinery: an introduction to the synthesis and analysis of mechanisms and machines", Worcester polytechnic institute, Massachusetts (1999).

Kratka biografija:

Boris Brkić rođen je u Vrbasu 1993. god. Master rad na Fakultetu tehičkih nauka iz oblasti Mehanizacija i konstrukciono mašinstvo odbranio je 2020.god.

kontakt: borisbrki93@gmail.com

Maja Čavić je rođena u Beogradu 1965. Doktorirala je 2012. na FTN. Od 2017. je zvanju vanredni profesor. Oblast interesovanja su teorija mehanizama i mašina i prenos snage i kretanja.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK 620.193/.199
DOI: https://doi.org/10.24867/10AM08Ponjevic

METALNI BIOMATERIJALI - KOROZIJA, ISPITIVANJE, PREVENCIJA I MONITORING

METALLIC BIOMATERIALS - CORROSION, TESTING, PREVENTION AND MONITORING

Nikola Ponjević, Dragan Rajnović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – PROIZVODNO MAŠINSTVO

Kratak sadržaj – *U ovom radu su detaljno opisane osobine i mikrostruktura metalnih biomaterijala sa akcentom na koroziji ovih materijala, njihovim ispitivanjem, prevencijom i monitoringom. Ispitivane su tri legure: nerđajući čelik 316LVM, titanijumova legura Ti-6Al-4V i kobaltova legura Co-Cr-Mo. U eksperimentalnom delu uzorci su potopljeni u rastvor soli i kiseline, te su nakon toga posmatrani na SEM i svetlosnom mikroskopu. Utvrđeno je da su sva tri materijala otporna u većoj meri na koroziju u rastvoru kiselina ili slanom rastvoru.*

Ključne reči: *Korozija, Biomaterijali, Ispitivanje*

Abstract – *This paper describes in details the properties and the microstructure of metallic biomaterials with emphasis on the corrosion of these materials, their testing, prevention and monitoring. Three alloys were tested: 316LVM stainless steel, Ti-6Al-4V titanium alloy and Co-Cr-Mo cobalt alloy. In the experimental part, the samples were immersed in a solution of salt and acid and then observed under the SEM and under the light microscope. It was found that all 3 materials are generally resistant to corrosion in solution of acids or salts.*

Key words: *Corrosion, Biomaterials, Testing*

1. UVOD

1.1 Metalni biomaterijali

Osnovni zahtev za biomaterijale je da moraju da poseduju odgovarajuće karakteristike, čak i pri radu u veoma zahtevnom okruženju, kakvo je ljudsko telo, odnosno da poseduju odgovarajuću biokompatibilnost. Biokompatibilnost je opisni izraz kojim se označava sposobnost materijala da se ponaša na odgovarajući način u sredini (živom tkivu) u kojem se koristi [1].

Metali poseduju odgovarajuće osobine, kao što su velika tvrdoća, duktilnost, otpornost na lom, čvrstoća, otpornost na koroziju, mogućnost oblikovanja i biokompatibilnost, neophodne za podnošenje opterećenja prilikom fiksiranja fraktura parcijalne i totalne artroplastike zglobova. Tri legure metala koje se najviše koriste u medicini su [2,3]:

- legure nerđajućeg čelika.
- legure na bazi titanijuma, i
- legure na bazi kobalta.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dragan Rajnović, docent.

2. KOROZIJA

Korozija se definiše kao degradacija materijala u sastavne atome pod uticajem hemijske reakcije koja se dešava između materijala i njihove okoline [5].

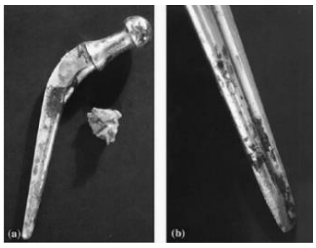
2.1 Korozija biokompatibilnih metala

Bioimplantati bi trebalo da poseduju veoma visoku otpornost na koroziju i habanje zbog izloženosti visoko korozionom okruženju unutar tela i različitim intenzitetima opterećenja. Pored ovoga, bioimplantati treba da budu otporni i na zamor i lom [6].

2.2 Vrste korozija u metalnim implantatima

Korozija metala se dešava kao rezultat interakcije između elektrohemijskih ćelija što vodi raznim vrstama korozionih reakcija [5]. Neke od najčešćih vrsta korozije navode se dalje u tekstu.

1) Ravnomerna ili uniformna korozija se dešava kada je čitava površina metala izložena katodičnim reagentima tokom lokalizovane korozije.
2) Piting korozija označava određenu vrstu lokalizovane korozije. Piting se javlja u obliku tačaka ili udubljenja na površini.
3) Tribokorozija je mehanizam trošenja pri kojem preovladavaju hemijske ili elektrohemijske reakcije materijala s okolinom.
4) Korozioni zamor je rezultat kombinovanog delovanja promenljivih napona i korozionog okruženja.
5) Naponska korozija se odnosi na razgradnju i/ili stvaranje produkata korozije određene površine metala ili legura u elektrolitu, zbog toga što je metal konstantno podvrgnut različitim naprezanjima.
6) Galvanska korozija se javlja u spoju dva metala od kojih metal sa pozitivnijim potencijalom predstavlja katodno, a metal sa negativnijim potencijalom predstavlja anodno mesto.
7) Korozija u zazoru se odigrava usled brzog utroška oksidacionog sredstva u maloj zapremini zazora.



Slika 1. *(a) Površinska korozija i (b) piting korozija nakon uklanjanja implantata [5]*

2.3 Efekti korozije na implantat

Kako materijal kreće da korodira, rastvaranje metala dovodi do erozije koja zatim dovodi do loma implantata [6]. Kada dođe do loma materijala, korozioni proces se ubrzava usled povećanog površinskog sloja i gubitka zaštitne oksidne prevlake. Ukoliko se metalni fragmenti ne uklone hirurški, moguće je daljnje rastvaranje i fragmentacija koji mogu dovesti do zapaljenja okolnog tkiva.

2.4 Efekti korozije na ljudsko telo

Korodirani implantati u ljudskom telu dovode do oslobađanja veoma štetnih i toksičnih jona u telesne tečnosti. Joni metala koji se oslobađaju iz implantata su prepoznati kao vodeći razlog kliničkih problema i alergijskih reakcija [5]. Oslobađanje jona može izazvati neželjene posledice po zdravlje, pa čak i uzrokovati smrt pacijenta.

2.5 Korozija konvencionalnih biolegura

Kod nerđajućeg čelika manje prisustvo ugljenika smanjuje verovatnoću stvaranja hrom karbida što dovodi do integranularne korozije. Ipak, nerđajući čelik je podložan lokalizovanoj koroziji usled delovanja hloridnih jona i redukovanih jedinjenja sumpora. Prisustvo mikro-organizama na metalnim površinama često dovodi do usko lokalizovanih oštećenja u koncentraciji elektrolitnih sastojaka, nivoa pH i kiseonika.

Upotreba titanijuma i njegovih legura je postala sveprisutna usled njegove čvrstoće, niskog modula elastičnosti, manje gustine i dobre kombinacije mehaničke i izvanredne korozione otpornosti [6]. Titanijum je inertan i otporan na koroziju usled izlaganja telesnim tečnostima i tkivu što ga čini potpuno biokompatibilnim, ali je takođe utvrđeno da mogu izazvati dugotrajne medicinske probleme kao što su periferna neuropatija, osteomalacija i Alchajmerova bolest usled ispuštanja jona aluminijuma i vanadijuma iz legure.

Kobaltove legure poseduju bolju čvrstoću, modul elastičnosti, otpornost na abraziju i koroziju u poređenju sa legurama nerđajućeg čelika. Kao i kod nerđajućeg čelika, hrom je u ovim legurama odgovoran za najveći deo njihove otpornosti na koroziju. Međutim, u ovim legurama, kobalt takođe ima veliki uticaj, što ih čini izuzetno otpornim na koroziju. Biokorozija u ovim legurama je jedan od glavnih problema pošto ona dovodi do ispuštanja jona metala koji izazivaju neželjena dejstva.

2.6 Monitoring (praćenje) korozije u biomaterijalima

Korozija u biomaterijalima se smatra elektrohemijskom reakcijom. Na bazi toga, većina tehnika za nadgledanje procesa degradacije koristi elektrohemijski pristup [7]. Elektrohemijska istraživanja su se uvek sprovodila u simuliranim telesnim tečnostima kao što su aerovana morska voda, Ringerov rastvor, Henkov rastvor i fosfatni pufer. Tehnike koje su korišćene uključuju potenciostatičku i potenciodinamičku polarizaciju, kao i spektroskopiju elektrohemijske impedanse, dok se količina jona merila koristeći uređaje za atomsku apsorpcionu spektroskopiju i atomsku emisionu spektroskopiju. Tri tehnike koje su razvijene i upotrebljavaju se za dobijanje detaljnih kvantitativnih informacija o elektrohemijskom ponašanju metalnih biomaterijala su skening elektrohemijska mikroskopija (SECM), testiranje pomoću elektrohemijskog skrečinga pri velikim brzinama i spektroskopiju impedanse sa postepenom polarizacijom.

2.7 Prevencija korozije biomaterijala

Preventivne mere za korozione sisteme uključuju: dizajn, selekciju materijala, površinske modifikacije (inhibitore i prevlake), modifikacije materijala (legiranje, termička obrada), kao i elektrohemijske tehnike (katodna i anodna zaštita) [7].

3. EKSPERIMENTALNA PROCEDURA

3.1 Ispitivani materijali

Ispitivanje pojave korozije je izvršeno na sledećim uzorcima polaznog materijala:

- 316LVM – šipka u vučenom stanju
- Ti-6Al-4V – šipka u kovanom stanju
- Co-Cr-Mo – šipka u livenom stanju

Tabela 1. *Uporedni prikaz standardnih hemijskih sastava ispitivanih materijala (mass. %) [8, 9, 10]*

Hemijski elementi	Legura		
	316LVM	Ti-6Al-4V	Liveni Co-Cr-Mo
Cr (%)	17.5	-	26.5 -30
Mo (%)	2.8	-	4.5 - 7.0
Ni (%)	14	-	max. 1.0
Fe (%)	osnova	max. 0.25	max. 1.0
Al (%)	-	5.5-6.5	-
V (%)	-	3.5-4.5	-
C (%)	≤0.025	max. 0.08	-

3.2 Priprema metalografskih uzoraka

Uzorci za ispitivanje mikrostrukture su isečeni iz polaznog šipkastog materijala, i potom pripremljeni standardnom metalografskom tehnikom pripreme koja se sastojala od montiranja, brušenja, poliranja i nagrizanja uzoraka.

3.3 Izlaganje uzoraka korozionoj sredini

Izlaganje uzoraka korozionoj sredini je izvršeno potapanjem uzoraka u 2 različita rastvora:

- Slani rastvor - Ringerov rastvor [11]

  Sastav: Destilovana voda + 8.60 gL^{-1} NaCl + 0.30 gL^{-1} KCl + 0.33 gL^{-1} $CaCl_2$

- Alkoholni rastvor kiselina [12]

  Sastav: Alkohol + 11.5% H_2SO_4 + 1.2% HCl + 1% $CuCl_2$ + 1% $FeCl_3$

Potapanje u oba rastvora je izvršeno u vremenu od 96 časova, na sobnoj temperaturi.

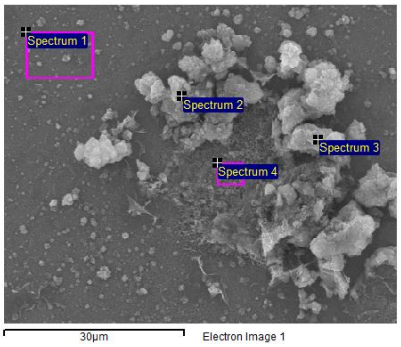
Uzorci su nakon potapanja u rastvore posmatrani na svetlosnom mikroskopu Leitz Orthoplan i SEM mikroskopu JEOL JSM 6460 LV sa EDS sistemom Inca, Oxford Instruments u cilju određivanja hemijskog sastava produkata korozije.

4. REZULTATI I DISKUSIJA

4.1 Nerđajući čelik 316LVM

Mikrostruktura nerđajućeg čelika 316LVM je u potpunosti austenitna sa jasno vidljivim deformacionim dvojnicima u poprečnom preseku. Veličina zrna na poprečnom preseku odgovara indeksu veličine zrna G=7.

Izgled produkata korozije izloženih dejstvu rastvora soli koji su ispitivani EDS analizom prikazan je na slici 2, dok je hemijski sastav produkata dat u tabeli 2. Produkti korozije su prvenstveno železni oksidi. Pored produkata korozije, prisutni su i tragovi slanog rastvora, u vidu elemenata Na, Ca, i K. Korozija se javlja na mestu smanjene koncentracije hroma (spec. 4).



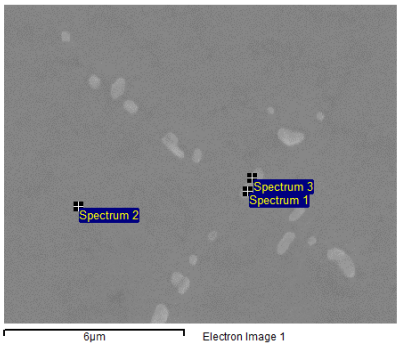
Slika 2. *Mesto EDS analize na leguri nerđajućeg čelika izloženog rastvoru soli*

Tabela 2. *Hemijski sastav produkata sa slike 2 [mass. %]*

Spec.	O	Na	Si	S	K	Ca	Cr	Mn	Fe	Ni	Mo
1	3.6	0.1	0.4	0.1	0.05	0.07	15.5	1.7	62.1	13.3	2.6
2	26.2	0.3	0.5	-	0.03	0.2	3.9	0.6	65.2	1.9	0.7
3	32.5	0.7	0.7	-	0.04	0.2	2.2	0.6	62	0.6	0.1
4	20.3	0.03	0.4	-	0.03	0.1	10.5	1.4	56.8	8.1	2.1

Izgled produkata korozije izloženih dejstvu rastvora kiseline koji su ispitivani EDS analizom prikazan je na slici 3, dok je hemijski sastav produkata dat u tabeli 3. Na površini dolazi do stvaranja sitnih zrnaca nastalih prilikom rastvaranja osnove sa rastvorom kiselina. U zrncima je prisutan natrijum (spec. 1 i 3), dok na neoštećenoj površini, osnovi, nema prisustva natrijuma (spec. 2).

Osim taloga natrijuma, nisu uočena druga koroziona oštećenja.



Slika 3. *Mesto EDS analize na leguri nerđajućeg čelika izloženog rastvoru kiseline*

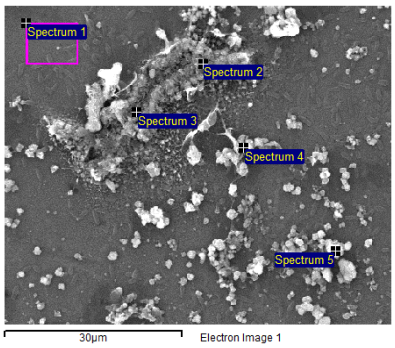
Tabela 3. *Hemijski sastav produkata sa slike 3 [mass. %]*

Spec.	C	Na	Al	Si	Cr	Mn	Fe	Ni	Mo
1	1.9	0.6	0.2	0.4	16.7	2.2	60.1	14.4	3
2	2.6	-	0.1	0.4	16.3	1.9	62.2	13.6	2.5
3	2.3	0.4	0.1	0.4	16.6	2	60.7	14.4	2.7

4.2 Legura titanijuma Ti-6Al-4V

Mikrostruktura Ti-6Al-4V legure se sastoji od krupne β faze (tamna) ravnomerno raspoređene u osnovi - α faze (svetla osnova). Veličina β faze iznosi oko 10 µm.
Takođe, prisutna je i mala količina nemetalnih uključaka, koji su slični klasi D (oksidno globularni), veličine 1, fini, prema ASTM standardu.

Produkti korozije usled dejstva rastvora soli, ispitivani EDS analizom, prikazani su na slici 4, dok je hemijski sastav produkata dat u tabeli 4. Nakon izlaganja uzoraka u rastvoru soli došlo je do korozije na mestu železnih uključaka (nečistoća), pri čemu nastaju železni oksidi (spec. 2 do 5).

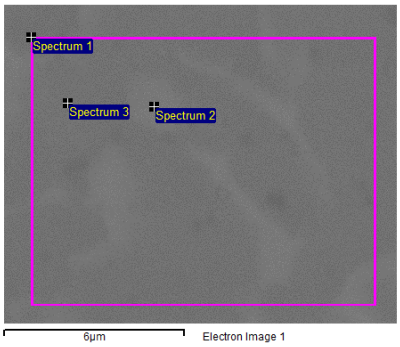


Slika 4. *Mesto EDS analize na leguri titanijuma izloženog rastvoru soli*

Tabela 4. *Hemijski sastav produkata sa slike 4 [mass. %]*

Spec.	O	Na	Al	Si	Cl	Ca	Ti	V	Fe
1	-	-	5.4	-	-	-	88.6	4.7	1.2
2	45.5	0.2	0.7	0.6	0.15	0.2	19.8	0.7	31.9
3	46.6	0.49	0.6	0.7	0.15	0.2	17.2	0.6	33.4
4	36.2	0.5	2.1	0.3	0.03	0.1	46.9	1.3	12.2
5	14.4	0.08	3.6	0.1	0.00	1.1	73.9	3.3	3.4

Prilikom potapanja legure titanijuma u rastvoru kiselina dolazi do delomičnog nagrizanja površine i uočavanja β-faze u α-osnovi, slika 5. Oštećenje površine kiselinom je ravnomerno bez nastanka posebnih produkata korozije na površini. EDS hemijski sastav površine nije detektovao značajnu razliku u hemijskom sastavu, tabela 5.



Slika 5. *Mesto EDS analize na leguri titanijuma izloženog rastvoru kiseline*

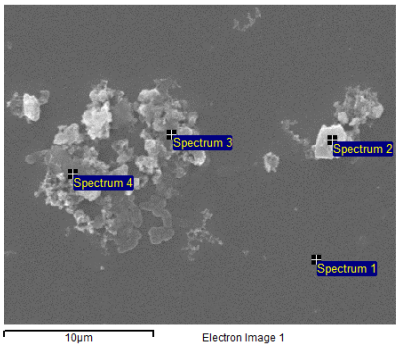
Tabela 5. *Hemijski sastav produkata sa slike 5 [mass. %]*

Spec.	Al	Ti	V	Fe
1	5.01	89.60	5.08	0.31
2	4.57	89.51	5.67	0.25
3	4.67	88.31	6.75	0.27

4.3 Legura kobalta Co-Cr-Mo

Mikrostruktura se sastoji od svetle osnove u kojoj se nalaze σ faza (tamna polja) i karbidi (svetla polja). U mikrostrukturi je prisutna i velika količina pora, nastalih verovatno prilikom skupljanja tokom livenja u međudendritskom području. Veliko prisustvo poroznosti može uticati na smanjenje mehaničkih osobina, posebno otpornosti na zamorni lom.
Izgled produkata korozije izloženih dejstvu rastvora soli koji su ispitivani EDS analizom prikazan je na slici 6, a hemijski sastav produkata dat je u tabeli 6. Pored konta-

minacije površine železnim oksidom u vidu mreže, prisutne su i mešovite čestice železnog oksida (spec. 2) i taloga soli (spec. 2-4). Nije uočeno dubinsko oštećenje površine, već su produkti prisutni na površini, slika 6.

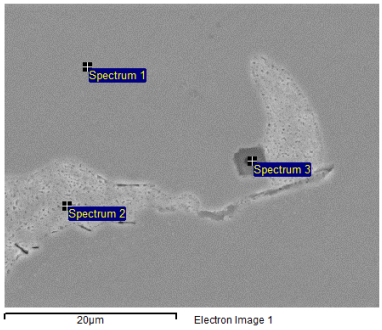


Slika 6. *Mesto EDS analize na leguri kobalta izloženog rastvoru soli*

Tabela 6. *Hemijski sastav produkata sa slike 6 [mass. %]*

Spec.	O	Na	Mg	Si	Cl	Ca	Cr	Mn	Fe	Co	Mo
1	-	-	-	0.9	-	-	27	0.7	-	65.5	5.8
2	15.9	0.7	0.3	1.2	0.3	0.3	18	0.5	17.6	40.2	4.6
3	7.1	0.5	1.1	1.8	-	0.1	23.3	0.5	2.8	57.9	4.4
4	4	-	-	1.2	-	-	25	0.5	2.2	62.7	4.2

Izgled površine legure kobalta Co-Cr-Mo nakon dejstva rastvora kiseline prikazan je na slici 7, dok je hemijski sastav uočenih mikrokonstituenata dat u tabeli 7. U mikrostrukturi osnove (spec. 1), prisutna je dendritska σ faza (spec. 2) bogata sa hromom i molibdenom, kao i mali uključak titanijuma (spec. 3). Dejstvo kiseline utiče ravnomerno na celu površinu, pri čemu je σ faza osetljivija i počinje da se nagriza.



Slika 7. *Mesto EDS analize na leguri kobalta izloženog rastvoru kiseline*

Tabela 7. *Hemijski sastav produkata sa slike 7 [mass. %]*

Sp.	Al	Si	Ti	Cr	Mn	Co	Mo
1	-	0.8	-	27.9	0.78	64.9	5.6
2	-	0.9	0.2	38.7	1.06	43.5	15.6
3	1.07	0.4	40.4	15.6	0.48	34.5	7.4

5. ZAKLJUČAK

Na osnovu rezultata ispitivanja ustanovljeno je da su svi materijali otporni u većoj meri na koroziju u rastvoru soli ili kiselina. Legura Ti-6Al-4V ima najveću otpornost u rastvoru kiselina, dok se kod čelika 316LVM stvara sitni talog, a kod legure Co-Cr-Mo dolazi do nagrizanja mikrostrukture (σ faza). S druge strane, prilikom potapanja u slani rastvor, došlo je do onečišćenja rastvora železnim česticama, i potom njihovog taloženja i korzije na površini uzoraka. U ovom slučaju potrebno je obratiti posebnu pažnju na čistoću površine materijala prilikom ugradnje u ljudsko telo, ne samo sa aspekta mikrobiološke čistoće, već i sa stajališta nežive materije (nalepljene nečistoće).

6. LITERATURA

[1] J. Black, G. Hastings, *"Handbook of Biomaterial Properties"*, London UK: Chapman and Hall, 1998.

[2] B.D. Ratner, A. S. Hoffman, F. J. Schoen, J. E. Lemons, *"Biomaterials Science An Introduction to Materials in Medicine"*, Academic Press, San Diego, 1996.

[3] J.B. Park, R.S. Lakes, *"Biomaterials An Introduction"*, Plenum Press, New York, 1992.

[4] D. M. Brunette, P. Tengvall, M. Textor, P.Thomsen, *"Titanium in Medicine"*, Springer, Berlin, 2001.

[5] R.I.M. Asri, W.S.W. Harun, M. Samykano, N.A.C. Lah, S.A.C. Ghani, F. Tarlochan, M.R. Raza, "Corrosion and surface modification on biocompatible metals: A review", *Materials Science and Engineering: C*, Vol. 77, pp. 1261-1274, 2017.

[6] G. Manivasagam, D. Dhinasekaran, A. Rajamanickam, "Biomedical implants: corrosion and its prevention-a review", *Recent Pat. Corros. Sci.*, Vol. 2, pp. 40–54, 2010.

[7] O.O. Ige, L.E. Umoru, M.O. Adeoye, A.R. Adetunji, O.E. Olorunniwo, and I.I. Akomolafe, "Monitoring, Control and Prevention Practices of Biomaterials Corrosion – An Overview", *Trends in Biomaterials and Artificial Organs*, Vol. 23 (2), pp. 93-104, 2009.

[8] D.J. Medlin, R. Compton, "Metallography of Biomedical Orthopedic Alloys", *Metallography and Microstructures*, Vol 9, ASM Handbook, ASM International, pp. 961–968, 2004.

[9] https://www.materials.sandvik/en/materials-center/material-datasheets/bar-anhollow-bar/bar/sandvik-316lvm/ (pristupljeno u julu 2020.)

[10] A. Marti, *"Cobalt-base alloys used in bone surgery, Injury"*, 31, pp. 18-21, 2000.

[11] N. Eliaz, *"Corrosion of Metallic Biomaterials: A Review"*, Department of Materials Science and Engineering, Tel-Aviv University, Ramat Aviv 6997801, Israel, 2019.

[12] M. Robinson, *"Encyclopedia of Materials: Science and Technology"*, HAVAR®, A Co–Cr Biocompatible Alloy for Medical Implants, 2005.

Kratka biografija:

Nikola Ponjević, mast. inž. maš., rođen je u Novom Sadu 1994.g. Gimnaziju „Svetozar Marković" je završio 2013.g. u Novom Sadu, Master rad na Fakultetu tehničkih nauka iz oblasti Proizvodno mašinstvo odbranio je 2020.g.

Dr Dragan Rajnović, docent, rođen je 1975.g. u Vukovaru. Diplomirao je na Fakultetu tehničkih nauka 2000.g., na kome je od tada i zaposlen u nastavi. U zvanje docenta izabran je 2015.g., za uno: Materijali i tehnologije spajanja. Specifična oblast interesovanja: mikroskopija i ispitivanje materijala.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.9
DOI: https://doi.org/10.24867/10AM09Radulovic

PROJEKTOVANJE I IZRADA ALATA ZA INJEKCIONO PRESOVANJE PLASTIKE

DESIGN AND MOLD MAKING FOR PLASTIC INJECTION MOLDING

Rajan Radulović, Milenko Sekulić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – MAŠINSTVO

Kratak sadržaj – *Cilj rada jeste da se ukaže na značaj savremenih vidova obrade u izradi alata za brizganje plastike uz podršku programskog sistema PTC Creo Parametric. Praktičan cilj rada je tehnološki postupak izrade alata za brizganje plastike za konkretan proizvod i proračun izrade konkretnog alata.*

Ključne reči: *savremeni vidovi obrade, brizganje plastike, PTC Creo Parametric.*

Abstract – *The aim of this paper is to point out the importance of modern types of processing in the development of plastic injection tools with hte support of the software system PTC Creo Parametric. The practical goal of this paper is the tehnological process of making plastic injection tools for a specific product and the calculation of making a specific tool.*

Keywords: *modern types of processing, plastic injection, PTC Creo Parametric.*

1. UVOD

Brizganje plastike je najvažniji ciklični proces prerade polimera. Alat za brizganje predstavlja specifični i središnji deo sistema za brizganje polimera. Sama izrada alata čini najveći deo troškova u neto ceni gotovog alata. U cilju smanjenja ovih troškova, alatničari danas koriste produktivnije postupke obrade (npr. visokobrzinsko glodanje), nego što je to klasična EDM sa punom elektrodom. U ovom radu je pokazana mogućnost kombinovanja konvencionalnih i nekonvencionalnih postupaka prilikom izrade kalupa, a u cilju optmizacije troškova njihove izrade i dobijanja traženog kvaliteta obrađene površine alata.

2. OSNOVE BRIZGANJA PLASTIKE

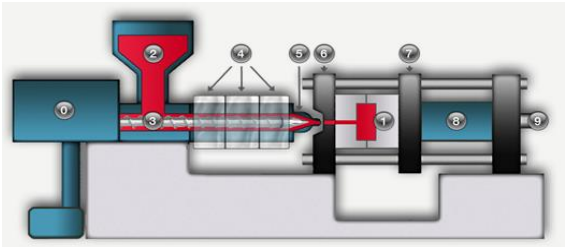
2.1 Istorija brizganja plastike

Braća Hyatt su 1872. napravili mašinu koja je omogućavala da se npr. drvena jegra četkice za sapunanje pri brijanju oblože slojem prirodnog plastomera, celuloznog nitrata.
Bio je to početak razvoja brizganja. Veliki preokret u razvoju brizgalica je bilo otkriće pužnog vijka 1950-ih. Razvijen je 1952. i patentiran 1956.

2.2 Brizganje plastike

Jedna od najćešćih metoda prerade plastičnih materijala je proces brizganja.

Kompletna oprema za brizganje plastike sastoji se od: mašine za brizganje plastike, alata za oblikovanje, uređaja za temperiranje [1]. Za brizganje polimernih rastopa potrebna je linija za brizganje plastike. Čine je tehnološki nužna oprema i dopunska oprema [2].



Slika .1 Mašina za brizganje plastike: 0-hidraulični motor; 1–alat za brizganje; 2–levak za dopunjavanje materijala; 3–puž; 4–grejači; 5–mlaznica; 6–nepokretna ploča; 7–pokretna ploča; 8–jedinica za zatvaranje kalupa; 9–izbacivačka jedinica.

Najvažniji parametri kod brizganja plastike su: pritisak ubrizgavanja, hidraulični pritisak u jedinici za ubrizgavanje, pritisak u kalupnoj šupljini, naknadni pritisak, temperatura rastopa, temperatura zida kalupne šupljine, brzina (protok) sredstva za temperiranje idr.

3.0 OSNOVE ZA DEFINISANJE ALATA

3.1 Zahtevi porizvoda

Za optimizaciju karakteristika dela potrebno je uskladiti mehaničke, dimenzione i estetske parametre.

3.1.1 Mehanički parametri

3.1.1.1 Zidovi

Ujendačena debljina zidova će minimizirati naprezanja, vreme hlađenja, razlike u skupljanju, moguće stvaranje šupljina i usaklina na površini

3.1.1.2 Radijusi

Unutrašnji oštri uglovi ili veoma mali radijusi, kao i zarezi, su jedan od glavnih uzroka neuspeha pri brizganju plastičnih proizvoda. Ako je moguće, radijusi treba da budu 0,6 mm ili više da bi se sprečila koncentracija napona između proizvoda i kalupa.

3.1.1.3 Zakošenja zidova

Na proizvodu moraju biti definisani uglovi. Oni treba da su minimalno 1° kod poliranih površina, a više od 3° kod hrapavijih površina.

3.1.1.4 Linije spajanja

Linije spajanja su slabe oblasti formirane ujedinjavanje tokova plastike, koje mogu dovesti do mehaničkog kvara proizvoda ukoliko nije dobro dizajniran.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Milenko Sekulić, red.prof.

3.1.1.5 Ulivni sistem

Optimalni dizajn ulivnog sistema će smanjiti rastojanje protoka istopljene plastike u šupljinu, smanjiti probleme popunjavanja kalupa idr.

3.1.2 Kritičnost dimenzija

3.1.2.1 Skupljanje

U proizvodnim procesima materijali koji prolaze kroz faze promena (od tečne faze do čvrste), uključuju smanjenje specifične zapremine i rezultuju skupljanju.

3.1.2.2 Tolerancije

Usko polje tolerancija može značajno povećati troškove brizganja delova jer može da doda nove korake od projektovanja pa do njegove proizvodnje.

3.2 Zahtevi alata

Pre izrade alata, konstruktor mora odrediti broj jezgara, vrstu čelika, upotrebu toplih dizni, položaj i broj izbacivača, kanale za hlađenje, kao i niz drugih osnovnih parametara alata. Pošto je potrebna temperatura u kalupu najmanje 120°C, potreban je dovoljan broj kanala za hlađenje. Preporučuje se prečnik kanala za hlađenje 10 mm.

4. PROJEKTOVANJE ALATA ZA BRIZGANJE PLASTIKE SA NJEGOVIM PODSISTEMOM

4.1 Savremeni razvoj proizvoda

U svim granama industrije nekoliko faktora dovelo je promena u preduzeću pre svega su to: malo vreme isporuke proizvoda, velika potražnja za unikatnim proizvodima, konkurisanje sličnim proizodima pristupačnih cena i kvaliteta. U cilju kraćeg ciklusa brizganja treba ići ka tome da rad alata bude što više automatizovan. To zahteva uvođenje CIM strategije vođenja preduzeća. CIM je skraćenica nastala od engleske reči (Computer Integrated Manufacturing), što bi u prevodu značilo računarom integrisana proizvodnja.

4.2 Predmet rada - Razdeljivač trafoa pod visokim naponom

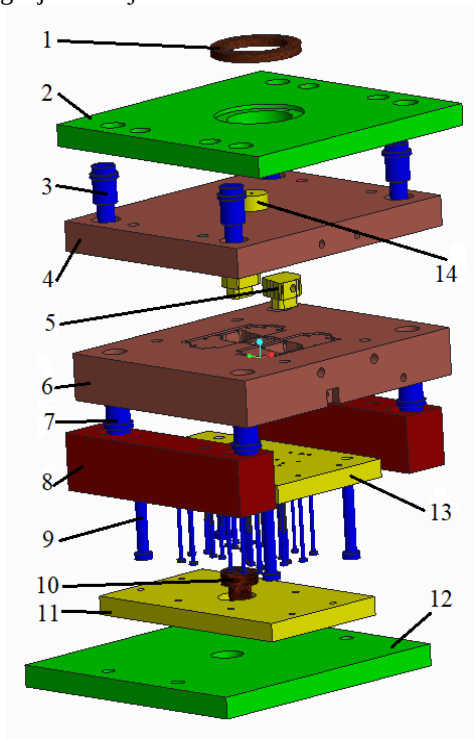
Razdeljivač trafoa je jedan od proizvoda koji se koristi da ne bi došlo do kontakta dva trafoa koji se nalaze jedan pored drugog i koriste se u trofaznom brojilu.

4.3 Primena PTC Creo Parametric 3.0 u procesu projekotvanja alata za brizganje plastike

Primena obuhvata realizaciju svih faza projektovanja (definisanje koncepta, projekotvanja u užem smislu, analize i verifikacije) kao i pripremu proizvodnje kako delova tako i podsklopova koji čine poizvod (definisanje tehnološkog procesa izrade, programiranje NUMA, optimizacija i verifikacija programa, izrada tehnološke dokumentacije).

4.4 Projektovanje celokupnog alata za brizganje plastike

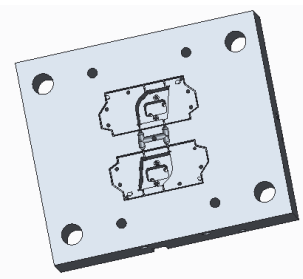
Za projektovanje alata za brizganje plastike uključen je tim inženjera, kako bi kao rezultat dobili pouzdan alat u što kraćem vremenu, naravno sa pristupačnom cenom. U nastavku se konkretno projektuje alat razdeljivača trafoa sa svim svojim specifičnostima. Kao optimalno rešenje usvojeno je da alat bude sa dve kalupne šupljine, zbog zahteva kupca koji je to tražio i njegovog obima proizvodnje. Na slici 2. dat je 3D prikaz celokupnog alata za brizganje razdeljivača trafoa.



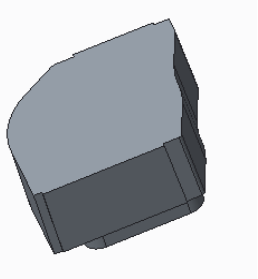
Slika 2 3D prikaz alata za brizganje razdeljivača: 1–centrirajući prsten, 2–gornja temeljna ploča, 3-vođice 4–noseća nepokretna ploča, 5–segmenti, 6–noseća pokretna ploča, 7–čaure, 8–odstojne letve, 9–izbacivači i povratnici, 10-čaura za izbacivanje, 11–izbacivačka ploča, 12–donja temeljna ploča, 13–ploča nosača izbacivača, 14–ulivna čaura

4.4.1 Projektovanje pokretne ploče

Alat za brizganje razdeljivača trafoa je primer sa segmenti koji su bili komplikovani za izradu u celini pokretne ploče. Projektovanje alata je vršeno uz pomoću softvera PTC Creo Parametric 3.0. Plastomer koji se koristi za brizganje ovog predmeta je PC, sa skupljanjem nakon hlađenja 0,5-0,7 %. Zbog toga je neophodno da sve dimenzije kalupne šupljine budu uvećane za tu vrednost kako bi se dobio gotov deo zahtevanih dimenzija. Ugao zida, zavisi pre svega od njegove visine i prosečno iznosi 0,5-2 %, i to za unutrašnje i spoljne površine.



Slika 3. Pokretna ploča



Slika 4. Segment

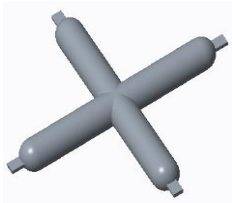
4.4.1.1 Materijal za izradu pokretne i nepokretne ploče

Za materijal od koga je izrađena pokretna i nepokretna ploča, izabran je čelik 40CrMnMo7) (Č2311). To je često primenjiv alatni čelik.

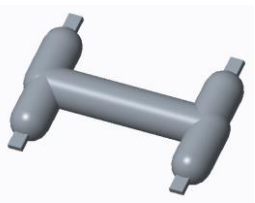
4.4.2 Projetovanje ostalih elemenata

4.4.2.1 Elementi ulivnog sistema

Osnovni elementi ulivnog sistema su sledeći: dovodni kanal, razvodni kanal i ulivni kanal. U samoj izradi alata prvobitni razvodni kanal je bio jednostavnog oblika, kao što se može videti na slici 4., ali je on ostavljao tragove ulivanja na samom gotovom proizvodu. Zbog toga je oblik ovog razvodnog kanala morao biti promenjen, s obzirom da gotov proizvod mora biti providan i poliran do punog sjaja, novi izgled razvodnog kanala dat je na slici 6.



Slika 5. Prvobitni razvodni kanal



Slika 6. Konačni razvodni kanal

4.4.2.2 Sistem za izbacivanje otpreska

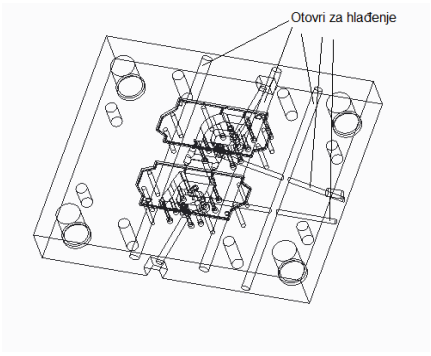
U izradi konkretnog alata korišteni su izbacivači ravne igle i profilni izbacivači odnosno igle sa ravnom stranicom (D igle).

4.4.2.3 Sistem za vođenje i centriranje alata

Alati sa paralelnim vođicama je najrasprostranjenije konstruktivno rešenje za vođenje. Najčešće se primenjuju četiri stuba za vođenje. Ovakve vođice su korištene za vođenje konkretnog alata koji je rađen u ovom master radu.

4.4.2.4 Elementi za temperiranje alata

Elementi za temperiranje alata postoje kod većine alata za brizganje plastike, kako bi se smanjio ciklus brizganja i proizvodnja bila ekonomičnija, potrebno je taj prenos ubrzati.



Slika 7. Kanali za hlađenje u pokretnoj ploči

4.4.2.5 Elementi kućišta alata

Svaki alat predstavlja jedinstvenu celinu, koja se sastoji od više elemenata kao što su ulivni sistem, izbacivački sistem, sistem za temepriranje alata, kalupna šupljina i sistem vođenja alata.

4.5 Standardizacija elemenata

Standardizacija elemenata alata za brizganje plastike je korisna, s čime se može smanjiti vreme isporuke gotovog alata, kao i cena koštanja istog. Standardni delovi konkretno alata su temeljne ploče, pokretna i nepokretna ploče (kokile), ploča nosača izbacivača, izbacivačka ploča, izbacivači, vodeći stubovi i čaure, odstojne letve, ulivna čaura.

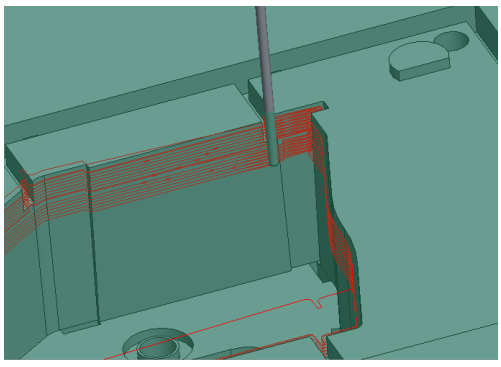
5. PROJEKTOVANJE TEHNOLOGIJE IZRADE ALATA ZA KONKRETAN PROIZVOD OD PLASTIKE

5.1 Postavka osnovnog tehnološkog postupka alata za brizganje

Tehnološku dokumentaciju je potrebno izraditi za sve delove alata koji se izrađuju u sopstvenom proizvodnom pogonu. Preduzeća koja posluju sa većim serijama proizvodnje alata, dužna su da tehnološku dokumentaciju izrade na način kako sam taj tip proizvodnje zahteva, dakle što detaljnije, za razliku od alatnica koje se bave izradom alata u pojedinačnoj proizvodnji kao što je ova. U izradi pokretne ploče alata korišćena je kombinacija konvencionalnih i nekonvencionalnih postupaka obrade. Kako se klasičnom EDM obradom može postići kvalitet obrađene površine N6, to je predviđen dodatak od 0,05 mm za naknadnu mehaničku obradu površina šupljina i ulivnih kanala, u cilju dostizanja zahtevanog kvaliteta N4, to se izvodi postupkom poliranja. Tehnološki postupak montaže u ovom primeru alata nije potreban jer se radi o pojedinačnoj proizvodnji alata i montiranje vrše obučeni radnici.

5.2 Tehnologija konvencionalnih postupaka obrade

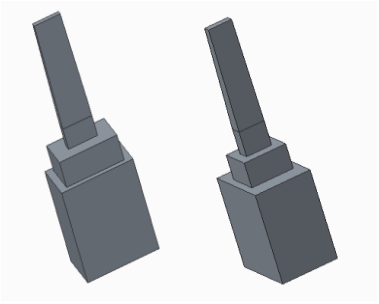
Konvencionalni postupci obrade, odnosno postupci brade rezanjem, kako se drugačije mogu nazvati, spadaju u postupke skidanja materijala rezanjem. Neke od tih obrada su bušenje, glodanje, brušenje i druge. Glodanje je postupak kojim se obrađuju najčešće ravne površine obimnim ili čeonim glodanjem te profilno glodanje idr. Glodanje se danas to vrši na savremenim CNC glodalicama, numerički upravljanim, te se slobodno može nazvati i programskim glodanjem. Kod izrade konkretnog alata korištena je CNC glodalica MAHO MH700W. Prednost je izbegavanje ručnog pisanja programa na upravljačkoj jedinici mašine, što bi beskrajno trajalo.



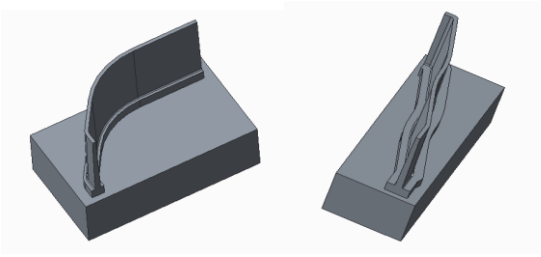
Slika 8. Linije prolaza glodala na obratku

5.3 Tehnologija nekonvencionalnih postupaka obrade

Elektroerozivna obrada (EDM) ploče pokretne forme odvija se nakon termičke obrade čelika. Za ovu vrstu obrade korištena je mašina MULTIFORM DECKEL DE-25. Kao najekonomičnija varijanta je usvojeno da se erodiranje šupljine vrši sa četiri elektrode za fino erodiranje (E1, E2, E3, E4). Zahtevani kvalitet površina šupljine je N6, što je kvalitet erodiranja, tako da je potrebna dodatna obrada poliranjem ili slično.



a) E1 b) E2



c) E3 d) E4

Slika 9. Prikaz elektroda za erodiranje šupljine u pokretnoj kokili

Potrebno je elektrode izraditi u dvostrukoj tačnosti u odnosu na traženu tačnost obratka nakon obrade. Materijal elektroda je elektrolitički bakar čistoće 99,99%, što obezbeđuje stabilan rad u širim dijapazonima obrade.

5.3.1 Tehnološki parametri obrade

Tehnološke parametre podešavanja generatora i mašine potrebno je pažljivo izabrati za pravilno odvijanje postupka elektroerozivne obrade. Oni se mogu usvojiti iz tabela, dijagrama proizvođača mašina, kao i kroz gotove programe u upravljačkoj jedinici. Elektroda E1 služi za erodiranje otvora za flahaste izbacivače u kalupnoj šupljini. Dok elektrode E2, E3 i E4 služe za erodiranje teško dostupnih površina za glodanje pa su napravljene elektrode da bi se dobila tačnost dimenzija koje su zahtevane. Za završnu obradu parametri su usvojeni takođe iz tabela, prema kombinaciji materijala i kvalitetu obrade.

6. PRORAČUN TROŠKOVA IZRADE ALATA ZA BRIZGANJE PLASTIKE

6.1 Troškovi konsturisanja

Troškovi konstruisanja se određuju na temelju sledećih parametara: broj radnih sati konstuisanja n_{rki},cena radnog sata za pojedinu fazu konstruisanja C_{rki} i ostali troškovi konstruisanja T_{KONS}. Iz toga proizilazi da osnovni trošak konstruisanja alata za brizganje razdeljivača trafoa iznosi [3]: $T_{KON} = (\sum n_{rki} \cdot C_{rki}) + T_{KONS} = (82 \cdot 1.800,00 + 40 \cdot 1.200,00) + 0 = 148.840,00[din]$

6.2 Troškovi izrade kalupa

Prilikom konstruisanja kalupa su korišteni standardni elementi gde god je to bilo moguće. Svi standardni delovi su od poizvođača Meusburger. Umeci su nestandardni delovi i moraju napraviti. Ukupna cena materijala, koji se koristi za izradu segmenata, centrirajućeg prstena i čaure za izbacivanje, iznosi $T_{om} = 15.000,00 [din]$ Broj sati za obrade n_{izi} iznosi: 175 h. Prosečna cena radnog sata operatera na mašini, iznosi $C_{izi} = 2.400,00 [din]$. Troškovi nabavke standardnih delova iznose $T_{sd} = 212\ 950 [din]$

Ukupna cena obrade kalupa iznosi:

$T_{iz} = \sum n_{izi} \cdot C_{izi} = 175 \cdot 2.400,00 = 420.000,00 [din]$

Ostali troškovi izrade kalupa, u koje ulaze nepredviđeni troškovi dorade ili ispravljanja grešaka tokom izrade kalupa: $T_{KOS} = 20.000,00 [din]$

Ukupan trošak izrade kalupa: $T_{IZ\_UK} = T_{om} + T_{iz} + T_{sd} + T_{KOS} = 15.000,00 + 420.000,00 + 212.950,00 + 20.000,00 = 667.\ 950,00 [din]$

Na osnovu prethodno date analize, dobija se fabrička cena koštanja kalupa, koja se sastoji od troškova konstruisanja kalupa (alata) i troškova njegove izrade $T_{UK} = T_{KON} + T_{IZ\_UK} = 148.\ 840,00\ din + 667.950,\ 00din = 816.790,00\ din$

Naime u firmi "Vertex" iz Rume, fabrička cena ovog kalupa se dobija prostim proračunom tako što se nabavna cena materijala pomnoži sa koeficijentom 4, tako da bi ovaj alat, prema tom proračunu koštao 911.800,00 dinara. Razlika između ove cene i cene dobijene detaljnim proračunom troškova iznosi +11,16 %.

7. ZAKLJUČAK

Osnovni cilj ovog rada se odnosio na mogućnost povećanja efikasnosti rada u delu tehnološke pripreme proizvodnje i u delu same proizvodnje alata za brizganje plastike u preduzeću "Vertex". Rezultati sprovedenih istraživanja mogu se podeliti u dve osnovne celine. Prva celina rada se bavi konkretnim projektovanjem alata za izradu razdeljivača trafoa. Drugu celinu rada čini detaljan prikaz tehnologije izrade alata, na osnovu prethodno izrađene tehničke dokumentcije. Poseban doprinos ovog rada predstavlja proračun troškova izrade gotovog alata za brizganje plastike. Brojni testovi pokazuju da procene troškova mogu varirati više od 20 % za iste uslove. Zbog toga moderne informacione tehnologije moraju biti korišćene u svakoj fazi procene troškova.

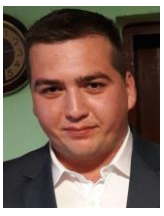
8. LITERATURA

[1] Vilotić, D.,: Mašine za injekciono presovanje, Fakultet tehničkih nauka, Novi Sad, 2008.

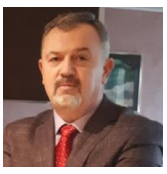
[2] Rogić, A., Čatić, I.: Injekcijsko prešanje polimera, Društvo plastičara i gumaraca, Zagreb, 1996.

[3] Vučić, S.: Metodičko konstruiranje kalupa za injekcijsko prešanje kućišta električnog sklopa, Diplomski rad, Fakultet strojarstva i brodogradnje, Zagreb, 2011.

Kratka biografija:



Rajan Radulović rođen je u Vukovaru 1994. godine. Master rad na Fakultetu tehničkih nauka iz oblasti Mašinstva odbranio je 2020.godine.



Milenko Sekulić rođen je u Prijepolju 1966. Doktorirao je na Fakultetu tehničkih nauka 2007. god., a od 2017. je zvanju redovnog profesora. Uža naučna oblast su mu procesi obrade skidanjem materijala.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.31
DOI: https://doi.org/10.24867/10BE01Petrovic

ANALIZA MATLAB SIMULINK MODELA ELEKTRIČNOG VOZILA U USLOVIMA URBANE VOŽNJE

ANALYSIS OF MATLAB SIMULINK MODEL OF ELECTRIC VEHICLE IN URBAN DRIVING CONDITIONS

Anđelija Petrović, Dejan Jerkan, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je razvijen dinamički model električnog vozila koje kao pogon koristi mašinu jednosmerne struje (DC mašina). U modelu je mašina napajana iz Li-Ion baterije, posredstvom odgovarajućeg pretvarača energetske elektronike, koji omogućuje rad pogona u dva kvadranta karakteristike napon-struja armaturnog kola. Performanse modela, kao i dinamika same baterije su analizirani na odgovarajućem test ciklusu vožnje za urbane sredine, tzv. NEDC (New European Driving Cycle).*

Ključne reči: *DC mašina, MATLAB Simulink, Li-Ion baterije, Ciklus vožnje, Električna vozila*

Abstract *The paper presents a dynamic model of an electric vehicle that uses a direct current machine (DC machine) as a drive. In the model, the machine is powered by a Li-Ion battery, by means of a suitable power electronics converter, which enables the operation of the drive in two quadrants of the voltage-current characteristic of the armature circuit. The performance of the model, as well as the dynamics of the battery itself were analyzed on the appropriate test driving cycle for urban environments, the so-called NEDC (New European Driving Cycle).*

Keywords: *DC machine, MATLAB Simulink, Li-Ion batteries, Driving Cycle, Electric vehicles*

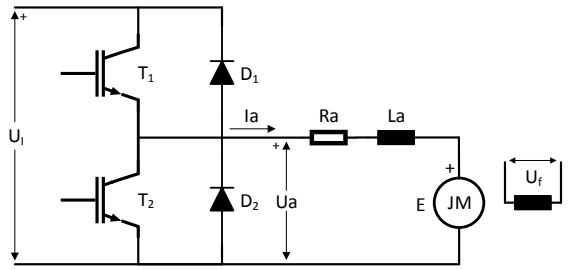
1. MAŠINE JEDNOSMERNE STRUJE

Mašine jednosmerne struje (DC mašine) spadaju u rotacione električne mašine kod kojih se vrši dvosmerno pretvaranje električne u mehaničku energiju, po pravilu posredstvom rotacionog kretanja. Najčešća izvedba koja pronalazi primenu u savremenoj tehnici jeste ona u kojoj mašina ima nezavisan pobudni namotaj, koji omogućava raspregnuto upravljanje pobudnim magnetnim poljem od regulacije napona i struja armaturnog kola. Upravo se iz tih razloga pogoni sa DC mašinama smatraju za one u kojima se mogu postići regulacija brzine i ostvarenog momenta sa visokim performansama, što se i nameće kao imperativ u primeni električnih mašina za pogon električnih vozila.

NAPOMENA:
Ovaj rad proistekao je iz master rada, čiji mentor je bio dr Dejan Jerkan, docent.

1.1. Upravljački sistem motora i energetski pretvarač

Upravljački sklop modelovanog pogona implementira kaskadnu regulaciju brzine DC mašine, odnosno struje rotorskog kola, kojom se direktno utiče na vrednost ostvarenog momenta na vratilu. Postizanje željene vrednosti struje je omogućeno realizovanjem napajanja rotorskog kola mašine jednosmernim naponom promenljive srednje vrednosti. U tu svrhu upotrebljen je DC-DC pretvarač energetske elektronike, čoper.
Regulaciona struktura pogona postavlja referencu željenog napona armature, koji se pomoću čopera ostvaruje odgovarajućim vrednostima faktora ispune, odnosno stanjem prekidačkih komponenti, koje su najčešće tranzistorskog tipa. Topologijom pretvarača, odnosno brojem i rasporedom prekidačkih komponenti postiže se rad u jednom ili više kvadranata karakteristike napon-struja. U radu se primenjuje dvokvadrantni čoper tipa C, koji omogućava rad u dva kvadranta, sa promenljivim smerom struje armaturnog kola DC mašine.

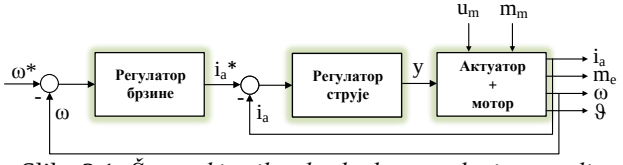


Slika 1.1. *Topologija dvokvadrantnog čopera*

Prolaskom struje kroz nultu vrednost ne dolazi do njenog prekidanja, već do promene smera proticanja. Usled toga, ne dolazi do rada u oblasti diskontinualne struje [2]. Rad u preostala dva kvadranta može se ostvariti prevezivanjem krajeva rotora ili pobude.

2. REGULACIONA STRUKTURA POGONA

Kaskadna regulacija predstavlja opšte usvojen princip regulacije u pogonima sa jednosmernom strujom, sa osnovnom idejom prikazanom na slici 2.1.



Slika 2.1. *Šematski prikaz kaskadne regulacione petlje*

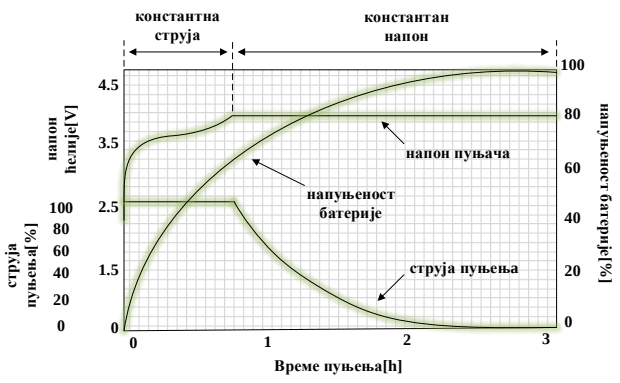
Regulacioni sistem sastoji se od dve regulacione petlje, poređane na određen hijerarhijski način [1]. Unutrašnja petlja, na nižem nivou, sa negativnom povratnom vezom struje motora i regulatorom struje, deluje direktno na aktuator i motor, regulišući struju uzimajući za referencu izlaz nadređenog regulatora.
Na sledećem, nadređenom nivou, nalazi se petlja sa povratnom vezom brzine i regulatorom brzine, čiji izlaz igra ulogu referentne vrednosti za podređenu, unutrašnju, strujnu petlju.
Iako je reč o ugnježdenim strukturama, to ne dovodi do nestabilnog rada regulacione petlje, budući da je reč o regulaciji fizičkih veličina sa veoma velikom razlikom u dinamici, gde se mehanička brzina osetno sporije uspostavlja i menja, nego što je to slučaj sa vrednostima armaturne struje.

3. BATERIJE ZA ELEKTRIČNA VOZILA

Električna vozila predstavljaju autonomni električno-mehanički sistem, te kao takva zahtevaju prenosivi izvor energije dovoljnog kapaciteta. Baterije u vidu hemijskih izvora električne energije upravo predstavljaju odgovarajuće rešenje za ovu namenu.
Postoje razne izvedbe električnih baterija, shodno tehnologiji i hemijskim elementima na kojima temelje svoj princip rada. U savremenim izvedbama električnih vozila vrlo čestu primenu imaju litujum-jonske (Li-Ion) baterije, koje predstavljaju jednu od vrsta punjivih baterija. Li-Ion baterije karakteriše visoka specifična gustina energije po jedinici mase, tako da za isti kapacitet predstavljaju manji teret za električno vozilo od drugih tipova baterija [3].

3.1. Procedura punjenja baterija

Zarad postizanja što veće autonomije električnog vozila, neophodno je na pravilan način upravljati punjenjem baterije, kako bi se u propisanom vremenskom intervalu ovim postupkom iskoristio puni kapacitet koji baterija poseduje.
Jedan od značajnijih zahteva koji se stavlja pred proizvođače baterija, ali i proizvođače punjača za iste, jeste mogućnost takozvanog brzog punjenja. Brzim punjenjem se minimizuje vreme za koje električno vozilo mora biti u mirovanju.
Procedura brzog punjenja obuhvata sledeće korake, prikazane slikom 3.1. Tokom prve faze punjenja, punjač kroz priključke baterije uspostavlja konstantnu struju, sve dok ona za posledicu ima porast napona ćelija.



Slika 3.1. *Procedura punjenja*

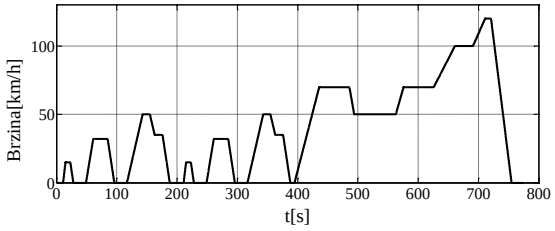
Tokom faze balansiranja, punjač postepeno smanjuje vrednost struje punjenja pri konstantnom naponu punjenja, dok se stanje napunjenosti ćelija dovodi na isti nivo pomoću balansirajućeg kola, sve dok se nivoi napunjenosti svih ćelija baterije ne uravnoteže u unapred definisanim granicama.
Kapacitet, odnosno stanje napunjenosti baterije neposredno utiču i na domet električnog vozila, koji podrazumeva prosečnu dužinu puta koji se može preći između dva ciklusa punjenja.
Domet vozila sa jednim punjenjem baterije u praksi zavisi i od broja i vrste korišćenih baterija, njihove starosti (trenutni kapacitet spram punog kapaciteta novih baterija), težine i tipa vozila, kao i samog terena i dinamike vožnje. Li-Ion baterijski sistemi za električna vozila u proseku omogućavaju domet od 320-480 km.

4. SIMULACIJE NA RAZVIJENOM MODELU ELEKTRIČNOG VOZILA

Na osnovu izloženih principa kaskadne regulacije, topologije pretvarača i izabrane vrste baterije, načinjen je simulacioni model pogona električnog vozila u MATLAB Simulink okruženju. Model podržava dinamičke simulacije kojima se u određenim aspektima verodostojno oponašaju fizički sistemi, te kao takav predstavlja dobro polazište za analizu ponašanja dinamičkih sistema, poput sistema električnih vozila.

4.1. NEDC ciklus vožnje

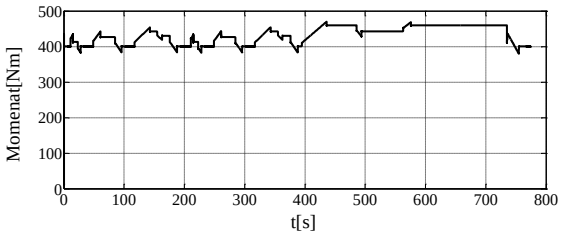
Za potrebe testiranja modela je iskorišćen standardizovani ciklus vožnje, kojim se pokrivaju karakteristični profili vožnje koji se imaju u urbanim sredinama. Na slici 3.2. prikazan je izgled NEDC (*New European Driving Cycle*) voznog ciklusa. Izvršena je simulacija sa redukovanim, odnosno sa NEDC ciklusom skraćenog trajanja. NEDC ciklus vožnje ima zadatak da reprezentuje tipične uslove za vožnju u velikim gradovima u kojima je saobraćaj gust. Pošto je namenjen vožnji po ravnom putu, ne očekuje se veliko opterećenje.



Slika 4.1. *Profil redukovanog NEDC ciklusa vožnje*

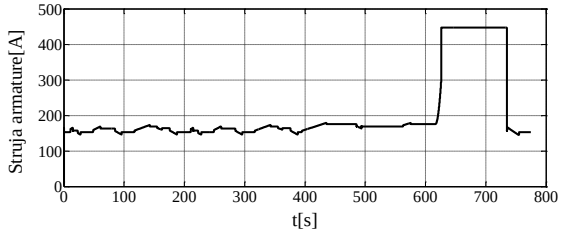
4.2. Rezultati simulacija

Na slici 4.2. prikazan je odziv momenta pogonske DC mašine tokom simulacije pogona u uslovima redukovanog NEDC ciklusa vožnje. Jasno se uočavaju vremenski intervali u kojima dinamički prelazi između stacionarnih vrednosti momenta odgovaraju zahtevima za usporavanjem, odnosno ubrzavanjem električnog vozila.



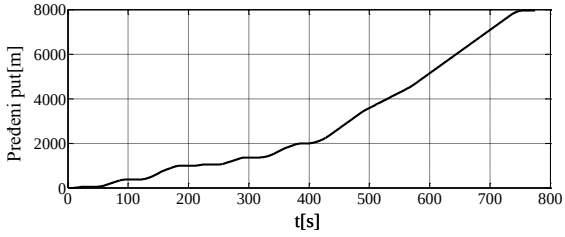
Slika 4.2. *Momenat na vratilu pogonske DC mašine*

Ostvareni momenat DC mašine je direktno proporcionalan struji armature, tako da se očekuje da i u odzivu struje postoji dinamika poput one koja se ima u odzivu momenta. Ono što je karakteristično za odziv struje jeste vremenski interval od 620 do 730 s, tokom kojeg vozilo stiže do visokih vrednosti brzina i do 120 km/čas. Postignuta brzina se nalazi u oblastima tzv. slabljenja polja, gde se pobudni fluks umanjuje zarad postizanja brzina većih od nominalnih. U tim uslovima se narušava princip direktne proporcionalnosti momenta i struje, jer se smanjenje pobudnog fluksa nadomešćuje dodatnim porastom armaturne struje. Karakterističan zaravnjeni oblik struje ukazuje da je došlo do zasićenja u odzivu njenog regulatora, koji je matematički nametnuo referentnu vrednost struje koju nije preporučljivo fizički ostvariti, zbog problema sa komutacijom mašine [4].



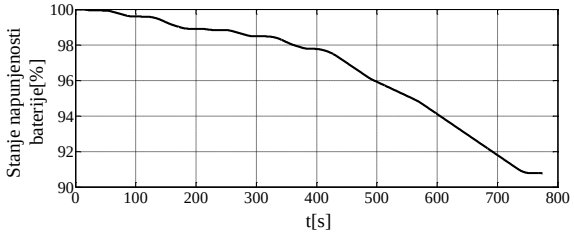
Slika 4.3. *Struja armature pogonske DC mašine*

Na slici 4.4. je prikazan odziv pređenog puta tokom sprovedenog test ciklusa. On iznosi oko 8 kilometara, što odgovara prosečnom ciklusu koji se ima u uslovima gradske vožnje.



Slika 4.4. *Pređeni put tokom test ciklusa*

Na slici 4.5. prikazana je trenutna vrednost napunjenosti baterije tokom sprovedenog ciklusa testiranja. Odziv je dobijen pod pretpostavkom da je pre začetka ciklusa baterija bila potpuno napunjena. Kapacitet baterije je odabran spram kapaciteta komercijalno dostupnih baterija električnih vozila. Pokazuje se da je u uslovima dinamične gradske vožnje domet električnog vozila nešto manji nego što bi to bio slučaj u uslovima vožnje sa konstantnom brzinom, što je bilo i za očekivati.



Slika 4.5. *Stanje napunjenosti baterije tokom test ciklusa*

5. ZAKLJUČAK

U radu su izloženi principi modelovanja pogona sa DC mašinom za potrebe električnih vozila. Prirodna težnja za proširenjem autonomije električnog vozila nameće potrebu za razvijanjem baterija odgovarajućih karakteristika, kao što su kapacitet i specifična gustina snage. Litijum-jonske baterije predstavljaju dobar balans među ovim zahtevima i danas se nameću kao standard u električnim vozilima. Razvijeni model u MATLAB Simulink okruženju analiziran je na standardizovanom NEDC test ciklusu vožnje, koji odgovara gradskim uslovima.

Pokazuje se da se u urbanim uslovima električno vozilo izlaže dinamičkim uslovima rada, koji se direktno ogledaju na preostali kapacitet baterije, ali i na fizička ograničenja regulacionih struktura, koja se tiču dozvoljenih strujnih limita.

6. LITERATURA

[1] Vladan Vučković, *„Električni pogoni“* , Akademska misao, 2002.

[2] Radojle Radetić, *„Prekidačka napajanja“*, Nauka Beograd, 2005.

[3] Miao, Y.; Hynan, P.; von Jouanne, A. „*Current Li-Ion Battery Technologies in Electric Vehicles and Opportunities for Advancements“*. Energies 2019, 12, 1074.

[4] B. Jeftenić, Đ. Oros, V. Vasić, *“Regulisani elektromotorni pogoni-rešeni problemi sa elementima teorije”,* Akademska misao, 2004.

Kratka biografija:

Anđelija Petrović rođena je u Somboru 1997. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Energetska elektronika i električne mašine odbranila je 2020.god.

Dejan Jerkan je docent na Fakultetu tehničkih nauka u Novom Sadu, na Katedri za Energetsku elektroniku i pretvarače. Oblast interesovanja su mu modelovanje i dijagnostika električnih mašina, kao i metoda konačnih elemenata.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE02Pejic

SOFTVERSKI SISTEM ZA PRAĆENJE I ANALIZU POKRETA OČIJU TOKOM IZRADE ON-SCREEN TESTA

SOFTWARE SYSTEM FOR EYE-TRACKING AND DATA ANALYSIS DURING ON-SCREEN TESTING

Marko Pejić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu predstavljen je softverski sistem za praćenje i analizu pokreta očiju ispitanika tokom izrade on-screen testa. Predloženi sistem sastoji se iz dve komponente: klijentska aplikacija koja komunicira sa uređajem za praćenje pokreta očiju i funkcionalnosti obrade i vizuelizacije prikupljenih podataka. Obrada podataka podrazumeva mapiranje sirovih parova koordinata dobijenih praćenjem pokreta očiju u sekvencu regija od interesa. Predloženi sistem obezbeđuje podršku za tri načina vizuelizacije: putem toplotnih mapa pogleda, putem grafa toka pogleda i putem grafa toka regija od interesa. Istraživanje je evaluirano primenom predloženog sistema u okviru eksperimenta u kom je 51 student rešavao on-screen test opšte kulture.*

Ključne reči: *praćenje pogleda, on-screen testiranje, analiza i vizuelizacija podataka*

Abstract – *The paper proposes a software system for eye-tracking and data analysis during on-screen testing. Proposed system consists of two components: client application that communicates with eye-tracking device and functions for processing and visualization of collected data. Data processing involves mapping raw pairs of coordinates obtained by eye-tracking device in a sequence of regions of interest. Proposed system provides support for three types of visualization: via gaze heatmaps, via gaze flow graphs and via flow of regions of interest. The research has been evaluated by applying the proposed system within an experiment in which 51 undergraduate students took a general knowledge test.*

Keywords: *eye-tracking, on-screen testing, data analysis and visualization*

1. UVOD

Primena računara za potrebe testiranja studenata donela je veliki broj prednosti kako predavačima, tako i samim studentima. Neke od tih prednosti su: brže i jednostavnije ocenjivanje, jednostavnije skladištenje testova, izgubila se potreba za štampom velikog broja testova, moguće je automatizovati ocenjivanje, itd. Pored navedenih prednosti, upotreba računara za potrebe testiranja donela je i veliki broj mogućnosti za prikupljanje i analizu različitih tipova podataka, kombinujući računarske komponente i različite senzore. Upotrebom računara moguće je imati uvid u znatno veći broj parametara samog toka rešavanja testova, za razliku od testiranja na papiru, kada je jedini parametar bio rezultat (osvojeni broj bodova). Sada je moguće znati koliko je student provodio vremena na kom pitanju, da li se dvoumio prilikom davanja odgovora (kliknuo više ponuđenih odgovora), da li više vremena provodi gledajući slike koje idu uz pitanje ili ponuđene odgovore i slično. Ovaj rad bavi se upravo primenom metode praćenja pogleda za prikupljanje podataka tokom izrade *on-screen* testa, uz obradu i vizuelizaciju dobijenih podataka.

Osnovna ideja ovog rada jeste prikupljanje podataka o ponašanju studenata prilikom testiranja metodom praćenja pogleda, kao i obrada i vizuelizacija prikupljenih podataka. Motivacija za implementaciju sistema za praćenje i analizu pokreta očiju tokom izrade *on-screen* testova jeste u mogućnostima koje postojanje ovakvog sistema otvara. Vizuelizacijom i analizom podataka prikupljenih na opisan način moguće je utvrditi koji delovi testa privlače najviše pažnje studentima, da li studenti obraćaju više pažnje na slike ili na tekst, moguće je identifikovati pitanja koja su izazivala najviše nedoumica kod studenata i slično. Prema tome, uz adekvatnu analizu vizuelizacija tokova pogleda ispitanika prilikom rešavanja *on-screen* testova moguće je i vršiti personalizaciju testova.

Pod personalizacijom testova podrazumeva se prilagođavanje testova studentima, u skladu sa zaključcima donesenim analizom rezultata vizuelizacije podataka. Ovakvom personalizacijom može se znatno poboljšati kvalitet nastave, kao i rezultati koje studenti postižu na testovima, bez snižavanja kriterijuma.

Pored personalizacije, još jedan motiv za razvoj ovakvog sistema jeste obezbeđivanje softverske podrške za prikupljanje podataka praćenjem pokreta očiju, što može biti veoma korisno za istraživanja iz različitih oblasti: neurologija, eksperimentalna psihologija, računarske nauke, itd.

2. PREGLED RELEVANTNE LITERATURE

Upotreba metodologije praćenja pogleda tokom posete veb sajtovima je veoma popularna, pa zbog toga postoji veliki broj radova na tu temu. U radu [1] akcenat je na prikupljanju podataka metodom praćenja pogleda i vizuelizaciji. Rad [1] bavi se i analizom razlika fokusa pažnje različitih polova, prilikom pregleda sadržaja veb stranice. Takođe, postoji i značajan broj radova koji se bave upotrebom metode praćenja pogleda u obrazovne svrhe. Jedan od

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Goran Savić, vanr. prof.

takvih radova je [2], čiji cilj je analiza pažnje studenata na slike u okviru *PowerPoint* prezentacije. U radu [2], akcenat je stavljen na vizuelizaciji prikupljenih podataka, da bi se jasno uočili delovi prezentacije na koje studenti obraćaju najviše pažnje, što je pristup sličan onom opisanom u ovom radu.

3. SOFTVERSKI SISTEM ZA PRAĆENJE I ANALIZU POKRETA OČIJU

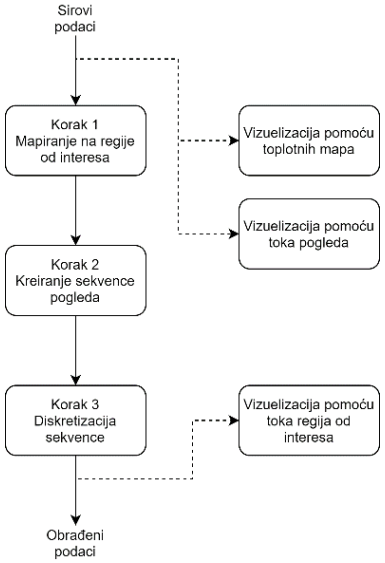
U ovoj glavi detaljnije će biti objašnjena implementacija klijentske aplikacije, procesiranje i vizuelizacija podataka dobijenih praćenjem pokreta očiju.

3.1. Implementacija klijentske aplikacije

Da bi prikupljanje podataka bilo moguće, neophodno je implementirati klijentsku aplikaciju koja će se koristiti za testiranje studenata. Zadatak ove aplikacije je da poseduje adekvatan korisnički interfejs, u okviru kog se prikazuju pitanja i korisniku je omogućen izbor jednog od ponuđenih odgovora, kao i povezivanje sa uređajem za praćenje pogleda i prikupljanje podataka. Korišćen je *Gazepoint GP3* uređaj za praćenje pogleda, koji pokreće server na *localhost*-u. Klijentska aplikacija treba da uspostavi komunikaciju sa *Gazepoint* serverom preko porta 4242, pošalje adekvatne komande u XML (*Extensible Markup Language*) formatu i skladišti podatke koje joj server šalje. Pored podataka dobijenih metodom praćenja pogleda, klijentska aplikacija beleži i odgovore na pitanja, kao i informaciju da li je izabrani odgovor tačan. Klijentska aplikacija implementirana je upotrebom .NET WPF tehnologije i C# programskog jezika.

3.2. Obrada podataka

Podaci dobijeni metodom praćenja pogleda su u formatu X i Y koordinata, što nije dovoljno semantički bogato za realizaciju svih navedenih tipova vizuelizacije. Zbog toga, neophodno je transformisati sirove podatke u odgovarajući oblik, što je zadatak procesiranja. Faza procesiranja realizovana je iz više koraka, od kojih su najznačajniji: mapiranje parova koordinata na regije od interesa, kreiranje sekvence pogleda i smanjivanje uticaja vremenske dimenzije. Slika 1 prikazuje dijagram toka aktivnosti prilikom procesiranja podataka.



Slika 1. *Dijagram toka procesiranja podataka*

Koracima prikazanim na slici 1 dobijaju se podaci koji omogućuju realizaciju svih potrebnih tipova vizuelizacije, kao i podaci koji su primenjivi za dalja istraživanja koja će se osloniti na ovo istraživanje.

Mapiranje parova koordinata na regije od interesa je prva faza procesiranja. Da bi ovakvo mapiranje bilo realizovano, neophodno je definisati regije od interesa i njihove granice (dve tačke – gornji levi i donji desni ugao) za svako pitanje. Kada postoje jasno definisane granice svih regija od interesa za sva pitanja testa, u 2D prostoru jednostavno se može odrediti kojoj regiji pripada svaka od tačaka. Ovim se sirovim parovima koordinata dobijenim od strane uređaja za praćenje pogleda daje određena semantika, tj. svaka tačka se mapira na tačno jednu regiju od interesa. Za predstavu svake regije od interesa odabran je po jedan karakter, tako da dobijena sekvenca predstavlja niz karaktera, tj. jedan string koji predstavlja način na koji je određeni student posmatrao jedno pitanje. Podaci koji se evidentiraju metodom praćenja pogleda sačinjeni su iz sledećih atributa: vreme (interna meru koju beleži uređaj), X i Y koordinate i validnost podatka. Ovaj skup sirovih podataka proširuje se rezultatima prvih koraka procesiranja, tj. dodaju se atributi: trajanje pogleda (razlika vremena dve susedne tačke) i regija od interesa na koju se date koordinate odnose.

Mapiranjem X i Y koordinata na regije od interesa dobija se sekvenca toka pogleda studenta u okviru jednog pitanja. Međutim, ovakva sekvenca je previše osetljiva na vremensku dimenziju. Pošto uređaj za praćenje pogleda šalje podatke velikom frekvencijom, očigledno je da će se u ovakvom skupu podataka pojavljivati grupe u okviru kojih se mnogo puta ponavlja isti simbol regije od interesa. Na primer, ukoliko je student potrošio dve sekunde na čitanje naslova, u skupu podataka će to biti zapisano kao 120 ponavljanja T simbola regije od interesa. Zbog toga, neophodno je transformisati dobijenu sekvencu regija od interesa u odgovarajući oblik.

Prvi korak u transformaciji sekvenci regija od interesa je smanjivanje dužine originalne sekvence. Ovo je realizovano tako što su svi susedni isti simboli regija predstavljeni pomoću jednog simbola, a sačuvana je i informacija o trajanju pogleda. Ovim se dobija znatno kraća sekvenca, ali vreme i dalje ima prevelik uticaj. Upravo zbog toga, neophodno je pronaći adekvatnu meru između potpunog zanemarivanja vremenske dimenzije i ostavljanja vremena u sirovom obliku.

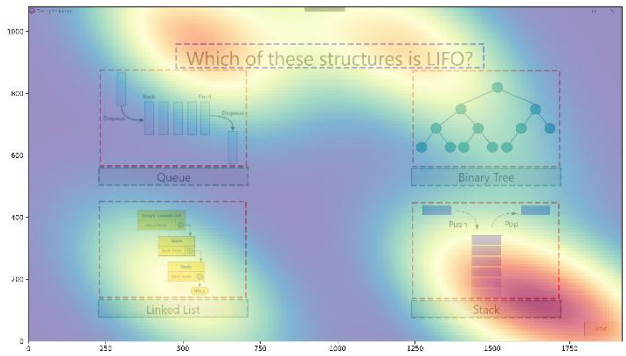
Smanjivanje uticaja vremenske dimenzije realizovano je diskretizacijom. Kontinualna vrednost vremena zadržavanja pogleda unutar neke regije diskretizovana je na dve moguće vrednosti, u zavisnosti od definisanog praga. Prag je određen kao medijan vremena zadržavanja unutar svih regija za sve studente, za određeno pitanje. Finalne sekvence formiraju se na sledeći način: ukoliko je vreme zadržavanja unutar regije od interesa označene karakterom ‘X’ manje od praga, u sekvencu se dodaje jedan karakter ‘X’. U suprotnom, u sekvencu se dodaju dva karaktera ‘X’.

3.3. Vizuelizacija

Vizualizacija je od velikog značaja za ovaj rad, jer omogućuje prikaz rezultata koraka obrade podataka, kao i dublju analizu interakcije ispitanika sa pitanjima *on-screen*

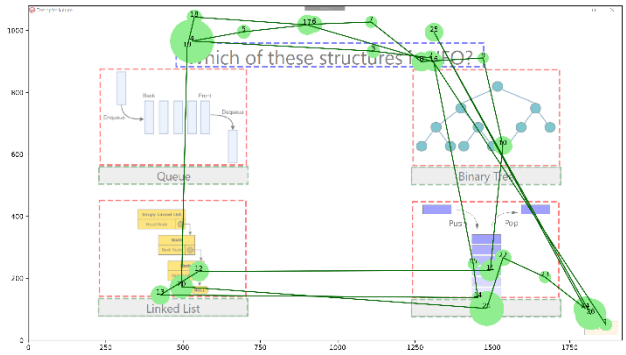
testa. U okviru ovog istraživanja realizovana su tri različita tipa vizuelizacije, koja su detaljnije objašnjena u nastavku.

Prvi način vizuelizacije je pomoću toplotnih mapa. Ovim načinom se korišćenjem sirovih (X, Y) koordinata pogleda iscrtavaju posmatrane regije od interesa. Sa ovakvih grafika moguće je zaključiti na koje delove pitanja je ispitanik obratio najviše pažnje, ali se gubi informacija o toku pogleda. Slika 2 ilustruje ovakav vid vizuelizacije.

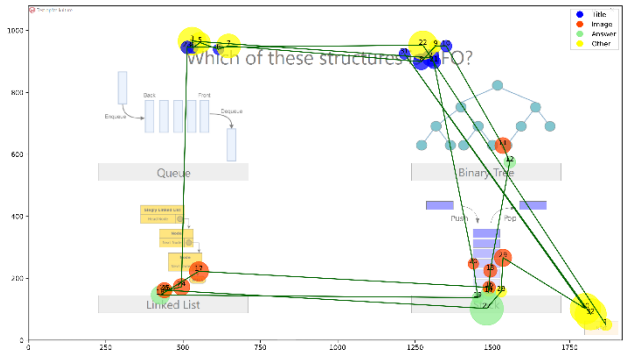


Slika 2. *Vizuelizacija putem toplotnih mapa*

Preostala dva načina vizuelizacije razvijena su sa ciljem očuvanja toka pogleda u vizuelnom prikazu. Drugi način vizuelizacije takođe koristi sirove (X, Y) koordinate, ali za razliku od toplotnih mapa iscrtava tok pogleda. Funkcija koja realizuje ovaj način vizuelizacije prvo grupiše tačke koje se nalaze u okviru iste delta okoline i potom iscrtava dijagram koji predstavlja tok pogleda između formiranih grupa tačaka. Prečnik čvora na grafu srazmeran je broju pogleda usmerenih u okolinu te tačke, tj. dužini zadržavanja pogleda. Brojevima unutar čvorova definisan je tok pogleda. Slika 3 ilustruje ovakav vid vizuelizacije.



Slika 3. *Vizuelizacija putem toka pogleda*



Slika 4. *Vizuelizacija putem toka regija od interesa*

Poslednji, treći način vizuelizacije predstavljen je na slici 4. Ovim načinom ilustruje se tok kojim su posmatrane regije od interesa u okviru određenog pitanja, kao i trajanje zadržavanja pogleda u odgovarajućim regijama.

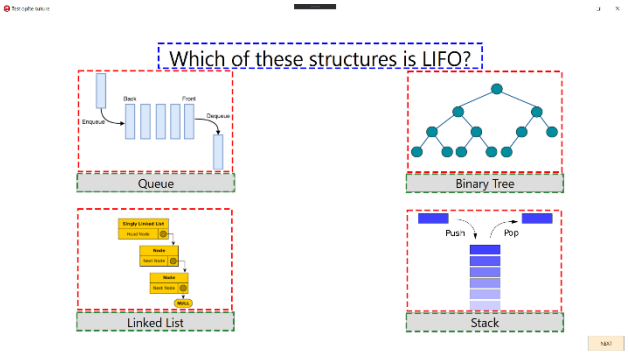
Implementacija svih obrada podataka i načina vizuelizacije realizovana je upotrebom *Python* programskog jezika i biblioteka *numpy* [3], *pandas* [4] i *matplotlib* [5].

4. STUDIJA SLUČAJA

Za potrebe ovog rada sproveden je eksperiment u kom je učestvovao ukupno 51 student. Velik deo ispitanika su studenti viših godina Fakulteta tehničkih nauka, uglavnom sa smerova koji se bave računarstvom. Svi učesnici su rešavali *on-screen* test opšte kulture kreiran za potrebe eksperimenta, koji se sastojao od ukupno 20 pitanja.

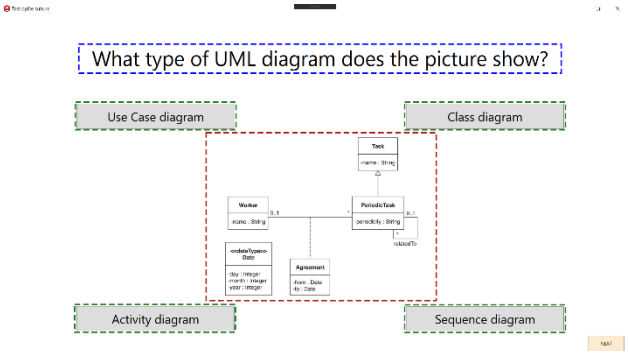
Za izvođenje eksperimenta neophodni su računar za kojim student rešava test i uređaj za praćenje pokreta očiju, postavljen ispod monitora, na adekvatnoj udaljenosti od ispitanika.

Ako sadržaj pitanja posmatramo sa vizuelne tačke gledišta, postojala su dva različita tipa pitanja. Pitanja prvog tipa uz svaki ponuđeni odgovor imaju i sliku. Prema tome, pitanja ovog tipa sastavljena su iz deset regija od interesa: tekst pitanja, četiri ponuđena odgovora, četiri slike uz ponuđene odgovore i ostalo, tj. deo površine pitanja koji nije neka od prethodno navedenih regija. Slika 5 ilustruje jedno od pitanja prvog tipa.



Slika 5. *Primer pitanja prvog tipa*

Pitanja drugog tipa imaju četiri ponuđena odgovora i jednu sliku koja detaljnije opisuje tekst pitanja ili je predmet pitanja. Ova pitanja sastavljena su iz sedam regija: tekst pitanja, četiri ponuđena odgovora, slika uz tekst i ostalo. Slika 6 prikazuje jedno od pitanja drugog tipa.



Slika 6. *Primer pitanja drugog tipa*

Podaci za studente prikupljani su tako što je uređaj za praćenje pogleda slao informacije tokom trajanja celog testa, a klijentska aplikacija je beležila te podatke u CSV (*Comma Separated Values*) fajl.

Rezultat eksperimenta jeste skup podataka sačinjen iz 51 CSV fajla koji sadrže sirove podatke dobijene od uređaja za praćenje pogleda. Pored toga, beleže se i odgovori ispitanika, i da li su ti odgovori bili tačni. Tabela 1 prikazuje deo sirovih podataka, zajedno sa izračunatim trajanjem i regijom od interesa, dobijenih procesiranjem prikupljenih podataka.

Tabela 1. *Skup podataka sa dodatim regijama od interesa*

Vreme	X	Y	Trajanje	Regija
7047.959	0.74574	0.2346	0.0166	Q
7047.975	0.74662	0.22547	0.01612	Q
7047.992	0.746	0.21951	0.0166	Q
7048.007	0.7408	0.1817	0.01562	T
7048.024	0.73939	0.17776	0.01661	T
7048.041	0.73447	0.17774	0.01709	T
7048.057	0.73098	0.18239	0.01562	T
7048.074	0.72809	0.18769	0.01709	O
7048.09	0.72524	0.18254	0.0166	T

U trećoj glavi detaljnije je objašnjen postupak transformacije podataka iz formata prikazanog u tabeli 1 u sekvencu regija od interesa. Tabela 2 predstavlja izgled nekoliko sekvenci kreiranih na već opisan način.

Tabela 2. *Obrađene sekvence regija od interesa*

Sekvenca regija od interesa
OPTOOTOOTTOOTTOTTOOTTOARROSSOPPOSSOB OQQBOORROSSQOOSSOSSOOSOOSBQTTOO
OSQTTOTOTOOTOOTOOTTOTOTQQBSSDDOSSORR CCODDSQTTOOTPARRORRCODDOSSOOSQTOOTOSOOSSO
OTTOTTOTTOOTTOQOPPOAARCORROCCOROSSQO TOOTTOOTTQQBSDDSSBOQQTTOOTQBSSOODDOODOO
OOSOOPPOOTOOQQBBSSDOOPRCCRROCCOOCCOOSO
PPTTOTOOTOOQQBOBOOSSDDOOCCORAOAAPPOQQB BOSSODDOSOROOAAOSSDOOSOP

5. ZAKLJUČAK

U ovom radu opisan je softverski sistem za praćenje i analizu pokreta očiju ispitanika tokom izrade *on-screen* testa. Sproveden je eksperiment sa preko 50 ispitanika koji su rešavali test opšte kulture, dok je uređaj za praćenje pokreta očiju beležio aktivnosti njihovih zenica.

Predloženi sistem sadrži funkcionalnosti prikazivanja *on-screen* testa i skladištenja podataka dobijenih praćenjem pokreta očiju, kao i funkcionalnosti obrade i vizuelizacije tih podataka.

Aplikacija za izvođenje on-screen testa i skladištenje podataka implementirana je upotrebom C# programskog jezika i .NET WPF tehnologije. Praćenje pokreta očiju ispitanika realizovano je upotrebom *Gazepoint GP3* uređaja. Obrada i vizuelizacija prikupljenih podataka realizovana je upotrebom *Python* programskog jezika.

Problem koji može uticati na validnost rezultata i podataka je osetljivost uređaja na pomeranje ispitanika. Zbog osetljivosti uređaja, poželjno je da ispitanik sedi mirno i pravi što manje pokreta glavom, a zbog same tematike testa neophodno je da se ponaša što prirodnije.

Ovaj rad može poslužiti kao osnova za dalja istraživanja, koja bi mogla biti usmerena ka primeni implementiranog softverskog sistema u različitim istraživanjima. Značaj ovog rada jeste i veliki broj učesnika u eksperimentu, što omogućava formiranje većeg, javno dostupnog skupa podataka.

Jedan od pravaca budućeg istraživanja može biti proširivanje predloženog sistema sa funkcionalnostima koje bi nad podacima primenjivale tehnike mašinskog učenja. Istraživanja koja se bave primenom mašinskog učenja nad podacima dobijenim praćenjem pokreta očiju mogu da koriste ceo proces obrade podataka opisan u ovom radu za formiranje sekvenci toka pogleda. Sekvence formirane na opisan način mogu se koristiti za dalju obradu tehnikama istraživanja podataka (eng. *data mining*). Neke od mogućnosti za primenu tehnika mašinskog učenja nad ovim podacima su identifikacija šablona toka pogleda, predikcija nivoa znanja ispitanika i slično.

Za ovakva istraživanja, od značaja mogu biti i implementirane vizuelizacije, kako bi se detaljnije analizirali rezultati algoritama mašinskog učenja.

6. LITERATURA

[1] M. Schiessl, S. Duda, A. Tholke and R. Fisher, Eye tracking and its applications in usability and media research, Berlin, 2003.

[2] D. A. Slykhuis, E. N. Wiebe and A. A. Len, Eye-Tracking Students Attention to PowerPoint Photographs in a Science Education Setting, 2005.

[3] "The fundamental package for scientific computing with Python" NumPy. [Online]. Available: https://numpy.org/. [Accessed: Jun-2020].

[4] "Python Data Analysis Library" Pandas. Available: https://pandas.pydata.org/. [Accessed: Jun-2020].

[5] "Visualization with Python" Matplotlib. [Online]. Available: https://matplotlib.org/. [Accessed: Jun-2020].

Kratka biografija:

Marko Pejić rođen je u Novom Sadu 1996. god. Master rad na Fakultetu tehničkih nauka iz oblasti Računarstva i automatike – Savremene obrazovne tehnologije i standardi odbranio je 2020.god. kontakt: markopejic@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/10BE04Dragoljevic

РАЗВОЈ СИСТЕМА ЗА ПРЕТРАГУ НАУЧНИХ РАДОВА УПОТРЕБОМ GraphQL–А И ПРОГРАМСКОГ ЈЕЗИКА СКАЛА

DEVELOPMENT OF SCIENTIFIC PAPERS SEARCH ENGINE USING GraphQL AND SCALA PROGRAMMING LANGUAGE

Смиљана Драгољевић, Милан Видаковић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај: *Задатак рада представља развој система за претрагу научних радова. Приликом имплементације серверског дела система употребљен је језик за комуникацију са сервером, GraphQL, а када су у питању програмски језици, систем је реализован употребом програмског језика Scala. Клијентски део система представља апликацију имплементирану помоћу React-a, JavaScript библиотеке. Спецификација система одрађена је употребом UML dijagrama.*
Кључне речи: *Scala, GraphQL, MongoDB, Elasticsearch, React*
Abstract: *The aim of this paper is to develop a search engine for scientific papers. For the implementation of the system's server part GraphQL was used. When it comes to programming languages, the system was implemented using the programming language Scala. The client application was implemented as a React application where React is a JavaScript library. The specification of the system was done using UML diagrams.*

Keywords: *Scala, GraphQL, MongoDB, Elasticsearch, React*

1. УВОД

У раду ће детаљно бити појашњен један од језика који комбинује објектно-оријентисано и функционално програмирање, а то је Скала. Скала ће се демонстрирати на примеру имплементирања веб апликације која представља систем за претрагу научних радова. Детаљан опис ове апликације следи на крају рада. Још једна област која ће бити темељно обрађена јесте једна од алтернатива REST-u (Representational state transfer), а то је *GraphQL*.

Информације о корисницима и радовима, као и *pdf* рада чувају се у *MongoDB* бази података, док је претрага рада по тексту и филтрација резултата омогућена употребом *Elasticsearch*-а, сервера који служи за претрагу и анализу података.

Клијентски део апликације урађен је као *React* апликација, где *React* представља једну од водећих *JavaScript* библиотека за изградњу корисничког интерфејса.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био проф. др Милан Видаковић.

2. ТЕХНОЛОГИЈЕ

У овом поглављу фокус је дат пре свега Скали и *GraphQL*-у, а након тога следи осврт на остале технологије помоћу којих је реализован поменути систем за претрагу научних радова. Како Скала поред подршке за објектно-оријентисано има и пуну подршку за функционално програмирање, дат је увод у функционално програмирање и поменути су основни појмови које ова парадигма уводи.

Функционално програмирање

Функционално програмирање спада у декларативно програмирање, где се акценат даје ономе што програм треба да обави, а не начину на који се неки задатак обавља. Основни појмови функционалног програмирања јесу функције вишег реда, непроменљиви подаци и чисте функције. Функција се посматра као и сваки други тип података. Уводи се појам функције вишег реда, тј. функције којој је могуће проследити функцију као параметар и вратити функцију као повратну вредност. Такође, могуће је варијабли доделити дефиницију функције. Чисте функције су функције које примају улазне параметре на основу којих израчунавају излаз и за исте улазне податке дају исту повратну вредност.

Скала

Скала је статички типизирана комбинација објектно-оријентисаног и функционалног програмирања [1]. Мартин Одески је почео са развојем Скале 2001. године на EPFL-u (École Polytechnique Fédérale de Lausanne), а прва верзија је званично објављена 2004. године. Скала код се компајлира у Јава бајт код који је углавном подједнако ефикасан као и Јава код [2]. Извршава се на Јава виртуелној машини, што пружа могућност коришћења Јава библиотека у Скала коду. Ово је чист објектно-оријентисани језик који сваки тип података, чак и оне који се сматрају прмитивним у другим језицима, посматра као објекат. Иако је у Скали могуће имплементирати апликацију коришћењем чисте објектно-оријентисане нотације, овај језик нуди доста предности уколико се употребљавају функционални концепти које он уводи. Једна од могућности коју нуди и подстиче функционалност Скале јесте рад са непроменљивим подацима (листинг 1).

```
val fruits = List("apple", "banana")
val fruitsUpperCase = fruits.foreach(
  fruit => fruit.toUpperCase())
```

Листинг 1 – Пример трансформације објекта

Поред овога, свака функција у Скали представља вредност те је могуће дефинисати функције вишег реда и угњеждавати функције.
Када се говори о Скали битно је поменути на који начин овај језик решава проблем паралелизације кода. Уводи се библиотека за подршку *Actor* модела. *Actor* је ентитет који комуницира са другим *Actor* -има путем порука [3].

Play framework

Play framework је радно окружење имплементирано у Скали и употребљава се за креирање модерних веб апликација. Користи се за израду кода који се компајлира у Јава бајт код, тј. Јава и Скала кода. *Play* троши минимално ресурса и пружа високо скалабилне апликације [4]. Архитектура овог радног окружења се заснива на MVC (*model-view-controller*) моделу што га чини једноставним и лаким за употребу.

GraphQL

GraphQL је језик за комуникацију са API-јем који нуди ефикасан приступ креирања и обраде захтева клијената упућеним серверској страни апликације. GraphQL сервер подржава, између осталог, и рад са Скалом, Јавом, C++-ом, Пајтоном, итд. Комуникација са сервером путем *GraphQL* -а врши се HTTP (*Hypertext Transfer Protocol*) протоколом преко једног endpoint-а, најчешће POST захтевом. *GraphQL* захтев је текст који се шаље серверу где се интерпретира и обрађује, који потом враћа JSON (*JavaScript Object Notation*) одговор [5]. Подржане су све операције CRUD-а (*create, read, update, delete*). На листингу 2 дат је пример *GraphQL* упита који као одговор очекује наведена поља, и сам одговор са сервера.

```
{
  person {
    name
    age
  }
}
{
  "data": {
    "person": {
      "name": "Anna",
      "age": 25
    }
  }
}
```

Листинг 2 – Пример *GraphQL* упита и одговора

Како би се одвијала комуникација са API-јем, неопходно је дефинисати шему која одређује могуће захтеве као и одговоре сервера на те захтеве. Свако поље шеме има свој назив и тип. *GraphQL* поља могу имати и придружене аргументе. За свако поље је неопходно специфицирати функцију која одређује вредност тог поља наког извршавања упита на серверу. Уколико вредност поља није скаларна, потребно је дефинисати функције за његова поља, све док се не достигну скаларне вредности. Корен *GraphQL* захтева јесте поље *schema*, које даље може да садржи поља типа *Query* и *Mutation*. *Query* означава операције добављања података, без споредних ефеката, док се *Mutation* користи за креирање, модификацију и брисање података. Листинг 3 приказује пример *Query* захтева.

```
query PersonNameAndAge {
  person(id: 1) {
    name
    age
  }
}
```

Листинг 3 – Пример *Query* захтева

Када је дефинисана шема, могуће је валидирати *GraphQL* упит у односу на ту шему. Неке од могућих грешака јесу синтаксна, затим ненавођење жељених поља у склопу поља чији тип није скаларна вредност или навођење поља унутар поља чија вредност јесте скаларна. Уколико је шема валидна, *GraphQL* сервер извршава упит и враћа резултат чија структура наликује упиту, типично у JSON формату [6].
Једна од највећих разлика између *GraphQL* -а и REST архитектонског стила јесте у начину приступа подацима. REST стил налаже да се сваки тип података представи као посебан ресурс и као такав шаље клијенту на захтев. Међутим, у доста случајева се јавља потреба за добављањем више типова података одједном или комбинацијом разних података. *GraphQL* нуди ефикаснији начин приступа ресурсима јер је могуће дефинисати *GraphQL* шему на такав начин да су сви подаци система повезани, те је могуће добавити жељене податке једним захтевом. У случајевима када клијенту нису потребне све ове информације, *GraphQL* има предност у односу на REST, јер се унутар захтева дефинишу жељени подаци које клијент добавља за своје потребе, те је могуће изоставити непотребне вредности.
Имплементација *GraphQL* -а у Скали омогућена је библиотеком *Sangria*. Ова библиотека у потпуности подржава дефинисање шеме са свим типовима које нуди *GraphQL*.

***MongoDB* база података**

MongoDB је дистрибуирана, објектно-оријентисана база података која податке чува у JSON формату. Овакав начин складиштења уводи додатну флексибилност приликом коришћења *MongoDB* јер структура података може да варира од документа до документа. Документи су организовани унутар колекција које углавном чувају документе сличне намене. Ову базу података је могуће користити и као систем датотека. *GridFS* је *Mongo* спецификација за складиштење и добављање већих датотека као што су слике, аудио и видео датотеке, итд [7].

Elasticsearch

Elasticsearch је брз и дистрибуирани сервер за претрагу и анализу разних типова података од безбедносних, преко претраге веб страница до претраге геопросторних података. Такође се користи за одређивање метрике система и складиштење и анализу лог фајлова. Подаци се групишу унутар индекса, где индекс представља сет међусобно повезаних докумената у JSON формату.

React

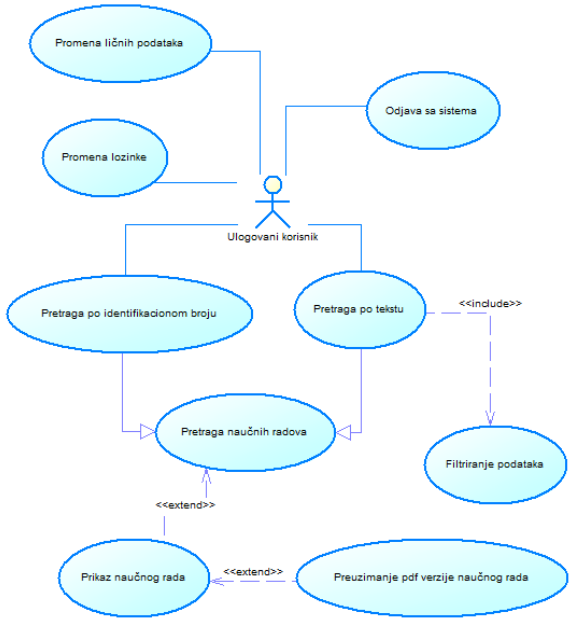
React је *JavaScript* библиотека која омогућава једноставно креирање веб и мобилних апликација. *React* води рачуна само о исцртавању веб страница док је за имплементацију осталих жељених функционалности као што су на пример рутирање апликације и

управљање глобалним стањем, потребно импортовати засебне библиотеке. Свака *React* апликација се састоји од компоненти које су налик функцијама које за повратну вредност имају HTML (*Hypertext Markup Language*) елементе. Приликом креирања апликације користи се екстензија *JavaScript*-а, JSX односно *JavaScript* XML. JSX омогућава писање HTML елемената унутар *JavaScript* -а.

3. СПЕЦИФИКАЦИЈА ЗАДАТКА

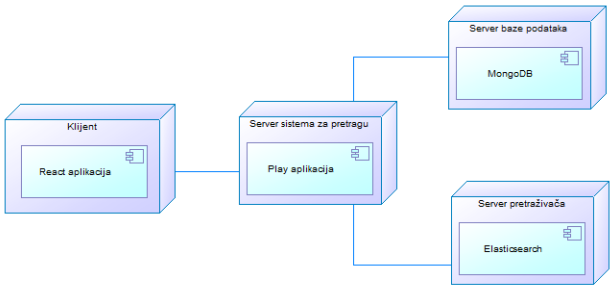
Систем описан у раду представља систем за претрагу научних радова, погодан за употребу унутар једног или више факултета. Информације о раду преузете су из самог рада, тачније из одељка кључна документацијска информација. Такође је доступна и финална верзија рада у *pdf* формату.

Корисник се пријављује на систем користећи свој *email* налог, а уколико није пријављен, нуди му се опција регистрације на систем. Након успешне пријаве, улогованом кориснику се приказују сви радови доступни систему које он потом може претраживати или филтрирати. Једна од претрага понуђена кориснику јесте претрага по тексту рада. Ове радове је могуће додатно филтрирати одабиром године, научне области, ментора, издавача или типа рада. Такође је могућа комбинација поменутих филтера. Уколико је кориснику познат идентификациони број научног рада, може му директно приступити. Пошто је извршена претрага, улоговани корисник има опцију приступа додатним информацијама о раду или по потреби преузимања *pdf*-а рада. Слика 1 приказује дијаграм случајева коришћења улогованог корисника.



Слика 1 – Дијаграм случајева коришћења улогованог корисника

Систем за претрагу научних радова реализован је помоћу неколико компоненти које међусобно комуницирају (слика 2). Серверска апликација представља засебан чвор система који садржи једну *Play* апликацију. Ова апликација комуницира са сервером базе података, конкретно *MongoDB* базе, који представља још једну компоненту система у којој се чувају све информације о раду заједно са његовом *pdf* верзијом. Поред базе података, серверска апликација има везу са *Elasticsearch*-ом чији је сервер такође реализован као засебна компонента која складишти само информације неопходне за претрагу као што су цео текст рада и подаци који се користе приликом филтрирања. Последња компонента јесте клијентска, *React* апликација која комуницира са *Play* апликацијом.



Слика 2 – *Deployment* дијаграм

4. ИМПЛЕМЕНТАЦИЈА

Систем за претрагу научних радова реализован је као вишеслојна апликација која се састоји од слоја података, слоја пословне логике и презентационог слоја. У наставку поглавља следи опис имплементације сва три слоја употребом технологија описаних у поглављу 3.

Складиште података

За реализацију слоја података, једно од коришћених складишта јесте *MongoDB* база података. Састоји се од две колекције које садрже податке о свим научним радовима система и податке о регистрованим корисницима система.

Подаци из *MongoDB* базе података користе се само приликом приказа детаљних информација о научном раду, док се за претрагу рада користе подаци из *Elasticsearch*-а. Апликација користи један индекс, *scientific-paper*, који индексира поља:

- *fulltext* - цео текст рада, употребљен приликом претраге рада по тексту,
- поља за која се формирају на основу филтера и
- поља за приказ радова након извршене претраге.

Серверски део система

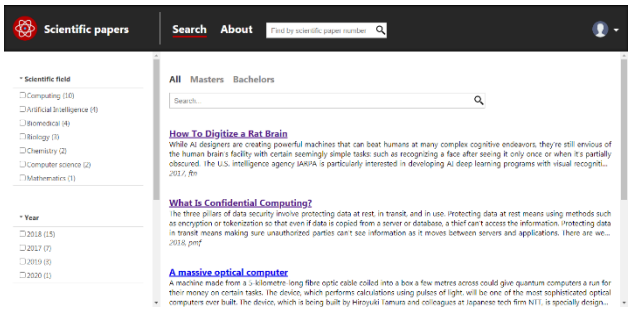
Серверски део система састоји се од једне апликације имплементиране употребом Скале и *Play Framework*-а. Комуникација са овим слојем извршена је употребом језика за комуникацију, *GraphQL*-а, тачније библиотеком која га имплементира у Скали, *Sangria*.

GraphQL шема дефинише поља која представљају операције за претрагу радова, пријаву или регистрацију на систем, промену података о тренутно улогованом кориснику као и промену лозинке уколико ју је корисник заборавио. Креиран је контекстни објекат који садржи методе за израчунавање

сваког од ових поља. Манипулација научним радовима и манипулација корисницима раздвојене су унутар засебних класа, и контекстни објекат садржи објекте ових класа, чијим се индиректним позивом израчунавају вредности *GraphQL* поља.
Приликом имплементације рада са *MongoDB*-ом искоришћен је додатак за *Play*, *ReactiveMongo*. Класа за рад са овом базом садржи све методе неопходне за рад са научним радовима и корисницима. Сервис за рад са *Elasticsearch*-ом имплементиран је употребом клијента *elastic4s*.
Добављање *pdf* формата рада измештено је у засебан контролер, независно од *GraphQL*-а. Рад се проналази на основу идентификационог броја и враћа се у виду *pdf*-а клијенту.

Клијентски део система

Клијентски део система реализован је као *React* апликација, док је комуникација путем *GraphQL*-а имплементирана употребом *Apollo* клијента, *GraphQL* клијента који се, између осталог, интегрише са *React* апликацијама.
Након успешне пријаве на систем, апликација аутоматски шаље захтев за претрагу свих научних радова, без филтрирања. Како је пагинација подржана у систему, клијенту се допрема првих 10 радова, колико је подешено (слика 3).



Слика 3 – Приказ радова након успешне пријаве на систем

Након овога, корисник може одабиром филтера са леве стране приказа филтрирати резултате или, уносом кључне речи у поље предвиђено за то сузити опсег претраге.
Приликом имплементације претраге, креирано је неколико компоненти од којих једна чува стање апликације као што су одабрани филтери, текст за претрагу или жељена страна резултата претраге. Поред овога у њој су дефинисане основне методе за руковање елементима претраге које су као такве, прослеђене одговарајућим компонентама за приказ филтера као и приказ резултата претраге. Приликом исцртавања ове компоненте, као и приликом измене њеног стања кога чине параметри претраге, креира се *GraphQL* захтев, којем се прослеђују поменути параметри, те се овакав захтев шаље серверу. За креирање и слање захтева задужен је *Apollo* клијент.

5. ЗАКЉУЧАК

Систем за претрагу научних радова развијен је обухватајући сет основних функционалности неопходних брзом приступу раду. Даља проширивост система подразумевала би прилагођавање система потребама разних научних установа и корисника. Савремене технологије употребљене приликом имплементације би омогућиле његову једноставну и ефикасну надоградњу, као и одржавање.
Иако и даље коришћен већином у научне сврхе, програмски језик Скала све више и више продире у израду система за свакодневну употребу те на тај начин уводи функционално програмирање у индустријски свет. Такође, језик за комуникацију *GraphQL*, проналази своје место у већим системима како би се корисници што ефикасније прилагодили услугама које нуди сервер.

6. ЛИТЕРАТУРА

[1] Programming in Scala, Martin Odersky, Lex Spoon, Bill Venners, 2007, 2008
[2] A Brief History of Scala, Martin Odersky, June 9, 2006
[3] akka-actor, https://www.javatpoint.com/akka-actor
[4] Play Framework, https://www.playframework.com
[5] GraphQL, https://engineering.fb.com/core-data/graphql-a-data-query-language
[6] GraphQL, https://graphql.org
[7] MongoDB, https://www.tutorialspoint.com/mongodb

Кратка биографија:

Смиљана Драгољевић рођена је 08.11.1995. године у Новом Саду, где је и завршила основну школу „Жарко Зрењанин“. Након тога у истом граду уписује гимназију „Јован Јовановић Змај“ коју завршава 2014. године. Исте године уписује Факултет техничких наука, Универзитета у Новом Саду на смеру Рачунарство и аутоматика. На трећој години студија опредељује се за смер Примењене рачунарске науке и информатика. Дипломирала је 2018. године те је уписала мастер академске студије на истом факултету, смер Рачунарство и аутоматика, модул Електронско пословање. Положила је све испите предвиђене планом и програмом.

Милан Видаковић рођен је у Новом Саду 1971. године. На Факултету техничких наука у Новом Саду завршио је докторске студије 2003. године. На истом факултету је 2014. године изабран за редовног професора из области *Примењене рачунарске науке и информатика*.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.38:620.9
DOI: https://doi.org/10.24867/10BE05Hrubenja

ANALIZA PROPADA NAPONA U POJEDINIM TAČKAMA PRENOSNE MREŽE SRBIJE

ANALYSIS OF VOLTAGE SAGS IN CERTAIN POINTS OF THE TRANSMISSION NETWORK OF SERBIA

Miron Hrubenja, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu se razmatraju propadi napona prilikom simulacije kvara na različitim udaljenostima. Rezultate simulacije upoređujemo sa rezultatima realnih merenja. Cilj je da se rezultati simulacije na približno istoj udaljenosti mesta kvara kao i u realnom sistemu što manje razlikuju od merenja u realnom sistemu. Razmatranja su izvršena na osnovu realnih snimaka uređajem mikroprocesorske zaštite SIPROTEC 47SA611, dok su rezultati simulacije dobijeni na osnovu modela dalekovoda modelovanog u programskom paketu MATLAB.*

Ključne reči: *propadi napona, prenosna mreža, monitoring, dalekovodi*

Abstract – *In this paper voltage sags during fault simulation at different distances are considered. We compare the simulation results with the results of real measurements. The goal is to make the simulation results at approximately the same distance from the fault location as in the real system as different as possible from the measurements in the real system. The considerations were performed on the basis of real measurements with the microprocessor protection device SIPROTEC 47SA611, while the simulation results were obtained on the basis of the transmission line model modeled in the MATLAB software package.*

Keywords: *voltage sags, transmission network, monitoring, transmission lines*

1. UVOD

Kratkotrajna podnaposka stanja nazivaju se propadi napona. Ona predstavljaju sniženje napona, odnosno smanjenje njegove efektivne vrednosti praćeno naponom oporavka nakon kratkog vremenskog perioda. U IEEE standardu 1159-1995, izraz „propad napona“ (engl. „voltage sag“ ili „voltage dip“) je definisan kao pad napona efektivne vrednosti na vrednosti između 0,1 do 0,9 r.j. u trajanju od 0,5 ciklusa do 1 min.

Pad napona tokom propada može biti usled kratkog spoja koji je prisutan u sistemu. U trenutku kada zaštitni sistem otkloni kvar kratkog spoja, napon počinje da se vraća na prvobitnu (nazivnu) vrednost [1]. U distributivnoj mreži uzroci propada, pored kvarova (kratkih spojeva), mogu biti i stavljanje pod napon velikih transformatora, kao i pokretanje (ili rasterećenje) velikih potrošača.

Trajanje propada se određuje sa vremenom otklanjanja kvarova. Kvarovi u prenosnim sistemima se obično brže uklanjaju od kvarova u distributivnim sistemima.

U prenosnim sistemima je vreme kritičnog uklanjanja kvarova prilično malo.

Takođe, prenosni sistemi normalno rade kao mreža, zahtevajući zaštitu na daljinu ili diferencijalnu zaštitu, koja omogućava brzo ukljanjanje kvara [2].

2. PROPADI NAPONA U PRENOSNIM MREŽAMA

Jednofazno okidanje se najčešće koristi u prenosnim sistemima za održavanje sinhronosti između obe strane dalekovoda. Jednofazno okidanje retko se koristi u distribuciji ili niskonaponskim sistemima.

Struja greške i dalje teče preko neispravne struje faze. To smanjuje šansu da se greška ugasi i na taj način povećava broj pokušaja prikrivanja i broj dugih prekida. Ali ako je APU uspešno, jednofazno okidanje ima jasne prednosti u odnosu na trofazno okidanje [2].

Jednom kada se kvar ukloni, normalno stanje se vraća što je pre moguće.Stalni kvarovi kratkog spoja moraju otkloniti uzroke kvarova. Stoga, pronalaženje lokacije trajnih kvarova neophodno je da bi se uklonio uzrok kvara i ponovo pokrenulo napajanje mreže.

Problem sa kvarom obično je povezan sa dalekovodom za koji su terminalna merenja dostupna.

U ovom slučaju, cilj merenja je pronaći tačnu lokaciju greške. Zadržani napon tokom greške daje jačinu pada napona. Zavisno od tipa kvara koji uzrokuje propad, napon tokom događaja može biti jednak ili različit u tri faze.

Prema simetričnoj klasifikaciji komponenti dve vrste prepoznaju se simetrična i nesimetrična naprezanja. Trofazni kvar izaziva uravnoteženi pad napona što znači da su fazni naponi tokom greške jednaki u sve tri faze.

Za ove vrste naponskih padova postoji samo jedan fazni napon potreban za karakterizaciju veličine i faznog ugla pada.

Nesimetrična greška može prouzrokovati propade napona sa glavnim padom u jednoj ili dve faze. Jednačine za napon tokom kvara izvedene su za simetrične i nesimetrične kvarove [2].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vladimir Katić, red.prof.

3. MONITORING PROPADA NAPONA

Monitoring propada napona u EES-u može pomoći pri određivanu uzroka poremećaja u sistemu, kao i na primer identifikacija stanja koja su bila pre prekida, poremećaja u sistemu.

Monitoring kvaliteta električne energije predstavlja proces prikupljanja, obrade, analize i interpretacije merenih podataka u svrhu dobijanja nekih korisnih informacija.

Proces prikupljanja podataka vršimo kontinuiranim merenjem napona i struje u određenom vremenskom periodu - intervalu.

Procesi obrade i analize podataka tradicionalno se vrše manuelno. Međutim, rapidan napredak oblasti za prenos i obradu signala, razvitak inteligentnih sistema kao i njihova implementacija nam omogućuje automatsku analizu.

Propade napona možemo okarakterisati dijagramom efektivne vrednosti napona tokom vremena dok se prekid napona može definisati vremenskim periodom [3].

3.1 Merni uređaji

U okviru JP EMS-a već se koriste tri uređaja tipa ELSPEC BLACKBOX SERIJE G4500, slika 3.1. U pitanju su uređaji klase A (CLASS A prema standardu IEC 61000-41-30).

Slika 3.1. *Uređaj-ELSPEC BLACKBOX G4500 [6]*

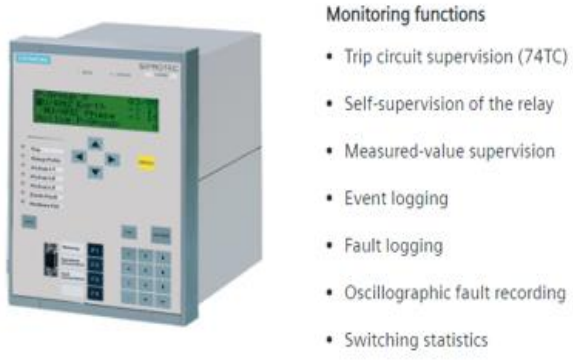
Ovakvi uređaji sasvim zadovoljavaju potrebe i tehničke specifikacije za povremeni i/ili kontinualni monitoring kvaliteta napona i struje.

Njihova osnovna prednost je u tome što imaju visoku frekvenciju odabiranja struja i napona tako da nije potrebno definisati nikakve pragove snimanja (eng trigger) već je ulazne veličine moguće beležiti u talasnom obliku a zatim naknadno vršiti odabir željenih rezultata merenja, načine sortiranja, poređenja, itd. [3].

SIPROTEC 4 7SA6 digitalna zaštita na daljinu opremljena je snažnim mikroračunarskim sistemom, slika 3.2. On omogućava potpunu numeričku obradu svih funkcija u uređaju, od dobijanja izmerenih vrednosti do izlaznih komandi na prekidaču.

Dostupnost uređaja uveliko se povećava nadgledanjem unutrašnjeg merenja, sklopova, pomoćnog napajanja, hardvera i softvera.

Nadgledaju se sekundarni krugovi transformatora, struje i naponi transformatora korišćenjem tehnika provere sažimanja i simetrije. Vrši se nadzor strujnog kruga. Proverava se impedansa opterećenja, izmerenog smera i redosleda faza. Nadgleda se i prenos signala opcionog digitalnog komunikacijskog puta [7].



Slika 3.2. *Uređaj SIPROTEC 4-7SA6 [7]*

3.2. Softveri za snimanje propada napona

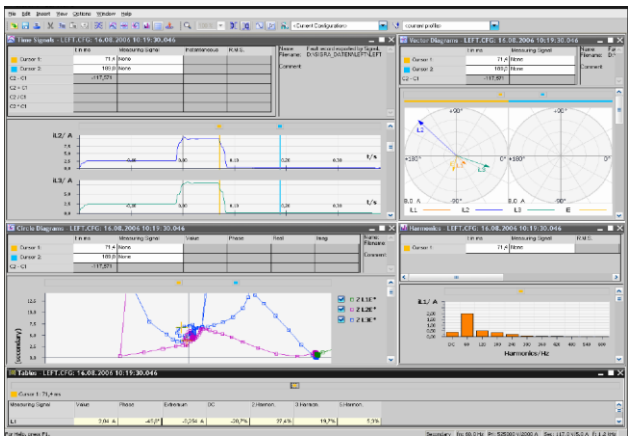
PQSCADA Sapphire nabavlja, obrađuje i čuva snimljene podatke sa bilo kojeg uređaja za snimanje kroz različite komunikacijske protokole i formate datoteka.

PQSCADA Sapphire će automatski izračunati i pohraniti približno 5000 parametara snage, uključujući pojedinačne harmonike od stečenih signala talasnog oblika.

Softver nudi jedinstveni set grafikona i izveštaja koji omogućavaju korisniku da planira, kontroliše i donosi odluke zasnovane na podacima. Omogućava pregled električne mreže i mernih uređaja na geografskoj mapi. Prikuplja komprimovane snimljene neobrađene podatke talasnog oblika koristeći ELSPEC PQZ patentirani protokol [5].

Slika 3.3 *Izgled snimka na interfejsu- PQSCADA [5]*

SIGRA je aplikacioni program koji podržava analizu događaja grešaka u mreži. Nudi grafički prikaz podataka snimljenih tokom događaja kvara i koristi izmerene vrednosti za izračunavanje dodatnih varijabli, kao što je impedansa, izlazi ili efektivne vrednosti koje olakšavaju analizu zapisa grešaka. Zatim, promenljive mogu da se predstave u bilo kom od dijagrama prikaza pokazanih na slici 3.4.
Primeri prikaza su vremenski signali ili vektorski dijagrami, dijagram krugova, harmonika, lokacije grešaka, tabele.



Slika 3.4 *Snimak kvara-različiti dijagrami i tabele [4]*

Možete izabrati da se u raznim prikazima prikažu signali greške. Primarne vrednosti ili sekundarne vrednosti, bez obzira na metodu zapisivanja izmerenih vrednosti u registratoru grešaka. Pored stvarnih signala moguće je prikazati i osnovnu komponentu kao i harmonike svakog signala i na taj način još bolje prepoznati posebne karakteristike signalnih krivih [7].

4. MODELOVANJE I SIMULACIJE

Veliki broj pojava u EES-u su posledica prekidačkih operacija u postrojenjima. Prekidačke operacije mogu biti uzrokovane pojavom kratkog spoja (isključenje kratkog spoja) ili uključenje/isključenje neopterećenih dalekovoda. Pri proračunu prelaznih pojava potrebno je izbor modela dalekovoda prilagoditi vrsti prelazne pojave koja se analizira.

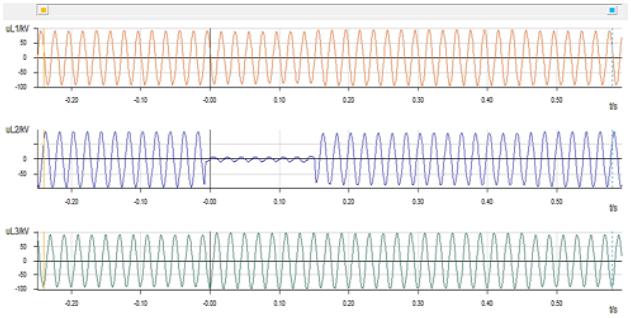
Dalekovodi imaju parametre, koji su po prirodi raspodeljeni, za razliku od transformatora i generatora čiji se parametri mogu smatrati koncentrisanim. EES u uslovima prelaznih stanja može biti podvrgnut naprezanjima usled napona i struja koji imaju širok frekvencijski spektar, od pogonske frekvencije do frekvencija reda 100kHz i više. Pri ovakvim frekvencijama parametri dalekovoda i zemlje kao povratnog puta imaju vrednosti koje se menjaju s frekvencijom [9].

Modelovanje dalekovoda je urađeno uz korišćenje standardnog modela dugačkog voda u prelaznim stanjima, a parametri su dobijeni od EMS-a [9]. Korišćen je Matlab softver, a radi verifikacije modela rezulatati simulacija su upoređeni sa rezultatima merenja. Pošto se radi o opšte poznatom modelu, ovde neće biti dat detaljniji opis.

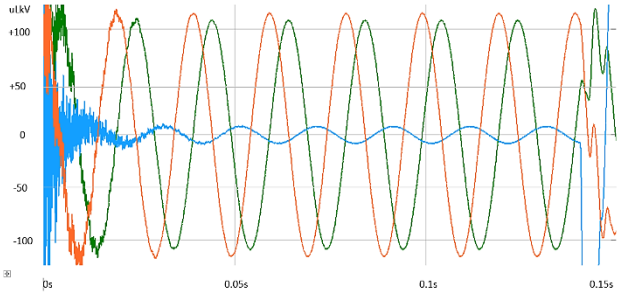
Nisu uzeti u obzir rezultati merenja sa uređaja ELSPEC BLACKBOX G4500 zato što se on nalazi poprilično duboko u mreži i uticaj prenosne mreže i njenih delova je veliki pa se rezultati dosta razlikuju od rezultata dobijenih iz mikroprocesorske zaštite SIPRTOEC 4 7SA611, kao i modela dalekovoda i mesta kvara na dalekovodu, modelovanog u MATLAB-u.

Analiza rezultata vršena je za dva dalekovoda, dok je u prvom redu prikazan samo jedan dalekovod (124/4) jer nema značajnijih razlika između dva dalekovoda.

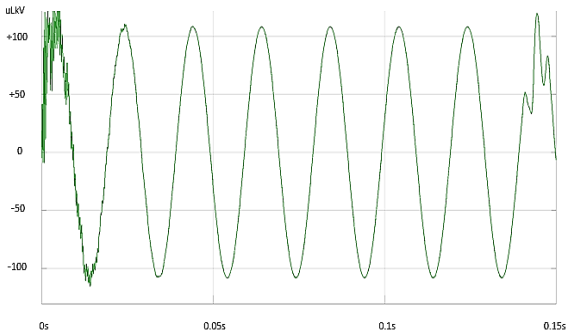
- 124/4 na pravcu između trafostanice Sremska Mitrovica 1 (110/20kV) i trafostanice Pećinci (110/20kV).
- 124/5 na pravcu između trafostanice Šabac 3 (220/110kV) i trafostanice Pećinci (110/20kV)



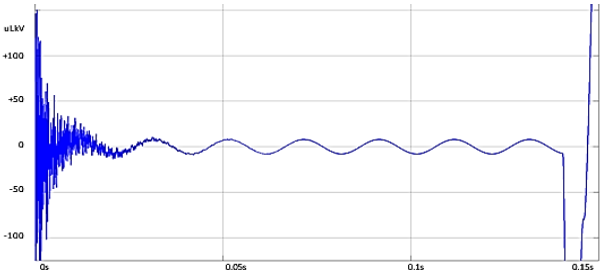
Slika 3.5 Fazni naponi (primarne vrednosti) tokom kvara u fazi L2-MP zaštita



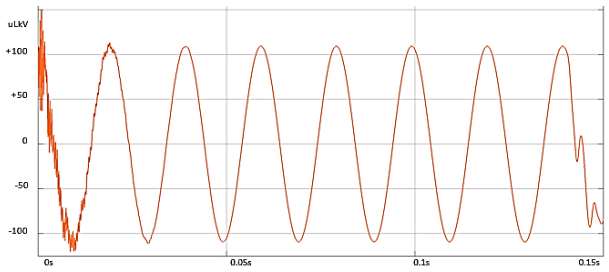
Slika 3.6 Naponi faze L1, L2, L3 tokom kvara u fazi L2 na 0.6km od TS Pećinci – simulacija u Matlabu



Slika 3.7 *Napon faze L1 tokom kvara u fazi L2 na 0.6km od TS Pećinci-Matlab*



Slika 3.8 *Napon faze L2 tokom kvara u fazi L2 na 0.6km od TS Pećinci – simulacija u Matlabu*



Slika 3.9 *Napon faze L3 tokom kvara u fazi L2 na 0.6km od TS Pećinci - simulacija u Matlabu*

Tabela 3.1 Propadi napona na DV 124/4 po udaljenosti do mesta kvara u fazi L2

Uređaj7SA611 i simulacije u MATALAB-u		7SA611	MATLAB	
L1	0.6km	-3.85%	0.6km	-3.2%
L2		-91.49%		-94.35%
L3		+8.02%		+12.5%
L1	0.6km	-3.85%	15.6km	-5.8%
L2		-91.49%		-75.5%
L3		+8.02%		+11.5%
L1	0.6km	-3.85%	25.6km	-7.8%
L2		-91.49%		-62.5%
L3		+8.02%		+10%

5. ZAKLJUČAK

Problematični propadi napona potiču od kvarova na prenosnoj mreži, odnosno na napojnim dalekovodima 110kV br. 124/4 i 124/5 koji ulaze u TS 110/20kV Pećinci.Korisnik DS Coficab iz Deča, obratio se ED Ruma sa žalbom na kvalitet električne energije (žalba se odnosi na propade napona i kratkotrajne prekide).Problemi na dalekovodima (prolazni kvarovi, uglavnom, ispadi...) su se javili u maju, nastavili u junu, u julu su kulminirali i nastavili se u avgustu.Pogone pokreću elektromotori sa frekventnom regulacijom. Propadi napona na pojedinim linijama prouzrokuju razilaženje električnog vratila, dovode do razlike brzine Δv što dovodi do kidanja provodnika.

Simulacijom kvarova na različitim udaljenostima možemo primetiti propagaciju propada napona kroz deo mreže, dubina propada napona opada sa približavanjem izvoru. Na osnovu dobijenih rezultata simulacije sa različitim udaljenostima mesta kvara od TS Pećinci vidimo da nam se najviše poklapaju rezultati simuliranog kvara na istoj udaljenosti kao i u realnom sistemu (0.6km na oba dalekovoda).

U periodu od maja ove godine do 16.08. na dalekovodu 110kV br. 124/4 bilo je 14 prolaznih kvarova i to: faza „0“ 5 puta, faza „4“ 4 puta i faza „8“ 5 puta. Na dalekovodu 110kV br. 124/5 koji je bliži deponiji bilo je 22 prolazna kvara i to faza „0“ 3 puta, faza „4“ 12 puta i faza „8“ 7 puta. Najugroženija je faza „4“, gornja faza, ukupno 16 puta na oba dalekovoda.

Operator(EMS AD) je u skladu sa analizom događaja zaključcima postavio mrežni analizator u DVP 110kV br. 124/4 u TS 110/20kV Pećinci 07.08.2018. godine i na mestu primopredaje, odnosno mernom mestu u ZTS 20/0,4 kV Coficab u Deču.[8]

6. LITERATURA

[1] V.Katić, A.Tokić, T.Konjić „Kvalitet električne energije“, Novi Sad: FTN, 2007.

[2] Math H.J.Bollen „Understanding power quality problems“,

[3] „Studija br.113021: „Monitoring Kvaliteta elektricne energije I faza“, INT, Beograd, 2014.

[4] SIEMENS „SIPROTEC, Fault record analysys, SIGRA V4.58-Manual“

[5] ELSPEC „PQSCADA Sapphire-user manual“

[6] M.Hrubenja, „Photography“.

[7] SIEMENS „SIPROTEC, Distance protection 7SA6 V4.61 nad higher“

[8] „Podaci o izveštaju kvaliteta električne energije kod korisnika DS COFICAB iz Deča“

[9] Ljupko Teklić „Modeliranje dalekovoda u proračunima prelaznih pojava“, CIGRE, 2011.

Kratka biografija:

Miron Hrubenja rođen je u Novom Sadu 1990. godine. Srednju školu gimnaziju "Svetozar Marković", završio je u Novom Sadu, 2009. godine. Fakultet tehničkih nauka, studijski program Energetika, elektronika i telekomunikacije upisao je školske 2009/2010. godine i diplomirao na osnovnim akademskim studijama 2018. godine. Upisao master akademske studije 2019. godine , smer Elektroenergetika - distribuirani elektroenergetski resursi i diplomirao 2020. godine.

Vladimir Katić je rođen 1954. godine u Novom Sadu. Doktorirao je na Univerzitetu u Beogradu 1991. godine. Od 2002. godine je redovni profesor Univerziteta u Novom Sadu. Oblasti interesovanja su energetska elektronika, obnovljivi izvori električne energije, električna vozila i kvalitet električne energije.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.3
DOI: https://doi.org/10.24867/10BE06Negulic

JEDNO REŠENJE PROGRAMSKE PODRŠKE ZA BEZBEDNO ČUVANJE PODATAKA U AUTOMOBILSKOJ INDUSTRIJI

ONE SOLUTION FOR PERSISTENT DATA STORAGE IN AUTOMOTIVE INDUSTRY

Ivan Negulić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu biće opisan dizajn i implementacija rešenja generičkog bezbednog sistema za trajno čuvanje podataka u programskom jeziku C, u AUTOSAR okruženju. Takođe, implementirane su metode zaštite podataka, algoritmima simetričnog, AES i asimetričnog RSA kriptovanja.*

Ključne reči: *Generički bezbedan sistem, kriptografija*

Abstract – *This paper will describe the design and implementation of a generic secure system solution for persistent data storage in the C programming language, in the AUTOSAR environment. Also, data protection methods, symmetric, AES and asymmetric, RSA, encryption algorithms have been implemented.*

Keywords: *Generic secure system, cryptography*

1. UVOD

Svakodnevna upotreba vozila zahteva unapređenje postojećih usluga, u smislu bezbednosti, sigurnosti i udobnosti vožnje, obezbeđivanju multimedijalnih sadržaja i slično. Sve ovo zahteva od kompanija koje se bave proizvodnjom automobilske opreme i vozila da stalno unapređuju svoje proizvode i usluge koje nude na svojim vozilima i uređajima. Stalno unapređenje fizičke arhitekture računara i sistemske programske podrške, kao i sistema zasnovanih na računaru obezbeđuje konstantan razvoj funkcionalnosti, različitih nivoa kompleksnosti, kao sto su elektronska kontrola stabilnosti, preciznije doziranje smeše za sagorevanje u cilindrima motora sa unutrašnjim sagorevanjem, koje obezbeđuje smanjenje potrošnje goriva, praćenje potrošnje električne energije baterija u vozilima na električni pogon, pa sve do auto pilota i autonomne vožnje. Računarski sistemi koji se koriste u automobilskoj industriji, odnosno u automobilskim sistemima, po performansama ne zaostaju za savremenim PC računarima. Modernim automobilima se postavljaju visoki zahtevi vezani za sigurnost, ekonomičnost i udobnost [1]. Jedan od najvećih izazova sa inženjerske tačke gledišta predstavlja prenos podataka, skladištenje i manipulacija podacima.

Trajno održanje sadržaja memorije u automobilskim sistemima može se posmatrati kao funkcija za praćenje pređene kilometraže, prosečne potrošnje goriva ili prosečne potrošnje električne energije u vozilima na električni pogon, pamćenje omiljenih radio stanica položaj sedišta, volana, retrovizora u zavisnosti od trenutnog vozača, do bezbednosnih kritičnih podataka, kao što su stanje elektronske parking kočnice, slike sa kamere, radara, i ostalih senzora u trenucima pre udesa automobila (crna kutija).

Razvoj programske podrške u automobilskim sistemima zahteva razvoj po najvišim bezbednosnim standardima, radi obezbeđivanja zahtevanog ponašanja sistema, ranog otkrivanja kvarova, smanjujući mogućnost nepredviđenog ponašanja sistema na prihvatljiv nivo.

2. TEORIJSKE OSNOVE

Softverski model - Softverski proces je skup aktivnosti i dobijenih rezultata čiji je cilj razvoj i evolucija softvera. Osnovne aktivnosti unutar softverskog procesa su: specifikacija, modelovanje, implementacija, verifikacija, i validacija. Softverski model je idealizovan prikaz softverskog procesa, kojim se određuje željeni način odvijanja i međusobnog povezivanja aktivnosti, odnosno model je opis sistema napisan korišćenjem jezika koji podržava automatsku interpretaciju od strane računara. Svi softverski modeli su okarakterisani sledećim aktivnostima:

- Specifikacija – analiza zahteva korisnika, utvrđivanje šta softver treba da radi.
- Modelovanje – definiše se sistem, način rada komponenti, komunikacija između komponenti, odnosno uprošćeno, projektuje se rešenje koje određuje kako će softver raditi.
- Implementacija – modelovani sistem realizuje se pomoću programskih jezika i alata.
- Verifikacija i validacija – proverava se da li softver radi prema zadatoj specifikaciji korisnika. Ovaj korak se svodi na testiranje, ali postoje i druge metode.
- Održavanje, evolucija – nakon uvođenja u upotrebu, softver se dalje popravlja, menja, nadograđuje, sve u skladu sa zahtevima korisnika.

2.1. Modeli i metamodeli

Model je apstrakcija sistema koji se često koristi za zamenu sistema koji se proučava. Model predstavlja delimični i pojednostavljeni prikaz sistema, pa je obično potrebno stvaranje više modela da bi se jasnije predstavio i razumeo sistem koji se proučava. Modeliranje je dobro poznata tehnika usvojena od strane inženjerskih oblasti, kao i drugih oblasti poput fizike, matematike, ekonomije i drugih. Ovde se fokusiramo na modele u softverskom inženjerstvu i informacionim sistemima. Ovde ćemo predstaviti suštinske pojmove na kojima se zasniva inženjersko upravljanje modelima (MDE – Model-Driven

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Ilija Bašičević, red. prof.

Engineering), odnosno koncept sistema, modela, meta-modela i njihovih odnosa. U kontekstu MDE, definišemo "sistem kao generički koncept za označavanje softverske aplikacije, softverske platforme ili bilo kog drugom softverskog artefakta". Osnovna ideja je razmatranje modela na visokom nivou apstrakcije, bez detalja implementacije. Podizanjem nivoa apstrakcije smanjuje se složenost modela. Povećanje produktivnosti se postiže maksimiziranjem kompatibilnosti između sistema, korišćenjem standardizovanih modela, uprošćavanjem procesa dizajniranja, olakšavanjem komunikacije između različitih timova koji rade na sistemu, korišćenjem standardizovane terminologije, i korišćenjem tehnika i tehnologija koje su se pokazale kao najbolje u praksi (*Best practice*). Obećavajući pristup za rešavanje problema kompleksnih fizičkih arhitektura i programskih podrški, usled nemogućnosti programskih jezika treće generacije da smanje kompleksnost, i izraze koncepte problema određenih domena, je razvoj programske podrške korišćenjem inženjerstva zasnovanog na modelima [2].

2.2. Generator koda

U računarskoj nauci, termin automatsko programiranje identifikuje vrstu računarskog programiranja u kojem neki mehanizam generiše računarski program kako bi omogućio ljudskim programerima da napišu kod na višem nivou apstrakcije [3]. Jedan od najranijih programa koji se može prepoznati kao prevodilac nazvao se Autocode. Parnas je zaključio da je "automatsko programiranje uvek bilo eufemizam za programiranje na jeziku višeg nivoa nego što je tada bio dostupan programeru" [4]. Pod generisanjem programskog koda u ovom radu misli se na programske alate koji se koriste za obradu šablona. Upotreba ovakvih alata je korisna u slučajevima gde se slični ili isti delovi koda pojavljuju više puta, ili kada se zahtevi i specifikacije često menjaju. U slučaju promene specifikacije, neophodno je samo ponovo generisati kod. Ovakav način rada znatno smanjuje vreme za pisanje koda, povećava čitljivost koda, podstiče organizaciju izvornog koda u operativno različite slojeve, povećava produktivnost.

2.3. Trajno čuvanje memorije

U računaru se memorija odnosi na uređaj koji se koristi za čuvanje informacija za trenutnu upotrebu u računaru ili srodnom hardveru [4]. Prema trajnosti podataka po nestanku napajanja prepoznajemo dve vrste memorija: promenljive (volatile) i nepromenljive (non-volatile) memorije. Nepromenljive memorije ili memorije sa stalnim sadržajem su memorije koje mogu da uskladištite informacije i po nestanku napajanja.

2.4. CRC

Ciklična provera redundantnosti (CRC – Cyclic Redudance Check) je algoritam za otkrivanje grešaka koji se obično koristi u digitalnim mrežama i uređajima za skladištenje podataka, da bi se otkrile slučajne promene neobrađenih podataka. Blokovi podataka koji ulaze u ove sisteme dobijaju priložene kratke kontrolne vrednosti, zasnovane na ostatku deljenja polinoma sadržaja blokova podataka. Ciklična provera redundantnosti (CRC) zasniva se na deljenju u prstenu polinoma preko konačnog polja, odnosno Galovog polja GF(2). Ako posmatramo niz bitova kao $a_n a_{n-1} a_{n-1} \ldots a_1 a_0$, onda taj niz odgovara polinomu $M(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0$. Izračunavanje CRC odgovara Euklidskom deljenju polinoma nad Galovim poljem GF(2):

$$M(x) \cdot x^n = Q(x) \cdot G(x) + R(x) \qquad (1)$$

Gde je M(x) originalni polinom poruke, a G(x) je polinom generator stepena n. $M(x) \cdot x^n$ je originalna poruka na kojoj je dodato n nula na kraju, Q(x) je polinom koji nas ne zanima, a R(x) je preostali polinom koji predstavlja CRC kontrolnu sumu. Korišćenjem operacije ostatka pri deljenju, odnosno operacije modula, može se reći da je:

$$R(x)\ M(x) \cdot x^n \bmod G(x) \qquad (2)$$

Koeficijenti ostatka deljenja dodaju se na kraj poruke. Ostatak je vrednost CRC koja se zajedno sa ulaznim podacima šalje prijemniku, koji ponovnim računanjem CRC primljenih podataka, koji upoređuje izračunatu CRC sumu i primljenu CRC sumu. Ako je vrednost CRC nula, tada prilikom prenosa verovatno ne nastaje bitna greška, a u slučaju da je vrednost različita od nule u prenosu sigurno postoji greška.

2.5. AES

Napredni standard šifriranja (AES), poznat i po originalnom imenu Rijndael [5] je specifikacija za šifrovanje elektronskih podataka koju je ustanovio Nacionalni institut za standarde i tehnologiju (NIST) u 2001. godini [6].

Specifikacija algoritma

AES je zasnovan na principu dizajna poznatom kao supstituciona i permutaciona mreža i efikasan je kako u softveru, tako i u hardveru. AES je varijanta Rijndela koji ima fiksnu veličinu bloka od 128 bita i veličinu ključa od 128, 192 ili 256 bita. Veličina ključa koji se koristi za AES šifru određuje broj krugova transformacije koji pretvaraju ulaz, koji se naziva *plaintext,* u konačni izlaz, koji se naziva *chipertext*. Broj rundi je:

- 10 rundi za 128-bitni ključ,
- 12 rundi za 192-bitni ključ,
- 14 rundi za 256-bitni ključ.

Svaki krug se sastoji od nekoliko koraka obrade, uključujući i ona koji zavisi od samog šifriranja. Skup obrnutih zaokruživanja primenjuje se za transformisanje šifriranog teksta nazad u originalni tekst koristeći isti ključ za šifriranje.

AES algoritam je blok-šifra koja koristi isti enkripcijski ključ u procesu šifrovanja i dešifrovanja podataka koja se primenjuje u nekoliko iteracija (rundi). AES ima 10 rundi (može imati 10, 12 ili 14 rundi u zavisnosti od dužine ključa), a svaka runda se sastoji od 4 operacije koje se primenjuje nad elementima matrice dimenzije 4x4 bajta, a to su:

1. SubBytes (nelinearna zamena bajtova pomoću supstitucione tabele).
2. ShiftRows (promena mesta unutar bajtova iste kolone).
3. MixColumns (transformacija bajtova unutar iste kolone).
4. AddRoundKey (inicijalno dodavanje ključa).

2.6. RSA

RSA (Rivest – Shamir – Adleman) je jedan od prvih kriptosistema sa javnim ključem i široko se koristi za siguran prenos podataka. U takvom kriptosistemu ključ za šifrovanje je javni i razlikuje se od ključa za dešifrovanje koji se čuva u tajnosti (privatni). U RSA se ta asimetrija zasniva na praktičnoj poteškoći faktorizacije proizvoda dva velika osnovna broja [7]. Kriptovanje se vrši po formuli:

$$C \equiv M^e \ (mod\ n) \qquad (3)$$

Dok se dekriptovanje vrši po formuli:

$$M \equiv C^d \ (mod\ n) \qquad (4)$$

Generisanje ključeva se vrši na sledeći način:

1. Biraju se dva cela prosta broja p i q.
   - Iz sigurnosnih razloga, celi brojevi p i q se biraju nasumično,
   - p i q se čuvaju u tajnosti.
2. Izračunava se
$$n = pq \qquad (5)$$
   - n se koristi kao modul i za javne i za privatne ključeve, njegova dužina je obično izražena u bitovima,
   - n se šalje kao deo javnog ključa.
3. Izračunava se Ojlerova funkcija
$$\varphi(n) = (p-1)*(q-1) \qquad (6)$$
   - $\varphi(n)$ se čuva u tajnosti.
4. Bira se ceo broj e tako da $1 < e < \varphi(n)$ i da zadovoljava $NZD(e, \varphi(n)) = 1$
   - e se šalje kao deo javnog ključa.
5. Izračunava se d kao
$$d \equiv e^{-1} \ (mod\ \varphi(n)) \qquad (7)$$
   - d se čuva kao u tajnosti kao eksponent privatnog ključa.

3. KONCEPT REŠENJA

Ovo poglavlje prikazuje koncept rešenja primenjen u ovom radu. Projekat se sastoji iz dva dela: ručno pisan deo programskog koda, i generisani deo programskog koda. Ručno pisani deo implementacije sadrži implementaciju koja je nezavisna od ulaznih podataka, dok implementacija u generisanom delu direktno zavisi od ulaznih podataka. Ovakva realizacija se javlja, jer smanjuje greške u pisanju koda, koju se u ovom slučaju generiše, a takođe je primećeno dobro ponašanje u radu na komercijalnim projektima.

3.1. Negenerisani deo

Negenerisani deo će obezbeđivati funkcionalnost očuvanja blokova podataka, koristeći stalne memorije. U ovom delu će biti implementirane funkcije koje obezbeđuju čitanje i pisanje podataka glavne i pomoćne datoteke blokova podataka u stalnoj memoriji. Kako bismo obezbedili siguran proces čitanja i pisanja blokova podataka koriće se dve datoteke glavna i pomoćna datoteka za svaku softversku komponentu u sistemu. Glavna datoteka obezbeđuje čitanje i skladištenje svih blokova podataka, dok pomoćna datoteka, takođe obezbeđuje ovu funkcionalnost, s tim da će se ona koristiti u slučaju da je glavna datoteka oštećena, ili ako je utvrđeno da podaci u glavnoj datoteci nisu validni, odnosno da je integritet podataka narušen. Integritet se može narušiti u slučaju kvara stalne memorije, sistemske greške prilikom čitanja ili pisana u stalnu memoriju, ili u slučaju neovlašćenog pristupa memoriji i promeni sadržaja memorije. Bezbednost pomoćne i glavne datoteke je obezbeđena kriptografskim metodama AES i RSA zaštite podataka, a očuvanje integriteta obezbeđuje algoritam za izračunavanje CRC vrednosti, gde se proverava, nakon uspešnog dekriptovanja podataka iz datoteke, ispravnost CRC vrednosti.

3.2. Generisani deo

Generisani deo obezbeđuje funkcije koje generišu izvorni programski kod na osnovu ulaznih datoteka. U ovom slučaju ulazna datoteka je u XML formatu (eXtensible Markup Language). Ukoliko ulazna datoteka nije XML formata prijavljuje se greška, i generisanje programskog koda se prekida. Generisani deo programskog rešenja obezbeđuje funkcije trajnog čuvanja sadržaja memorije celom AUTOSAR sistemu. Funkcionalnosti koje se obezbeđuju sistemu se mogu podeliti u dve grupe: trajno čuvanje sadržaja memorije pre gašenja uređaja, i povratak sadržaja memorije po uključenju. Ovakav pristup omogućava iterativni i inkrementalni razvoj programske podrške. U slučaju promene ulaznih parametara, dovoljno je samo ponovo izgenerisati deo programske podrške koji zavisi od ulaznih parametara modela, bez potreba dodatnih ručnih izmena u programskom kodu. Na ovaj način se omogućava brz odgovor na promenu ulaznih podataka (XML datoteke).

4. VALIDACIJA I VERIFIKACIJA

Sastavni deo svake programske podrške su zahtevi za čestom promenom specifikacije zahteva. Specifikacija zahteva koja je na početku zahtevana sklona je čestim promenama, i samim tim potrebne su odgovarajuće promene u izvornom programskom kodu. Ove promene se zahtevaju kako bi programsko rešenje dovelo do stabilnijeg rada u praksi. U praksi česte promene, izmene procesa i dodavanje novih funkcionalnosti je čest uzrok nastanka grešaka i nepouzdanog izvršavanja programske podrške. Iz svih ovde navedenih razloga bitna stavka u razvoju programske podrške zahteva određena testiranja kako bi se utvrdila ispravnost funkcionalnosti programskog rešenja i potvrdila da zadovoljava specifikacije zahteva. Tokom izrade rešenja ispitivanje je vršeno na platformi sa Nvidia Tegra K1 procesorom[8], sa Linux distribucijom Jetson TK1 R21.5 i kernel verzijom 3.10.40 [9]. Tokom ispitivanja programskog rešenja dolazi se do zaključka da je u slučaju validnih ulaznih parametara, generisani izvorni C kod, kog generiše kod generator, ispravan i da ga je moguće kompajlirati. Na ovom rešenju je izvršeno pet grupa testova: grupa testova koja verifikuje ispravno čitanje podataka iz stalne memorije, grupa testova koja verifikuje ispravno upisivanje podataka pročitanih sa RTE

magistrale u stalnu memoriju, grupa testova koji verifikuju detekciju greške nastalu u stalnoj memoriji, grupa testova koja verifikuje ispravnost pročitanih podataka koji su kriptovani RSA algoritmom, grupa testova koja verifikuje ispravnost pročitanih podataka koji su kriptovani AES algoritmom. Svih pet grupa testova je prikazala da testovi u potpunosti zadovoljavaju zadatu specifikaciju.

5. ZAKLJUČAK

Prikazano rešenje programske podrške za trajno čuvanje sadržaja stalne memorije je razvijeno po AUTOSAR i MISRA C:2004 standardima, i se može lako integrisati u AUTOSAR projekat, ili kao deo posebnog ECU-a. Rešenje je moguće nadograditi dodatnim funkcionalnostima u zavisnosti od potrebe. Dodatne funkcionalnosti je potrebno implementirati u negenerisanom delu programske podrške. Sve promene u generisanom delu zavise isključivo od promene ulaznih podataka. U slučaju razdvajanja programskog koda na generisani deo i na deo koji se piše ručno (negenerisani), omogućena je brza i laka promena izvornog koda na osnovu ulaznih parametara, kao i podrška za iterativni i inkrementalni razvoj programske podrške. Kako bi se obezbedilo generisanje koda koji je moguće kompajlirati (prevesti na mašinski jezik) generator koda proverava validnost ulaznih podataka i parametara. Razvoj ovog rešenja u budućnosti može ići u pravcu implementacije dodatnih funkcija kao što su dijagnostičke informacije o stanju memorijskog uređaja, brojač upisa u memorijski uređaj i slično. Stalni napredak u automobilskoj industriji dovodi do toga da se u modernom vozilu svake sekunde stvara i obrađuje velika količina podataka, pa je neminovno da je potreba za čuvanjem i obezbeđivanjem podataka jedna od primarnih zadataka inženjera, kao što je i demonstrirano u ovom rešenju.

6. LITERATURA

[1] M. Jovanović *et al*, "Programska implementacija sistema za prikupljanje i obradu signala sa elektronskih kontrolnih jedinica u automobilskoj mreži," Telekomunikacioni forum TELFOR, Beograd, Srbija, 2015.

[2] D.C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," IEEE Computer, vol 39, no., pp. 25-31, Feb. 2006.

[3] Ricardo Aler Mur, "Automatic Inductive Programming Archived 2016-03-04 at the Wayback Machine", *ICML 2006 Tutorial*. June 2006.

[4] D. L. Parnas. "Software Aspects of Strategic Defense Systems." *American Scientist*. November 1985.

[5] *Daemen, Joan; Rijmen, Vincent (March 9, 2003).* "AES Proposal: Rijndael" *(PDF). National Institute of Standards and Technology. p. 1.* Archived *(PDF) from the original on 5 March 2013. Retrieved 21 February 2013.*

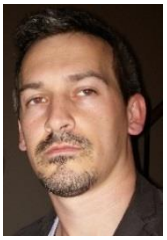
[6] "Announcing the ADVANCED ENCRYPTION STANDARD (AES)" *(PDF). Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST). November 26, 2001.* Archived *(PDF) from the original on March 12, 2017. Retrieved October 2, 2012.*

[7] *Rivest, R.; Shamir, A.; Adleman, L. (February 1978).* "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" *(PDF).* Communications of the ACM. ***21*** *(2): 120–126.* CiteSeerX 10.1.1.607.2677. doi:10.1145/359340.359342.

[8] "Tegra K1 Next-Gen Mobile Processor", 2017. http://www.nvidia.com/object/tegra-k1-processor.html.

[9] "Linux For Tegra R21.5", 2017. https://developer.nvidia.com/linux-tegra-r215.

Kratka biografija:



Ivan Negulić rođen je u Novom Sadu 1983. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva - Računarska tehnika i računarske komunikacije odbranio je 2019. god.

kontakt: ivan.negulic@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.38:620.9
DOI: https://doi.org/10.24867/10BE08Todorovic

MODELOVANJE I TESTIRANJE DEVETOFAZNE SINHRONE MAŠINE SA STALNIM MAGNETIMA NA ROTORU ZA RAD U VETROAGREGATU „HALIADE“

9 PHASE PERMANENT MAGNET SYNCHRONOUS MACHINE MODELING AND TESTING FOR OPERATION IN WINDTURBINE ”HALIADE“

Jovana Todorović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu su opisane glavne karakteristike sinhronih mašina, kao i njihova uloga u pretvaranju energije vetra u električnu energiju. Zatim je dat matematički model devetofazne sinhrone mašine sa stalnim magnetima. Na osnovu ovog modela je implementirana mašina u Typhoon HIL Control Center Schematic Editor-u. Rezultati simulacije rada mašine za date parametre mašine i režim praznog hoda su prikazani grafički. Na kraju rada je predstavljen i koncept vektorskog upravljanja ovom mašinom, odnosno upravljanja naponima statora da bi se ostvarile željene statorske struje.*

Ključne reči: *Sinhrona mašina, modelovanje i testiranje sinhrone mašine, vektorsko upravljanje sinhronom mašinom*

Abstract – *This paper describes the main characteristics of synchronous machines as well as their role in the conversion of wind power to electrical power. Then, a mathematical model of 9 phase permanent magnet synchronous machine is given. The machine is implemented in Typhoon HIL Control Center Schematic Editor based on this mathematical model. Simulation results for given parameters and no-load condition are presented graphically. At the end of this paper, the vector control concept is shown in the case of this machine, i.e. the control of the stator voltages in order to achieve desired stator currents.*

Keywords: *Synchronous machine, synchronous machine modeling and testing, vector control of synchronous machine*

1. UVOD

Veća efikasnost sinhrone mašine od asinhrone mašine predstavlja značajnu prednost, usled globalnog nedostatka energije i potrebe za štednjom.

Pri konverziji energije vetra u električnu mogu se koristiti sporohodni višepolni sinhroni generatori. Ovi generatori se vezuju direktno sa vetroturbinom, odnosno bez posredstva reduktora, što je njihova ogromna prednost.
Modelovana mašina predviđena je za rad u vetroagregatu „Haliade”, koji General Electric pravi i testira u francuskom gradu Sen Nazer. Ovo je jedan od najvećih i najsnažnijih vetroagregata na svetu. Njegova snaga je 6 MW. U drugom delu rada date su karakteristike sinhronih mašina i njihova primena u pretvaranju energije vetra u električnu. U trećem delu je dat matematički model mašine. Četvrti deo je posvećen rezultatima simulacije dok je peti deo konceptu vektorskog upravljanja ovom mašinom. Poslednja dva dela su zaključak i literatura za pisanje ovog rada.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je dr Stevan Grabić, vanr. prof.

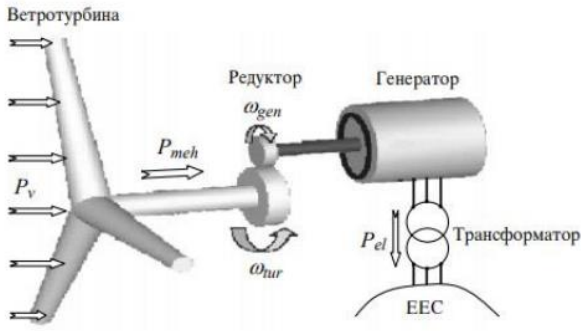
2. KARAKTERISTIKE SINHRONIH MAŠINA I ULOGA U PRETVARANJU ENERGIJE VETRA U ELEKTRIČNU

Mogući radni režimi sinhrone mašine su generatorski (proizvodnja aktivne, proizvodnja/potrošnja reaktivne energije), motorski (potrošnja aktivne, proizvodnja/potrošnja reaktivne energije) i kompenzatorski. Kompenzatorski režim predstavlja motorski prazan hod. Reaktivna energija se takođe može trošiti ili proizvoditi. U generatorskom režimu rada ulazna snaga je mehanička dok je u motorskom režimu električna.
Sinhrona mašina može raditi na krutu ili sopstvenu mrežu gde napaja individualne potrošače.
Sinhrona mašina se uglavnom koristi kao generator. U generatorskom režimu kroz provodnike rotora teče jednosmerna struja usled koje nastaje stalno magnetno polje koje preseca provodnike statora i u njima indukuje elektromotornu silu. Kod trofaznog namotaja, indukovane elektromotorne sile svake faze su jednake po amplitudi, a vremenski pomerene za jednu trećinu periode. Ako se stator optereti nekim simetričnim opterećenjem onda će kroz namotaje statora teći struje vremenski pomerene za jednu trećinu periode. Ove struje formiraju Teslino obrtno polje, a brzina obrtanja rotora je jednaka brzini obrtanja ovog polja, odakle potiče naziv sinhrone mašine.

U elektromotornim pogonima se umesto standardnih sinhronih mašina sa pobudnim namotajem koriste sinhrone mašine kod kojih su na rotor ugrađeni stalni magneti. Ovim se postiže jednostavnija konstrukcija mašine, a mašina je pouzdanija jer nema problema sa napajanjem rotorskog namotaja iz jednosmernog izvora [4].

2.1 Princip konverzije energije vetra u električnu energiju

Na slici 2.1 prikazana je pojednostavljena šema konverzije energije vetra u električnu u vetroagregatima koji su priključeni na mrežu.



Slika 2.1 -Pojednostavljena šema konverzije energije vetra u električnu energiju

Kinetička energija vetra se transformiše u mehaničku energiju pomoću vetroturbine. Mehanički reduktor prilagođava brzinu obrtanja vetroturbine zahtevanoj brzini generatora. Električni generator može biti sinhroni ili asinhroni, a može raditi sa fiksnom ili promenljivom brzinom obrtanja. Prilagođavanje napona generatora elektroenergetskom sistemu se vrši pomoću energetskog transformatora.

Osnovni nedostatak vetroagregata je korišćenje reduktora između turbine i generatora. Ova mana je izbegnuta kod koncepta sa sporohodnim višepolnim sinhronim generatorom sa stalnim magnetima koji se povezuje direktno sa vetroturbinom, a prema mreži pomoću pretvarača.

Vetroagregati sa sporohodnim sinhronim vetrogeneratorima se koriste za agregate malih snaga (300 kW), ali i za vetroagregate većih snaga. Mana im je velika cena zbog korišćenja nestandardne mašine i konvertora velike snage. Zbog velikog broja polova, generator ima veliku masu i prečnik [3].

3. MATEMATIČKI MODEL DEVETOFAZNE SINHRONE MAŠINE U ROTIRAJUĆEM dq KOORDINATNOM SISTEMU

Primenom Klarkove i Parkove transformacije dobija se matematički model devetofazne sinhrone mašine u dq domenu, koji je jednostavnija predstava mašine, pogodnija za analizu njenog rada [1].

Jednačine električnog podsistema glase:

$$[u_{dq}] = [R] * [i_{dq}] + \frac{d[\psi_{dq}]}{dt} + \omega r * \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} * [\psi_{dq}] \quad (1)$$

$$[\psi_{dq}] = [L_{dq}] * [i_{dq}] + [\psi_{PM}] \quad (2)$$

gde su: $[R] = diag[Rs \;\; Rs \;\; Rs \;\; Rs \;\; Rs \;\; Rs]$ – dijagonalna matrica termogenih otpornosti statorskih namotaja, $[u_{dq}] = [u_{dx} \;\; u_{qx} \;\; u_{dy} \;\; u_{qy} \;\; u_{dz} \;\; u_{qz}]^T$ - vektor dq napona trofaznih statorskih namotaja x, y i z, $[i_{dq}] = [i_{dx} \;\; i_{qx} \;\; i_{dy} \;\; i_{qy} \;\; i_{dz} \;\; i_{qz}]^T$ – vektor struja statorskih namotaja x, y i z u dq domenu, $[\psi_{dq}] = [\psi_{dx} \;\; \psi_{qx} \;\; \psi_{dy} \;\; \psi_{qy} \;\; \psi_{dz} \;\; \psi_{qz}]^T$ – vektor fluksnih obuhvata statorskih namotaja x, y i z u dq domenu, $[\psi_{PM}] = [\psi_{PM} \;\; 0 \;\; \psi_{PM} \;\; 0 \;\; \psi_{PM} \;\; 0]^T$ – vektor flukseva stalnih magneta na rotoru, ω_r – električna ugaona brzina rotora.

Matrica induktivnosti ima sledeći oblik:

$$[Ldq] = \begin{bmatrix} Ld & 0 & Mdxdy & Mdxqy & Mdxdz & Mdxqz \\ 0 & Lq & -Mdxqy & Mdxdy & -Mdxqz & Mdxdz \\ Mdxdy & -Mdxqy & Ld & 0 & Mdxdy & Mdxqy \\ Mdxqy & Mdxdy & 0 & Lq & -Mdxqy & Mdxdy \\ Mdxdz & -Mdxqz & Mdxdy & -Mdxqy & Ld & 0 \\ Mdxqz & Mdxdz & Mdxqy & Mdxdy & 0 & Lq \end{bmatrix} \quad (3)$$

gde su *Ld, Lq* – sopstvene induktivnosti namotaja d ili q ose odgovarajućeg sada dvofaznog namotaja, dok ostale induktivnosti predstavljaju međusobne induktivnosti između d i q, d i d ili q i q osa odgovarajućih namotaja x, y, z.

Moment konverzije glasi:

$$m_{el} = \frac{3}{2} P \left((\psi_{dx} i_{qx} - \psi_{qx} i_{dx}) + (\psi_{dy} i_{qy} - \psi_{qy} i_{dy}) + (\psi_{dz} i_{qz} - \psi_{qz} i_{dz}) \right) \quad (4)$$

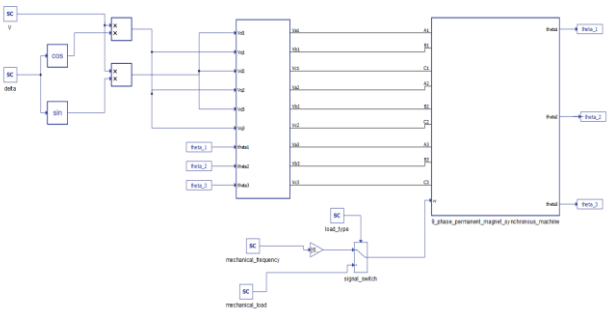
Jednačine mehaničkog podsistema mašine imaju sledeći oblik:

$$m_{el} - m_m = J_m \frac{d\omega_m}{dt} \quad (5)$$

gde su: m_{el} - moment konverzije; P - broj pari polova; m_m - mehanički moment; Jm - moment inercije; ω_m - mehanička ugaona brzina.

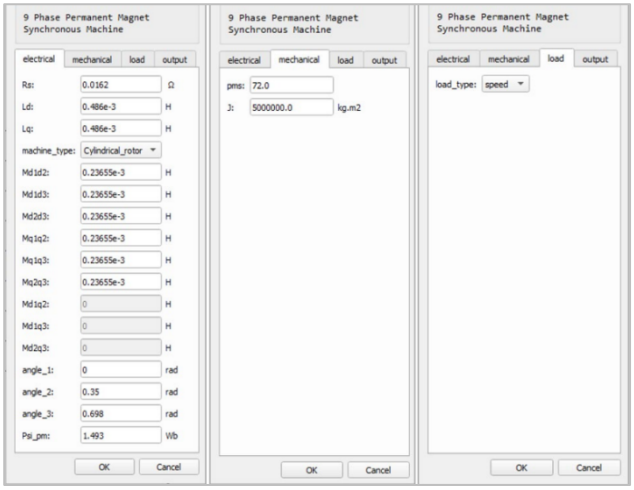
4. REZULTATI SIMULACIJE RADA MAŠINE ZA REŽIM PRAZNOG HODA

Na slici 4.1 prikazan je model mašine implementirane u Typhoon HIL Schematic Editor –u. Na SCADA se zadaju amplituda faznog napona i ugao opterećenja mašine. Na osnovu njih se računaju d i q komponente napona koje se inverznom Parkovom transformacijom transformišu u fazne napone koji se dovode na ulaze mašine.



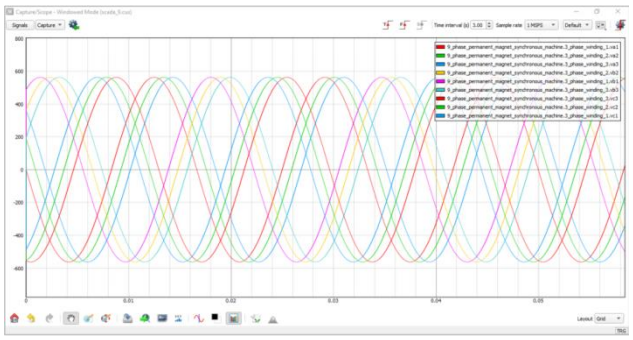
Slika 4.1 - Model mašine u Typhoon HIL Schematic Editor-u

Pored ovih napona na ulaz mašine se dovode mehanička brzina ili mehanički moment u zavisnosti od tipa regulacije mašine (po brzini/momentu). Putem maske dodeljene komponenti, koja je prikazana na slici 4.2 unose se parametri mašine.

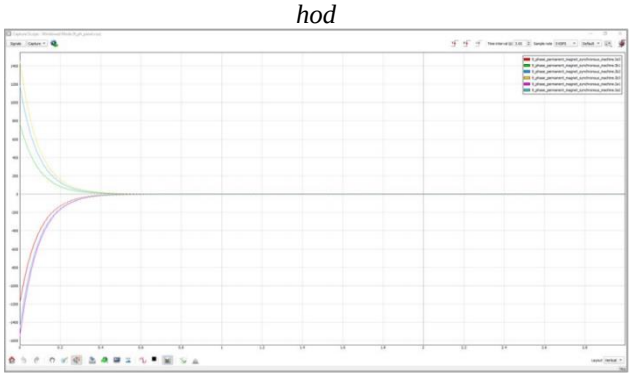


Slika 4.2 - Parametri mašine

Ogled praznog hoda izvodi se uz pretpostavku da su statorske struje približno jednake nuli. Na osnovu jednačina naponske ravnoteže (1) i fluksnih obuhvata (2), sledi da je uz tu pretpostavku napon d ose jednak nuli dok je napon q ose jednak $\omega r * \psi_{PM}$.
Ukoliko je mašina modelovana kao na slici 4.1, tj. zadaju se amplituda faznog napona (V) i ugao opterećenja (delta), sledi da je V = Vq, Vd = 0, za delta = 0. Ovi naponi se inverznom
Parkovom transformacijom transformišu u fazne napone, koji se dovode na ulaze mašine (kao tri trofazna naponska izvora).
Za date parametre mašine, nominalnu brzinu (f = 60Hz) i režim praznog hoda dobijaju se sledeći odzivi napona, struja i elektromehaničkog momenta, prikazani na slikama 4.3, 4.4 i 4.5, respektivno.

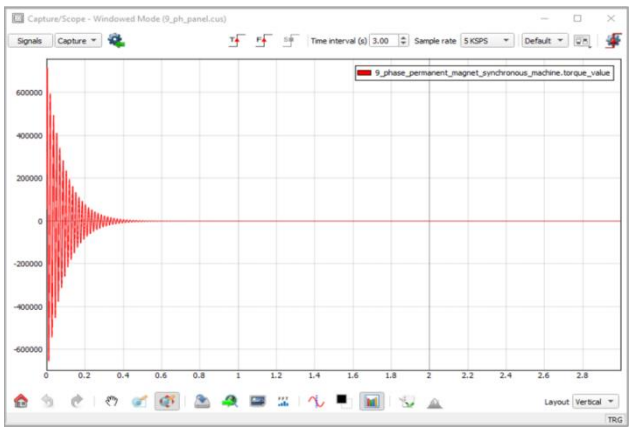


Slika 4.3 - Fazni naponi statorskih namotaja – prazan hod



Slika 4.4 - Fazne struje statorskih namotaja – prazan hod

Sa prikazanih rezultata vidi se da struje i razvijeni moment nakon 0.4 s dobijaju nulte vrednosti što je i očekivano za režim praznog hoda.



Slika 4.5 - Elektromehanički moment mašine – prazan hod

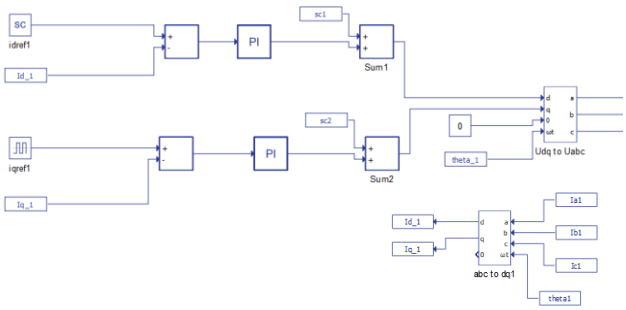
5. VEKTORSKO UPRAVLJANJE DEVETOFAZNOM SINHRONOM MAŠINOM SA STALNIM MAGNETIMA

Vektorsko upravljanje je koncept koji se pojavljuje 1960-ih godina, ali glavni razvoj i primena vektorskog upravljanja počela je pojavom bržih i jeftinijih mikroprocesora.

Koncept vektorskog upravljanja se zasniva na pronalaženju dve veličine kojima će se nezavisno upravljati fluksom i momentom.

Podužna komponenta statorske struje upravlja fluksom dok poprečna upravlja momentom. Regulaciju vektora struje u dq domenu vrše dva nezavisna strujna regulatora. Nezavisan rad strujnih regulatora je ostvaren zahvaljujući dekuplovanju, odnosno eliminisanju komponenata koje ne zavise od d komponente struje iz jednačine naponske ravnoteže d ose. Takođe i iz jednačina naponske ravnoteže q ose eliminišu se komponente koje ne zavise od q komponente vektora struje. Eliminisane komponente se dodaju na izlaz PI regulatora struje [2].

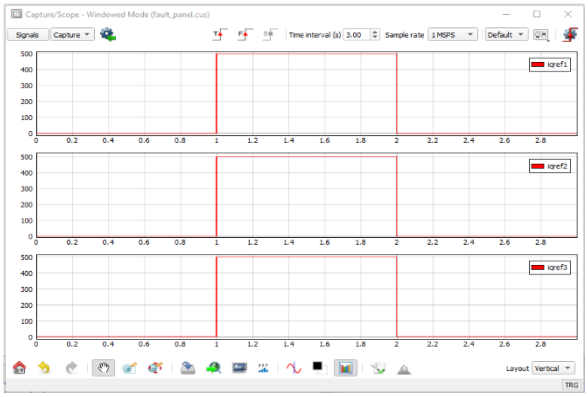
Na slici 5.1 prikazana je regulacija d i q komponenti struje jednog statorskog namotaja devetofazne sinhrone mašine u Typhoon HIL Schematic Editor-u. Analogno se vrši i regulacija struja ostalih namotaja.



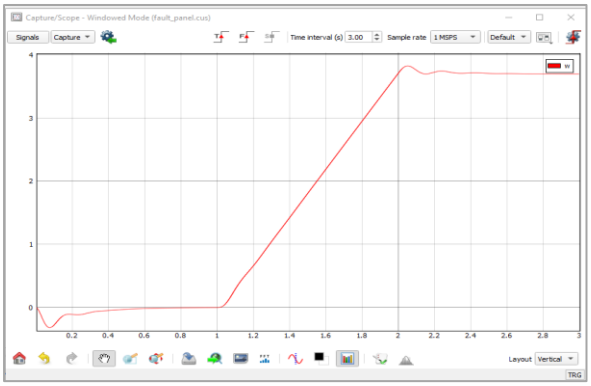
Slika 5.1 – Regulacija d i q komponente struje jednog statorskog namotaja

Referenca struje d ose se drži na nuli, da bi se ostvarilo upravljanje momentom preko struje q ose. Na izlaze regulatora se dodaju dekuplujući članovi. Regulatori upravljaju d i q komponentama napona tako da se razlike između referentnih i merenih vrednosti d i q komponenti struja svedu na nulu. Naponi se iz dq domena transformišu u abc domen i dovode na mašinu.

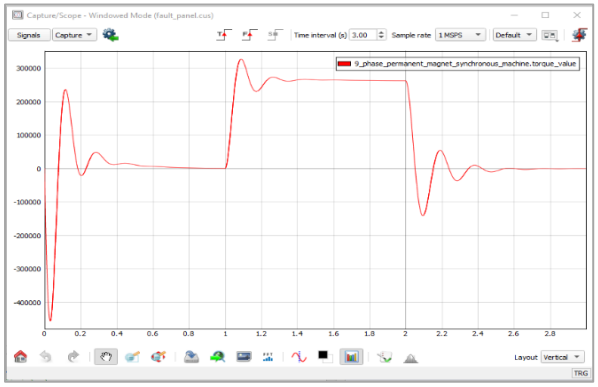
Kod vektorski upravljane devetofazne sinhrone mašine umesto dva regulatora kao kod trofazne sinhrone mašine, postoji 6 regulatora, zbog postojanja tri trofazna statorska namotaja. Dekuplujući članovi se dobijaju iz jednačina naponske ravnoteže (1) i fluksnih obuhvata (2). Princip je isti kao kod trofazne mašine. Na slici 5.3 dat je odziv brzine mašine za referentne vrednosti struja $i_d = 0$ i i_q vrednosti na slici 5.2. Na slici 5.4 prikazan je elektromehanički moment vektorski upravljane mašine za date referentne struje.



Slika 5.2 - Referentne vrednosti struja q ose

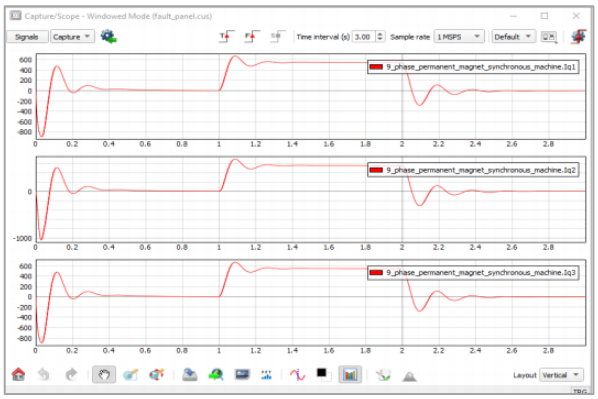


Slika 5.3 - Odziv brzine vektorski upravljane devetofazne sinhrone mašine



Slika 5.4 -Elektromehanički moment vektorski upravljane devetofazne sinhrone mašine

Kao što se vidi iz dobijenih rezultata, brzina se zadržava na nuli dok je referenca q struje na nuli, kad referenca q struje promeni vrednost na 500A, brzina počinje da raste, da bi se zaustavila na nekoj vrednosti padom reference q struje ponovo na nulu. Na slici 5.5 prikazane su merene vrednosti struja q ose.



Slika 5.5 - Merene vrednosti struja q ose vektorski upravljane devetofazne sinhrone mašine

Sa slika 5.4 i 5.5 se vidi da elektromehanički moment prati odziv q struje, jer smo referencu d struje postavili na nulu.

6. ZAKLJUČAK

Sporohodni višepolni sinhroni generator sa permanentnim magnetima rešava glavni nedostatak vetroagregata – korišćenje reduktora između turbine i generatora.
Rezultati dobijeni modelovanjem ove mašine u Typhoon HIL Schematic Editoru-u su u skladu sa očekivanim ponašanjem sinhronih mašina poznatim iz dosadašnje teorije i praktične primene.
Vektorskim upravljanjem ovom mašinom postignuta je kontorola napona d i q ose tako da stvarne vrednosti struja tih osa prate referentne. U daljim analizama rada ove mašine bi se moglo uzeti u obzir modelovanje gubitaka i uticaj zasićenja magnetnog kola.

7. LITERATURA

[1] VladanVučković, *Opšta teorija električnih mašina*, Nauka Beograd, 1992.

[2] Darko P. Marčetić, *Mikroprocesorsko upravljanje energetskim pretvaračima*, FTN Izdavaštvo, 2014.

[3] A. Josifović, *Vetroturbine*, Diplomski rad, Mašinski fakultet, Beograd

[4] Branko Mitraković, *Sinhrone mašine*, Naučna knjiga, Beograd, 1983.

Kratka biografija:

Jovana Todorović je rođena u Novom Sadu 1995. godine. Osnovne studije završila je na Fakultetu tehničkih nauka 2018. godine iz oblasti Elektrotehnike i računarstva – Elektroenergetski sistemi. Master rad je odbranila 2020. godine, na istom fakultetu, na smeru Energetika, elektronika i telekomunikacije – Energetska elektronika i električne mašine.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 656.8
DOI: https://doi.org/10.24867/10BE09Malencic

PRIMENA ETHEREUM BLOKČEJN PLATFORME ZA RAZVOJ DECENTRALIZOVANE APLIKACIJE ZA GLASANJE

APPLICATION OF THE ETHEREUM BLOCKCHAIN PLATFORM FOR THE DEVELOPMENT OF A DECENTRALIZED VOTING SYSTEM

Nikola Malenčić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu će biti predstavljene mogućnosti primene blokčejn tehnologije za podršku sistemima za glasanje i opisan razvoj jednog takvog sistema u vidu aplikacije na Ethereum blokčejn platformi. Aplikacija je implementirana na distribuiran i decentralizovan način kako bi se ispitale prednosti i mane softverskog rešenja ovog tipa.*
Ključne reči: *distribuirani sistemi, blokčejn, pametni ugovori.*
Abstract – *This work will present the possibilities for the application of blockchain technology in supporting voting systems and describe the development of such a system using the example of an Ethereum blockchain platform application. The application is implemented in a distributed and decentralized fashion, so as to explore the positive and negative aspects of a software solution of this type.*

Key words: *distributed systems, blockchain, smart contracts.*

1. UVOD

Tema ovog rada jeste implementacija softverskog rešenja koje predstavlja decentralizovanu aplikaciju za glasanje. Ideja je da se uporede prednosti i mane centralizovanih i decentralizovanih sistema kako bi se ustanovilo koji sistem više vredi razviti u različitim situacijama. U daljem tekstu uvoda će biti dat kratak pregled odnosa centralizovanih i decentralizovanih sistema. U drugom poglavlju će biti dat detaljan opis distribuiranih sistema kao celine. Treće poglavlje se bavi blokčejnom i tehnologijama distribuirane glavne knjige, dok je u četvrtom fokus na Ethereum-u i decentralizovanim aplikacijama [6]. U petom poglavlju su opisane korišćene tehnologije i alati, dok šesto i sedmo poglavlje predstavljaju opis rešenja i zaključak,

Centralizovani sistemi su tipično jednostavniji za implementaciju nego decentralizovani. Softver kod decentralizovanih sistema je veoma složen zbog brojnih problema konzistentnosti sistema. Decentralizovani sistem mora krajnjem korisniku da odaje utisak da intereaguje sa jednim koherentnim sistemom. Potrebni su protokoli koji će svakom članu sistema (čvoru) dati jedinstvenu spregu ka sistemu.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dušan Gajić, docent.

Takođe, u decentralizovanom okruženju, više čvorova mogu istovremeno da pristupaju jednom resursu, što znači da protokoli moraju da obezbede i konzistentnost podataka.

Decentralizovani sistemi se dobro skaliraju. Najrazličitiji uređaji koji implementiraju protokole sistema mogu lako da se pridruže, dok kod centralizovanih sistema to ne mora uvek biti slučaj. Uređaji koji učestvuju u sistemu mogu biti rasuti po celom svetu ali komuniciraju sa lokalnim uređajima kako ne bi bilo preteranog čekanja. Takođe, kod centralizovanih sistema povećanje broja korisnika može dovesti do toga da centralni serveri ne mogu da opsluže sve klijente, dok kod decentralizovanih sistema nema ovoga tipa problema.

Sa strane performansi pojedinačnih uređaja, kod decentralizovanih sistema se može javiti slabije iskorišćenje samih hardverskih jedinica. Ovo je zato što se tipično kod ovakvih sistema javlja mnogo komunikacije između čvorova koja nije vezana za samo rešavanje problema. Ova komunikacija služi kako bi se postigla konzistentnost i koherentnost sistema.

Sistemi sa centralizovanom arhitekturom su podložni otkazima. Kod ovakvih arhitektura je dovoljan mali kvar da paralizuje rad sistema. U distribuiranom okruženju sistem nastavlja da radi i ako neki čvorovi otkažu.

Bezbednost sistema je zanimljiva tema jer ne postoji očigledna prednost centralizovanog ili decentralizovanog načina rada.

S jedne strane, centralizovane sisteme je lakše osigurati jer je potrebno voditi računa samo o jednom kritičnom delu sistema, dok je kod drugih potrebno voditi računa o ponašanju svih čvorova sistema. Sa najnovijim protokolima neka decentralizovana rešenja uspevaju da nude veliku bezbednost, po nekim čak i bolju nego što nude centralizovani sistemi.

Jedna od najaktuelnijih tema u modernom dobu je privatnost. Popularnost decentralizovanih rešenja u velikoj meri proističe kao reakcija na monopol i neetičke postupke velikih tehnoloških kompanija u 21. veku. Ljudi gube poverenje u ove kompanije jer smatraju da im podaci nisu adekvantno zaštićeni i da se ti podaci koriste protiv njihovih interesa.

Zbog toga na svetsku scenu nastupaju velike platforme koje su bazirane na distribuiranim tehnologijama koje garantuju sigurnost i privatnost svih korisnika koji deluju na njima. Dakle, može se reći da decentralizovani sistemi nude veću zaštitu ličnih podataka nego centralizovani sistemi.

2. DISTRIBUIRANI SISTEMI

Distribuirani sistem je kolekcija autonomnih računskih elemenata koja se svojim korisnicima prikazuje kao jedan koherentni sistem [2]. Svaki pojedinačni računski element se naziva čvor. Čvorovi mogu biti ili hardverske jedinice ili softverski procesi. Moderni distribuirani sistemi se često sastoje od najrazličitijih vrsta čvorova, od računara visokih performansi do najsitnijih senzor uređaja. Osnovni princip je da su čvorovi u mogućnosti da deluju nezavisno, ali ukoliko se oni ignorišu međusobno onda nema smisla da se postavljaju u isti distribuirani sistem. U praksi su čvorovi isprogramirani da sarađuju jedni sa drugima što se realizuje razmenjivanjem poruke ili sa nekom vrstom deljene memorije.

Važno je primetiti da se kao posledica međusobnog delovanja raznih nezavisnih čvorova javlja odsustvo zajedničke vremenske reference. Naime, ne može se pretpostaviti da postoji nešto kao što je globalni sat. Ovaj nedostatak zajedničkog vremena dovodi do velikih problema što se tiče sinhronizacije i koordinacije između čvorova. Distribuirani sistem bi trebalo da se prikazuje korisnicima kao jedinstven i koherentan sistem. Neki kažu da krajnji korisnici uopšte ne bi trebalo da primećuju da rade sa procesima i podacima koji su rasprostranjeni po čitavoj distribuiranoj mreži. Ovo se uglavnom smatra preterivanjem, i neki stabilni konsenzus je na tome da bi sistem trebao da se ponaša onako kako korisnici očekuju da se ponaša. Kolekcija čvorova bi trebala da radi na isti način kakva god bila interakcija između korisnika i sistema. Distribuirani sistemi su uglavnom organizovani tako da imaju poseban softverski sloj koji je logički postavljen iznad operativnog sistema pojedinačnog računara koji je deo sistema. Ovaj sloj se popularno naziva middleware. Na neki način, middleware je distribuiranom sistemu isto što i operativni sistem jednom računaru. To je softverski sloj koji deluje kao upravljač resursima koji nudi svoje aplikacije kako bi se ti resursi efikasno podelili i iskoristili u mreži. Pored upravljanja resursima, može da nudi i druge servise koji su tipični za operativne sisteme. Ovo može uključivati servise za bezbednost, servise za kontrolu naloga, servise za oporavak od otkaza, servise za međuaplikacijsku komunikaciju itd. Razlika između ovih servisa i servisa operativnog sistema jeste to što se ovi servisi nude u mrežnom okruženju.

3. TEHNOLOGIJA DISTRIBUIRANE GLAVNE KNJIGE I BLOKČEJN

Distribuirana baza podataka je vrsta baze podataka kod koje se podaci čuvaju na više čvorova, tj. distribuirani su na više lokacija. Glavna knjiga (eng. Ledger) je knjiga koja služi za beleženje i sumiranje ekonomskih transakcija, sa debitima i kreditima u posebnim kolonama, takođe sa početnim i krajnjim stanjem dva računa. Dakle, distribuirana glavna knjiga, ili tehnologija distribuirane glavne knjige (eng. Distributed Ledger Technology – DLT) je vrsta distribuirane baze podataka u kojoj se pretpostavlja prisustvo malicioznih korisnika tj. čvorova.

Postoje tri glavne karakteristike tehnologija distribuiranih glavnih knjiga:

Jezik transakcija: Mora da postoji određeni format po kome će čvorovi da traže promenu stanja glavne knjige, tj. formalan način na koji će inicirati transakcije

Protokol: Moraju postojati pravila po kojima će se među učesnicima u distribuiranom sistemu postići konsenzus o tome koje će se transakcije prihvatiti i u kom redosledu će se upisati u glavnu knjigu

Model podataka: Trenutno stanje glavne knjige mora biti definisano određenim modelom podataka koji je odabran u vreme tehničke izvedbe sistema

Koncept tehnologija distribuirane glavne knjige postoji već dugo. Još 1982. godine su Lamport, Šostak i Pis opisali "problem Vizantijskih generala" i postavili pitanje kako će se računarski sistem nositi sa suprotstavljenim informacijama u neprijateljskom okruženju. 1999. godine istraživanja dovode do prvog algoritma koji na praktičan način rešava prethodno pomenuti problem. Ova tehnologija je osnova za novu generaciju aplikacija na bazi transakcija koje uspostavljaju transparentnost, odgovornost i poverenje u računarskim sistemima, a pri tom racionalizuju poslovne procese i pravna ograničenja kroz automatizaciju.

Blokčejn predstavlja jednu vrstu distribuirane baze podataka u kojoj se skladište digitalne transakcije. Ova tehnologija predstavlja podskup tehnologija distribuirane glavne knjige. Sa pojavom Bitcoin-a je blokčejn doveo do revolucije u distribuiranim sistemima i načinu na koju ljudi gledaju finansijsko poslovanje, lance isporuke, privatnost itd.

Postoje dve osnovne ideje koje razlikuju blokčejn od ostalih tehnologija distribuirane glavne knjige.

Prva je ideja da blokčejn bude projektovan tako da bi se postizao pouzdan i konzistentan dogovor između nezavisnih čvorova o zapisu događaja. Ovo se postiže mehanizmom konsenzusa koji omogućava da za svakog učesnika u mreži bude isti pogled na deljenju bazu podataka [4]. Dakle, učesnici u blokčejn mreži postižu konsenzus o promenama u deljenoj bazi podataka bez potrebe da se proverava integritet bilo kog čvora u mreži.

Druga ideja je specifična struktura podataka koja omogućava ovoj tehnologiji da reši problem dvostruke potrošnje, tj. onemogućava korisnicima da isti digitalni fajl kopiraju i prenose više puta. Blokčejn se iz ovih razloga može koristiti za razmenu valute, vrednosti ili drugih podataka centralnim bez potrebe za autoritetom.

Osnovna podela blokčejnova je na javne i privatne. Javnim sistemima može da pristupi bilo ko, dok privatnim mogu da pristupe samo autorizovani korisnici. Takođe je bitno uvideti da mogu postojati razne kontrole pristupa. Tako na nekim blokčejn sistemima može bilo ko i da čita i da piše na glavnu knjigu (Bitcoin, Ethereum), na nekima bilo ko može da čita, a samo autorizovani učesnici mogu da pišu (Sovrin), a na nekima postoji kontrola pristupa i za operacije čitanja i pisanja. Blokčejn koristi razne kriptografske tehnike za različite svrhe.

Prvenstveno, glavna knjiga u blokčejn sistema predstavlja lanac kriptografski povezanih blokova. Blok je skup određenog broja transakcija koje se istovremeno dodaju u lanac. U zaglavlju svakog bloka se nalazi referenca na prethodni blok u vidu kriptografskog heša tj. rezultata heš funkcije. Heš funkcije su jednosmerne (računanje inverzne funkcije je matematički zahtevno) matematičke funkcije koje na osnovu nekog skupa podataka daju heš, tj. vrednost fiksne dužine koja je dobijena na osnovu tog skupa podataka. Svrha ovoga je da se onemogući izmena blokova nakon dodavanja blokova u lanac. Ukoliko neki

učesnik pokuša da izmeni neki blok, ostali učesnici će to lako primetiti jer se heš tog izmenjenog bloka neće poklapati sa hešom koji je sadržan u zaglavlju novog bloka. Osim kriptografskog heša se u zaglavlju svakog bloka nalazi i koren Merkleovog stabla. Merkleovo stablo ili heš stablo je stuktura podataka koje predstavlja stablo kome je svaki terminalni čvor označen hešom bloka podataka, a svaki neterminalni čvor je označen hešom oznaka njegovih potomaka. Ovakva struktura omogućava pouzdanu i efikasnu pretragu i verifikaciju velikih struktura podataka.

Svako zaglavlje takođe sadrži vremenski otisak i nonce vrednost. Vremenski otisak predstavlja trenutak u vremenu kada je blok dodat u blokčejn. Nonce vrednost je slučajan broj koji se koristi samo jedanput, i ima ulogu u procesu validacije blokova.

U blokčejnu se takođe koristi infrastruktura javnih ključeva. Na blokčejnu postoje javne adrese kojima su pridružene određeni resursi (u primeru Bitcoin-a bi to bila neka količina Bitcoin-ova). Ideja je da svaki korisnik ima svoj privatan ključ, na osnovu kojeg on može da generiše javne ključeve ili adrese. Uz pomoć ovog privatnog ključa, korisnik može da dokaže posedovanje neke javne adrese, tj. adrese koja je izvedena iz tog ključa. Dakle, svi resursi se nalaze u glavnoj knjizi, korisnici putem svojih ključeva pristupaju glavnoj knjizi i dobijaju mogućnost da manipulišu resursima koji su pridruženi njihovim javnim adresama. Postizanje dogovora oko novog stanja glavne knjige je jedan od najvećih problema distribuiranih sistema. Konsenzus je proces postizanja dogovora između svih čvorova u mreži o ispravnom stanju podataka i cilj je da svi čvorovi dele iste podatke. Ovo se postiže konsenzus algoritmima. Ovi algoritmi osiguravaju da su upisani podaci u glavnu knjigu isti za sve čvorove u mreži i tako sprečavaju maliciozne učesnike da manipulišu podacima.

Osvrnućemo se na neke najpopularnije konsenzus algoritme.

Dokaz posla (eng. Proof of work) je konsenzus algoritma baziran na lutriji u kome se podrazumeva rešavanje matematičke zagonetke koja je zahtevna za izračunavanje kako bi se kreirao novi blok u sistemu. Najpoznatiji sistem baziran na dokazu posla je heškeš (Adam Bek 1997.) i služi za limitiranje spama i sprečavanje denial-of-service napada. Jedini način da se pronađe rešenje matematičke zagonetke su brute-force algoritmi koji manje više pogađaju tačno rešenje. Provera rešenja je jednostavna i ne zahteva kompleksna izračunavanja. Proces izračunavanja se naziva rudarenjem, a čvorovi u mreži koji pokušavaju da dođu do rešenja se nazivaju rudari. Rudari imaju ekonomski podsticaj jer pronalazak heša novog bloka dovodi do stvaranje novih resursa koji pripadaju rudaru (u nekim blokčejn sistema) i dobijaju proviziju od transakcija koje su validirane u tom bloku.

Dokaz uloga (eng. Proof of stake) je konsenzus algoritam koji je takođe baziran na lutriji i predstavlja generalizaciju dokaza posla. U ovom slučaju su čvorovi poznati kao validatori i oni validiraju transakcije kako bi zaradili proviziju od istih (ovde nema stvaranja novog resursa). Čvorovi se slučajno biraju za validatore blokova, a ova verovatnoća zavisi od veličine uloga nekog čvora. Ukoliko čvor A poseduje dva resursa, a čvor B jedan resurs, čvor A ima veću šansu da bude postavljen kao validator. Ključni aspekt je slučajnost, kako bi se izbegla situacija gde najbogatiji čvorovi stalno dobijaju ulogu validatora.

4. ETHEREUM I DECENTRALIZOVANE APLIKACIJE

Ethereum je jedna od najpoznatijih platformi baziranih na blokčejn tehnologiji. Ova platforma je otvorenog koda, javna i omogućuje razvoj i rad raznih projekata koji žele da iskoriste prednosti decentralizacije. Za razliku od Bitcoin-a, što je platforma koja isključivo služi za razmenu digitalne valute, na Ethereum-u mogu da se implementiraju najrazličitije funkcionalnosti.

Rane blokčejn aplikacije kao što su Bitcoin su jedino dozvoljavale korisnicima ograničen skup predefinisanih operacija. Za razliku od ovakvih projekata, Ethereum nudi korisnicima da prave svoje operacije. Ovo nas dovodi do koncepta Ethereum Virtualne Mašine (EVM). EVM je Ethereumovo okruženje za izvršavanje, i služi za izvršavanje pametnih ugovora. Pošto svaki Ethereum čvor izvršava Ethereum Virtualnu Mašinu, aplikacije izgrađene na ovoj bazi dobijaju pogodnosti decentralizacije bez potrebe da koriste neki posebno dizajniran blokčejn.

Slično kao i u Bitcoin-u, Ethereum ima svoj koncept rudarenja i svoju kriptovalutu Etar. Ethereum trenutno koristi dokaz posla kao konsenzus algoritam (kao i Bitcoin), ali je u planu prelazak na dokaz uloga. Do tada Etar stvara platforma i isplaćuje ga rudarima koji izračunaju heš novog bloka transakcija.

Pametni ugovori su programski kodovi koji na visokom nivou opisuju interakcije između korisnika i distribuirane glavne knjige. Ovi kodovi mogu da se kompajliraju u EVM bajtkod i da se izvršavaju na Ethereum Virtualnoj Mašini. Kao što Bitcoin ima svoj BitcoinScript, tako i Ethereum ima svoj jezik za pisanje pametnih ugovora, Solidity.

Razlika između ova dva je to što je Solidity Tjuring kompletan i nudi mnogo više mogućnosti, dok BitcoinScript podržava samo finansijske transakcije.

Pametni ugovori se postavljaju na Ethereum blokčejn gde dobijaju svoje javne adrese. Ovo zapravo znači da će kod ovih pametnih ugovora izvršavati mašine-čvorovi od rudara. Korisnici mogu pristupati ovom ugovoru putem te javne adrese i vršiti interakciju sa njim. Pojedinosti interakcije zavise od funkcionalnosti samog pametnog ugovora.

Na primer, Alisa i Bob žele da se opklade u 100 etra u ishod fudbalske utakmice. Oni se dogovore oko poverljive veb stranice za rezultate, i svako pošalje po 100 etra na javnu adresu pametnog ugovora. Nakon što se utakmica završi, pametni ugovor će proveriti rezultat utakmice na osnovu poverljive veb stranice. U zavisnosti od rezultata, pametni ugovor će automatski isplatiti 200 etra pobedniku opklade. Originalna zamisao Ethereuma je da on deluje kao „svetski računar“. Čvorovi mreže bi svi imali EVM kao midlver sloj, koji bi davao utisak da su svi računari deo jednog koherentnog sistema.

Tradicionalna veb aplikacija bi imala tri dela: frontend, bekend i bazu podataka. Kod decentralizovanih aplikacija postoji sličan pristup.

Frontend deo, koji predstavlja prednji deo aplikacije sa kojim korisnik intereaguje, može biti hostovan na nekom

centralnom serveru, a može biti i hostovan na nekim decentralizovanim platformama za skladištenje kao što su Svarm. Ovako bi se uspostavila potpuna decentralizacija.
Bekend deo, tj. serverski deo koji predstavlja logiku i spoj između podataka i frontenda, bi se implementirao kao pametni ugovor na nekoj blokčejn mreži.
Ovako se taj bekend ne bi izvršavao na nekom centralizovanom serveru, već bi ga izvršavali svi čvorovi mreže na potpuno distribuiran način.
Baza podataka bi u ovom slučaju bila sama distribuirana glavna knjiga blokčejn sistema. Zamisao osnivača Ethereum-a, Vitalik Buterina, je od početka bila da Ethereum ima ulogu svetskog računara gde čvorovi izvršavaju decentralizovane aplikacije. Ideja je da decentralizacija nudi brojne prednosti u finansijskoj, proizvodnoj, zdravstvenoj i informativnoj industriji.

5. KORIŠĆENE TEHNOLOGIJE I ALATI

Kao podrška razvoju okruženja za bavljenje blokčejn sistemom je iskorišćen Truffle Suite [5]. Truffle Suite obuhvata nekoliko alata za olakšan rad sa sistemima na bazi Ethereuma.
Ganache je personalni blokčejn namenjen za brzi razvoj Ethereum i Corda distribuiranih aplikacija. Može se koristiti tokom celog ciklusa razvoja, omogućavajući razvoj, pokretanje i testiranje decentralizovanih aplikacija u bezbednom i determinističkom okruženju. Ganache ima svoju desktop aplikaciju koja je iskorišćena u sklopu ovog rada da se simulira ponašanje Ethereum blokčejna. Razvijati decentralizovane aplikacije na primarnom Ethereum blokčejnu nije praktično iz više razloga.
Prvenstveno, transakcije na Ethereum-u koštaju. Zvog ovoga Ethereum ima svoju test mrežu, gde Ether nema vrednost, ali ni ona nije praktična za korišćenje. Ovo je zbog toga što je računarski zahtevno biti punopravni čvor na Ethereum mreži. Zato je korišćen Ganache, koji praktično simulira ponašanje pravog Ethereum blokčejn sistema.
Truffle je okruženje za rad sa blokčejnovima koji rade u sklopu Ethereum virtuelne mašine bazirano na NodeJS. NodeJS je okruženje otvorenog koda za izvršavanje JavaScript koda van pretraživača. Truffle olakšava razvoj i testiranje Ethereum aplikacija.
Neke stvari koje Truffle nudi su: ugrađeni mehanizmi za kompajliranje pametnih ugovora i linkovanje binarnih datoteka, mogućnost pisanja automatskih testova za brzi razvoj, pisanje skripti za podešavanje ugovora i migracija, upravljanje sa stanovišta mreža, upravljanje paketima i bibliotekama, interaktivna konzola.
Truffle se skida i instalira putem NPM-a (Node Packet Manager). Nakon podešavanja samog okvira, možemo započeti korišćenje Truffle-a.
Web3.js je JavaScript biblioteka koja omogućava rad sa Ethereum čvorovima [7]. Svaki Ethereum čvor ima RPC API i metode iz ovog API-ja je moguće pozivati putem metoda iz Web3.js biblioteke.
MetaMask je novčanik za kriptovalute i sprega pretraživača i blokčejna. Dolazi u formi plagina za pretraživač (Mozilla, Chrome..). Omogućava pametno upravljanje Ethereum adresama i ključevima iz pretraživača. Pomoću MetaMask-a, frontend aplikacija može da prosleđuje relevantne korisničke podatke ka blokčejnu.

6. REŠENJE

Predstavljeno rešenje sastoji se iz tri dela: distribuirane baze podataka, bekend aplikacije, i frontend aplikacije. Ulogu distribuirane baze podataka vrši sama Ethereum blokčejn platforma. Ovoj bazi pristupa bekend aplikacija, u vidu pametnog ugovora koji reguliše transakcije iz i ka blokčejnu. Frontend aplikacija, napisana u JavaScript-u, komunicira sa bekend aplikacijom u cilju ispravnog prikazivanja relevantnih podataka korisniku.
Aplikacija podržava sledeće mogućnosti: pravljenje novih decentralizovanih izbora, dodavanje novih kandidata na izbore, dodavanje novih učesnika (glasača) na izbore, glasanje na izborima, prikazivanje rezultata izbora. Trenutna verzija aplikacije omogućava isključivo deljenje informacija o izborima relevantnim korisnicima proverama javnih adresa korisnika. Ovo ograničenje se može prevazići u budućnosti dodavanjem servisa za članstvo (eng. membership service). Ovako bi korisnici mogli da učestvuju na više izbora istovremeno, a i mogli bi da prate rezultate drugih izbora na kojima ne učestvuju, ukoliko imaju dozvolu za to.

7. ZAKLJUČAK

Prednost ove decentralizovane aplikacije se ogleda u tome da pruža mnogo veću zaštitu nego tradicionalne centralizovane aplikacije. Ovakav pristup omogućava bezbedno i pouzdano deljenje informacija među korisnicima. Maliciozni učesnici koji imaju za cilj da kompromituju neke izbore imaju značajno redukovane mogućnosti u ovakvom decentralizovanom sistemu. S druge strane, ovakav sistem je mnogo zahtevniji za implementaciju, kao i za održavanje, od tradicionalnih centralizovanih sistema.

LITERATURA

[1] Dr Dušan Gajić, *Materijali sa predmeta Paralelni i distribuirani algoritmi i strukture podataka*, dostupno na: http://www.acs.uns.ac.rs/sr/node/237/4468699, poslednji pristup jul 2020.
[2] Maarten van Steen, Andrew S. Tanenbaum., *Distributed Systems (third edition)*, Maarten van Steen 2018.
[3] Don Tapscott, *Blockchain Revolution*, Penguin Random House LLC 2016.
[4] *Blockchain Consesnus*, dostupno na: *https://devopedia.org/blockchain-consensus* , poslednji pristup mart 2020.
[5] *Truffle* , dostupno na: www.trufflesuite.com, poslednji pristup jul 2020.
[6] *Ethereum*, dostupno na: https://ethereum.org/en/, poslednji pristup jul 2020.
[7] *Web3js*, dostupno na: https://web3js.readthedocs.io/en/v1.2.11/, poslednji pristup jul 2020.
[8] *Metamask*, dostupno na: https://metamask.io/, poslednji pristup jul 2020.

Kratka biografija:

Nikola Malenčić rođen je u Novom Sadu, Republika Srbija, 11. novembra 1996. godine. Osnovne akademske studije je upisao na Fakultetu tehničkih nauka Univerziteta u Novom Sadu 2015. godine. Diplomirao je 2019. godine.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.38
DOI: https://doi.org/10.24867/10BE10Stojkovic

ELEKTRONSKA KONTROLNA JEDINICA (ECU) ELEKTRIČNOG MOTORA AUTOMOBILA

ELECTRONIC CONTROL UNIT (ECU) OF THE CAR ELECTRIC MOTOR

Aleksa Stojković, Vlado Porobić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu predstavljeni su razvoj električnog motora automobila i trofaznog invertora na Typhoon HIL emulatoru i faze testiranja upravljačkog algoritma indirektno vektorsko upravljanje implement-iranog na elektronskoj kontrolnoj jedinici. Primenjen je koncept Finite state machine radi prevođenja električnog motora iz jednog radnog režima u drugi.*

Ključne reči: *HIL emulator, DSP, električni motor*

Abstract – *In this paper the development of the car electric motor and three-phase inverter with Typhoon HIL emulator are represented with test phases of the IFOC algorithm which is implemented on ECU. Also, the Finite state machine modeling method is used to change the operational points of the electric motor.*

Keywords: *HIL emulator, DSP, electric motor*

1. UVOD

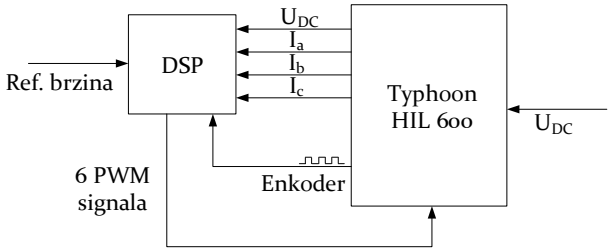
Zadatak elektronske kontrolne jedinice električnog motora automobila jeste da obezbedi njegov optimalan rad. Potrebno je da u odgovarajućim vremenskim trenucima dobija analogne i digitalne merne signale neophodne za izvršavanje upravljačkog algoritma, da te signale na odgovarajući način procesira i da na svom izlazu generiše upravljačke signale ka energetskom pretvaraču zahvaljujući kojima će sve veličine od interesa imati željene vrednosti.

Ulogu elekričnog motora i energetskog pretvarača u ovom radu obavlja HIL emulator, dok ulogu ECU obavlja TI Docking stanica zajedno sa TI DSP kontrolnom karticom.

2. HIL MODEL

Typhoon HIL uređaj u ovom radu ima zadatak simulacije rada energetskog dela sistema, odnosno trofaznog kaveznog asinhronog motora i trofaznog invertora. TI Docking stanica je ploča uz pomoć koje se obezbeđuje razmena signala između HIL-a i DSP-a. Na njoj se nalazi i odgovarajuće mesto za postavljanje TI DSP kontrolne kartice, korišćena je TMDSCNCD28335 kartica. Slika 1. prikazuje koji se signali razmenjuju između HIL-a i DSP-a, na kojima se nalaze model energetskog dela i upravljački algoritam, respektivno. Takođe, može se videti za zadavanje kojih veličina je zadužen korisnik.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji je mentor bio dr Vlado Porobić, vanr. prof.

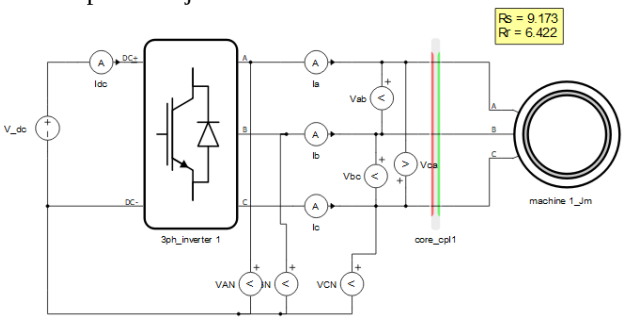


Slika 1. *Prikaz razmene signala između DSP-a i HIL-a*

Za pravljenje modela i posmatranje promena veličina od interesa upotrebljavan je softver Typhoon HIL Control Center V2020.2, najnovija verzija u trenutku izrade rada.

2.1. Schematic Editor

Schematic Editor prozor u okviru pomenutog softvera služi za pravljenje energetskog dela modela. Izgled modela prikazan je slikom 2.



Slika 2. *Model trofaznog invertora i trofaznog kaveznog asinhronog motora u Schematic Editor-u*

U blok trofaznog kaveznog asinhronog motora su uneti odgovarajući električni i mehanički parametri [1]. U pitanju je motor nominalne snage 1,1 kW. Za merenje položaja rotora odabran je inkrementalni enkoder rezolucije 3600 impulsa/obrtaju.

Za trofazni invertor izabrano je upravljanje prekidačima preko digitalnih ulaza. U ovom bloku je pre svega bilo potrebno podesiti na koji digitalni ulaz HIL-a se dovode signali sa ePWM modula DSP-a. Njihov izbor nije proizvoljan i izvršen je spram [2]. Za logiku upravljanja određena je active high logika, koja podrazumeva da će prekidači biti u provodnom stanju u slučaju logičke jedinice, a otvoreni u slučaju logičke nule.

2.2. HIL SCADA

Nakon kompajliranja modela na HIL emulator otvara se prozor HIL SCADA u kojem je omogućeno zadavanje željenih ulaznih veličina, poput zadavanja napona DC kola ili momenta opterećenja. Takođe, u ovom prozoru se

određuje na koje analogne izlaze HIL-a će se dovoditi merene struje i napon DC kola, kao i na koje digitalne izlaze će se dovoditi signali A, B i INDEX sa inkrementalnog enkodera. Izbor analognih i digitalnih izlaza je takođe izvršen spram [2].

3. DSP MODULI

Za koncept upravljanja električnim motorom automobila izabrano je indirektno vektorsko upravljanje (IFOC). Koncept indirektnog vektorskog upravljanja i način njegovog razvoja u MATLAB-ovom alatu Simulink objašnjen je u [3], čime je programiranje DSP-a moguće realizovati automatskom generacijom koda (Automatic Code Generation) putem razvoja softvera baziranog na modelovanju (Model Based Development).

Moduli DSP-a koje je neophodno koristiti prilikom razvoja indirektnog vektorskog upravljanja su: ADC modul, ePWM (Enhanced Pulse Width Modulator) moduli i eQEP (Enhanced Quadrature Encoder Pulse) modul. Njihova uloga jeste prihvatanje i procesiranje signala koje ka DSP-u šalje HIL emulator, kao i generisanje željenih 6 PWM signala kojima se upravlja prekidačima trofaznog invertora prikazanog na slici 2. Njihova odgovarajuća inicijalizacija izvršena je preko blokova u Simulink modelu i to je urađeno uporedo čitanjem kako uputstva korišćenog DSP-a [4], tako i uputstva korišćenog HIL emulatora [2].

4. FAZE TESTIRANJA UPRAVLJAČKOG ALGORITMA

Testiranje upravljačkog algoritma je izvršeno tako što su u svakoj od faza uključivani odgovarajući blokovi i podsistemi Simulink modela [3] i proveravani dobijeni odzivi veličina od interesa. Za praćenje promena varijabli definisanih u C kodu, dobijenom nakon automatske generacije koda iz Simulink modela, korišćen je softver PERUN PowerDesk (PPD). Jednom istestiran softverski modul na odgovarajući način se podrazumeva da radi i njega u narednim fazama nije potrebno dalje testirati.

Razvoj i testiranje upravljačkog algoritma se može sastojati iz faza predloženih u [5], gde ih je predloženo pet. Međutim, u ovom radu su zbog načina implementacije upravljačkog algoritma, kao i razvoja modela na HIL emulatoru, primenjene četiri faze. Pored toga, uvedena je i faza 0 koja podrazumeva merenja analognih veličina, tri linijske struje i napona DC kola.

4.1. Faza 0

Faza 0 ima zadatak da vrednosti linjskih struja i napona DC kola očitanih u PPD softveru odgovaraju očitanim vrednostima u HIL SCADA prozoru, odnosno njihovim stvarnim vrednostima. Za ispitivanje faze 0 dovoljno je upotrebljavati samo blok ADC modula u Simulink-u i logiku kojom se vrši korekcija izmerene veličine i njeno pretvaranje u sistem relativnih vrednosti.

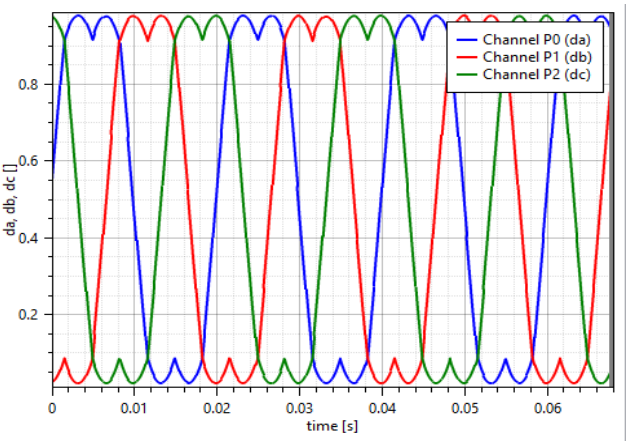
4.2. Faza 1

Zadatak prve faze razvoja i testiranja upravljačkog algoritma ogleda se pre svega u testiranju ispravnosti rada SVPWM generatora čija je uloga da izračuna vremena vođenja prekidača trofaznog invertora, odnosno faktore ispune, gde se kao rezultat dobija generisanje željenog vektora napona na motoru [6]. Takođe, u okviru ove faze testiran je i rad IPARK i ePWM modula. U fazi 1, kao i fazi 2, koriste se i moduli RAMPCTRL i RAMPGEN koji služe da se za zadatu referencu brzine omogući menjanje po rampi ugla dq koordinatnog sistema.

U ovoj fazi se kao reference zadaju tri varijable: napon po d osi (Vdtesting), napon po q osi (Vqtesting) i referentna brzina (SpeedRef), odnosno frekvencija (Freq). Takođe, u ovako koncipiranom sistemu zadaje se i napon DC kola iz opcije Sources u HIL SCADA prozoru. U ovom radu je ta vrednost konstantna i iznosi $U_{dc} = 563\ V$ i ova vrednost se kao i sve ostale merene veličine pretvara u relativne vrednosti.

Ispravnost rada SVGEN_AB modula potvrđena je slikom 3., sa koje se može videti da su faktori ispune međusobno fazno pomereni za 120° i da je perioda svakog signala 20ms. Prikazani odzivi faktora ispune za sve tri grane invertora dobijeni su za sledeći set varijabli:

- $V_{dtesting} = 0{,}41396\ r.j.$
- $V_{qtesting} = 0\ r.j.$
- $SpeedRef = 1500\ o/min\ (Freq = 50\ Hz)$



Slika 3. *Faktori ispune tri grane trofaznog invertora*

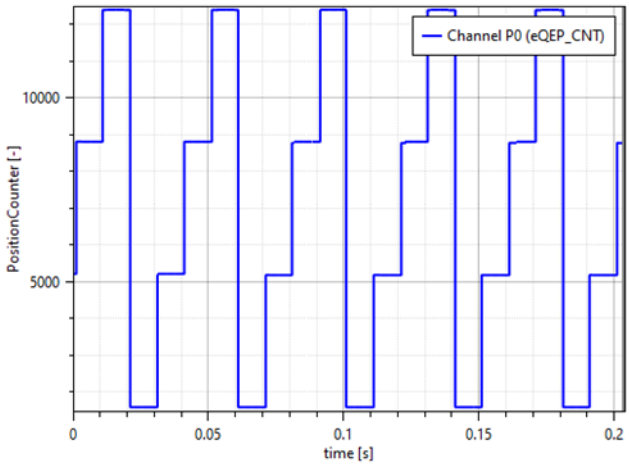
Pravilan rad IPARK modula potvrđen je pravilnim radom SVGEN_AB modula, s obzirom da su ulazne veličine za ovaj modul izlazne veličine modula inverzne Park transformacije. Dobra podešenost blokova ePWM modula potvrđena je dobijenim faznim naponima sinusnog talasnog oblika odgovarajuće amplitude i međusobne fazne pomerenosti.

4.3. Faza 2

Faza 2 testira rad CLARKE; PARK i eQEP modula. Referentne veličine koje se zadaju u ovoj fazi su iste kao i u prethodnoj fazi.

Podsistem u kojem je implementirana Clarke transformacija je potvrđen odgovarajućim odzivima i_α i i_β., dok je zaključak o pravilnom radu podsistema Park transformacije izveden na osnovu odgovarajućih odziva svih ulaznih veličina u ovaj podsistem.

Ispravna podešenost eQEP modula prikazana je slikom 4. To je situacija kada je trofazni kavezni asinhroni motor u praznom hodu. Smer obrtanja je u obrnutom smeru od kazaljke na časovniku, pozitivno definisan smer, što se vidi na osnovu inkrementovanja vrednosti brojačkog registra. Pravilan rad je potvrđen i za obrtanje u suprotnom smeru, kada se vrednost upisana u brojački registar dekrementuje.



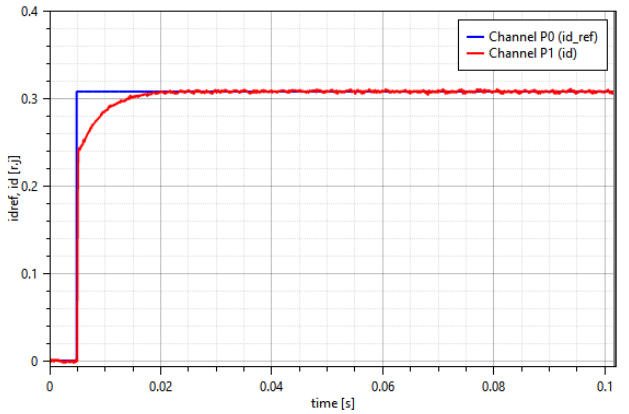
Slika 4. *Brojački registar eQEP periferije*

4.4. Faza 3

Ova faza obuhvata testiranje strujnih regulatora po d i q osi, kao i podsistema u kojem je realizovan algoritam za indirektnu procenu položaja vektora fluksa rotora (CUR_MOD). Reference koje se zadaju u fazi 3 jesu $IdRef$ i $IqRef$.

U cilju je pre svega podesiti parametre PI regulatora koji su identični za oba strujna regulatora. Oni se određuju po Dahlin-ovom postupku s obzirom da se želi aperiodičan odziv struje. Dahlinov postupak za određivanje parametara regulatora objašnjen je u [7], te su oni spram toga i podešeni.

Slika 5. prikazuje odziv struje i_d na zadatu odskočnu referencu. Može se videti da struja ima aperiodičan odziv i željenu dinamiku dostizanja zadate referentne vrednosti.



Slika 5. *Odziv struje i_q na zadatu odskočnu referencu*

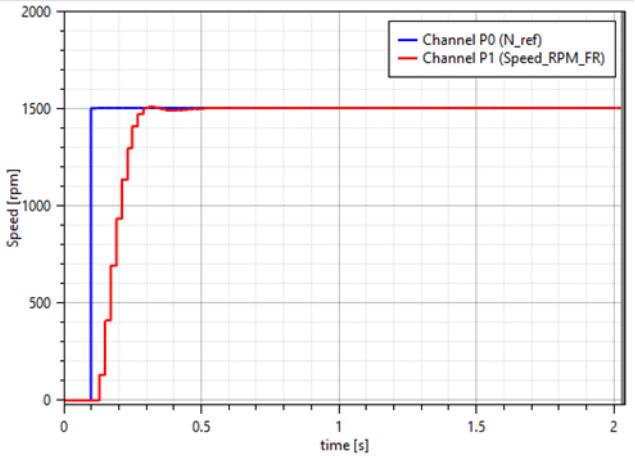
Potvrđen je i željeni odziv struje i_q, čime je zaključeno da su parametri strujnih regulatora dobro podešeni. Ispravnost CUR_MOD potvrđena je očekivanom promenom ugla dq koordinatnog sistema.

4.5. Faza 4

Poslednja faza razvoja upravljačkog algoritma ogleda se u zatvaranju povratne sprege po brzini. U ovoj fazi se uvodi brzinski regulator (PI ω) čiji je zadatak da otkloni grešku između zadate i stvarne brzine obrtanja motora. Kao referentne veličine u ovoj fazi se zadaju $IdRef$ i $SpeedRef$.

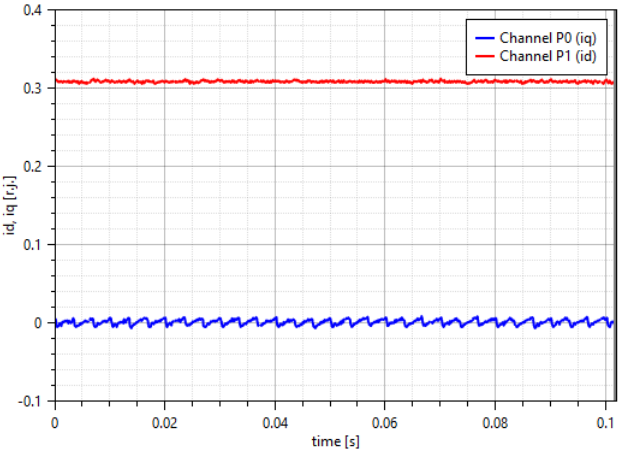
Kao i u slučaju odziva struja i kod odziva brzine želi se aperiodičan odziv. Parametri brzinskog regulatora su podešeni na način kojim se dolazi do optimalnih vrednosti parametara objašnjen u [3]. Međutim, nizom eksperimenata je utvrđeno da se najbolji odziv brzine dobija za korigovane vrednosti parametara koje iznose: $K_{pw} = 9{,}023$ i $K_{iw} = 1{,}012$, te su ove vrednosti parametara regulatora i usvojene.

Slika 6. prikazuje odziv brzine prilikom zadavanja $SpeedRef = 1500\ o/min$, ali pre toga je zadata struja $IdRef$ sa istom vrednošću kao u prethodnoj fazi, čime je obezbeđeno magnetno pobuđivanje mašine.



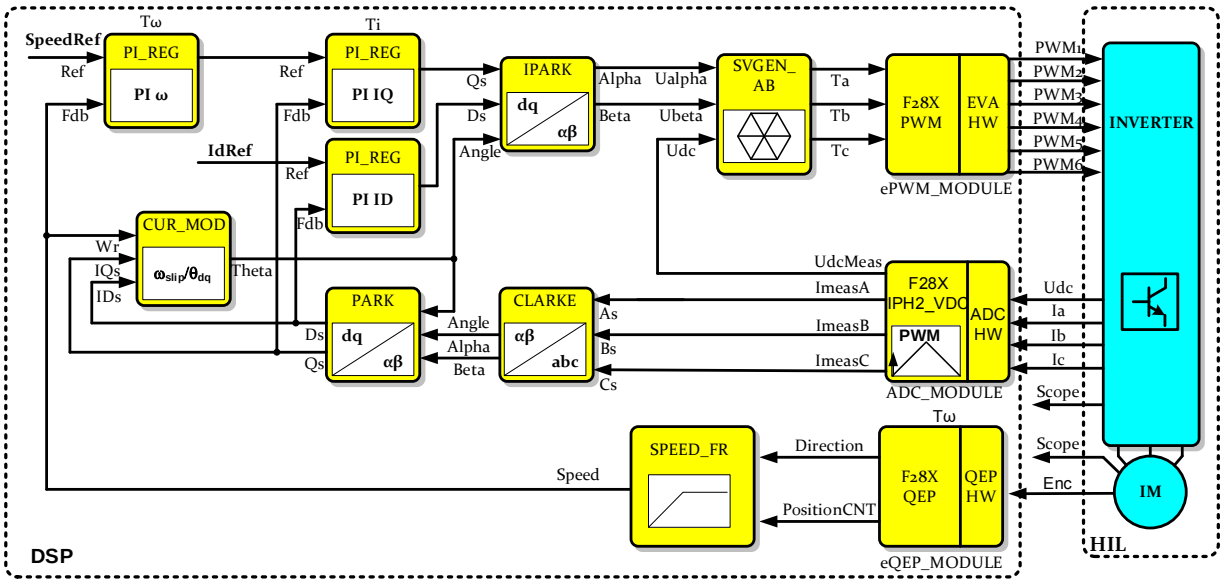
Slika 6. *Odziv brzine na zadatu odskočnu referencu*

Da je postignut željeni cilj da se AM ponaša kao linerani konvertor momenta, gde se u stacionarnom stanju očekuje vremenska nepromenljivost komponenti vektora struje statora i_d i i_q [6], prikazuje slika 7. S obzirom da je u ovom slučaju električni motor neopterećen, očekivana vrednost q komponente struje statora je $i_q = 0$, dok d komponenta struje statora ima istu vrednost kao na slici 5.



Slika 7. *Odzivi struja i_d i i_q u stacionarnom stanju*

Završetkom poslednje faze testiranja dobija se upravljački algoritam indirektno vektorsko upravljanje implementiran na DSP. Prikaz svih softverskih modula korišćenih za adekvatno upravljanje nalazi se na slici 8.



Slika 8. *Prikaz softverskih modula u sklopu upravljačkog algoritma (DSP) i energetskog dela (HIL)*

5. FINITE STATE MACHINE (MAŠINA SA KONAČNIM BROJEM STANJA)

Finite state machine predstavlja metod modelovanja ponašanja, odnosno stanja u kojima se može naći neki fizički sistem. Na osnovu definisanih radnih režima u kojima se može naći električni motor definisane su vrednosti referentnih ulaznih veličina sa slike 8. Prelazak iz jednog stanja u drugo definisano je na osnovu komandi koje zadaje korisnik. Jedan od postignutih radnih režima jeste rad trofaznog kaveznog asinhronog motora u praznom hodu prikazan slikom 6. Moguća stanja u kojima se može naći električni motor i uslovi za prelazak iz jednog stanja u drugo definisani su putem Stateflow dijagrama u Simulink-u.

6. ZAKLJUČAK

Cilj ovog rada jeste bio da pokaže postupak razvoja modela objekta upravljanja i energetskog pretvarača, kao i simulaciju njihovog rada pomoću Typhoon HIL uređaja. Napravljen je osvrt na podešavanje svih DSP modula neophodnih za realizaciju indirektnog vektorskog upravljanja. Prikazane su faze testiranja upravljačkog algoritma i priloženi su odzivi od interesa, uz komentarisanje najznačajnijih karakteristika svakog. Na kraju je predočena metoda pomoću koje se vrši prevođenje mašine iz jednog radnog stanja u drugo.

7. LITERATURA

[1] Evgenije Adžić, "Regulisani pogon asinhronog motora sa minimalnim brojem senzora", doktorska disertacija, Fakultet tehničkih nauka, Novi Sad, 2013.

[2] Typhoon HIL Control Center V2020.2

[3] Nikola Diklić, Evgenije Adžić, Marko Vekić, "Realizacija vektorskog upravljanja asinhronog motora, upotrebom alata za automatsko generisanje koda.", master rad, Fakultet tehničkih nauka, Novi Sad, 2019.

[4] https://www.ti.com/lit/ug/sprui07/sprui07.pdf?-ts=1597851689910&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTMS320F28335%253Fqgpn%253Dtms320f28335 (pristupljeno u julu 2020.)

[5] Bilal Akin, Manish Bhardwaj, "Sensored Field Oriented Control of 3-Phase Induction Motors", Application Report, SPRABP8 – July 2013

[6] Darko P. Marčetić, "Mikroprocesorsko upravljanje energetskim pretvaračima", FTN Izdavaštvo, Novi Sad, 2014.

[7] Stevan Grabić, "Upravljanje energetskim pretvaračima", Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2016.

Kratka biografija

Aleksa Stojković rođen je 20.09.1996. godine u Vršcu. Diplomirao je u septembru 2019. godine, nakon čega se u oktobru iste godine zaposlio na Fakultetu tehničkih nauka kao saradnik u nastavi na Katedri za energetsku elektroniku i pretvarače.

Prof. dr Vlado Porobić, zaposlen je na Fakultetu tehničkih nauka u Novom Sadu, u zvanju vanrednog profesora. Predmet interesovanja su mu upravljanje pretvaračima energetske elektronike i automotive oblast.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE11Vujakovic

KOMUNIKACIONE TEHNOLOGIJE INTERNETA STVARI SA PRIMENOM U INDUSTRIJI

COMMUNICATION TECHNOLOGIES OF THE INTERNET OF THINGS FOR USAGE IN THE INDUSTRY

Miljana Vujaković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je prezentovana komparativna analiza komunikacionih CoAP i MQTT komunikacionih tehnologija za internet stvari. Dodatno, eksperimentalno su određivani faktori performansi CoAP, MQTT protokola. Predstavljena je ideja hijerarhijskog, multiprotokolskog rešenja za online nadgledanje bandera pametnih mreža oslanjanjem na analizirane protokole.*

Ključne reči: *internet stvari, komunikacioni protokoli, CoAP, MQTT, pametne mreže, nadgledanje bandera*

Abstract – *The paper presents the comparative analysis of CoAP and MQTT communication technologies essential for the Internet of Things. Additionaly, performance factors of CoAP, MQTT protocols are determined experimentally. The idea of hierarchical, multiprotocol solution for online monitoring of smart grid poles by relying on the analyzed protocos was presented.*

Keywords: *Internet of Things, Communication protocols, CoAP, MQTT, Smart Grid, Utility Pole Surveillance*

1. IOT PARADIGMA

Sintagma *Internet of Things* referiše na distribuirani sistem čiju srž čine mnogobrojni entiteti povezani preko komunikacionih mreža na Internet, a generički pojam „stvar“ (eng. *Thing*) ukazuje na širok doseg interneta stvari. Značaj se uočava kroz aplikacije orijentisane ka potrošaču (nosivi uređaji, pametne kuće..) i posebno kroz poslovne aplikacije gde se svrstava i primena IoT-a u industriji. Industrijski IoT je osnaživanje industrijskog inženjerstva senzorima i softverom sa ciljem blagovremenog uočavanja problema, i kreiranja efikasnih, ekonomičnih i optimalnih procesa. Značajna IIoT aplikacija je pametna mreža koja predstavlja evoluciju elektroenergetskih sistema (EES) integracijom savremenih informaciono-komunikacionih tehnologija u svaki od EES podsistema (proizvodnja, prenos, distribucija, potrošnja), sa ciljem obezbeđivanja kontrolisane, pouzdane, efikasne, stabilne i bezbedne isporuka električne energije. Za realizaciju povezivanja, automatizacije, kontrolisanja i praćenja brojnih infrastrukturnih elemenata pametne mreže koji poseduju različite mehanizme konektivnosti je potrebna dvosmerna komunikaciona infrastruktura, prisutna u IoT tehnologiji. U poglavlju 4. je obrađen jedan od primera upotreba IoT tehnologije u distributivnom podsistemu za nadzor bandera.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Vukmirović, vanr.prof.

2. KOMUNIKACIJE U IOT EKOSISTEMU

Protokoli TCP/IP modela su namenjeni upotrebi u mrežama računara opšte namene, gde su performanse računara zadovoljavajuće. Zbog svojstava IoT rešenja, pojavio se niz novih tehnologija za senzorske i telekomunikacione mreže. Od mnogo faktora koji utiču na performanse IoT rešenja, komunikacija na aplikativnom nivou je jedan od glavnih i u nastavku su predstavljeni MQTT i CoAP aplikativni protokoli.

2.1 CoAP

U julu 2013, Internet Engineering Task Force (IETF) je objavio CoAP - Constrained Application Protocol [1] (IETF RFC725260) protokol aplikativnog nivoa prilagođen za resursno ograničene uređaje male potrošnje u Internet mreži. Realizuje zahtev/odgovor komunikacioni obrazac. U CoAP IoT sistemu interakcija je bazirana na peer-to-peer klijent-server obrascu i svaki uređaj označen kao CoAP endpoint izlaže REST API. CoAP bazirana mreža se jednostavno povezuje sa HTTP mrežom koristeći proksi uređaje. Zahtevi i odgovori se razmenjuju asinhrono preko CoAP poruka. Može biti implemetiran preko UDP-a ili bilo kog drugog transportnog mehanizma baziranog na datagram-u. Da bi se omogućio kvalitet servisa na nivou aplikacije - postoje dva režima u kojima se poruke razmenjuju između CoAP čvorova (klijenata i servera): ne zahtevaju potvrdu prispeća (eng. Non-confirmable) i zahtevaju potvrdu prispeća (eng. Confirmable) od servera.

2.2 MQTT

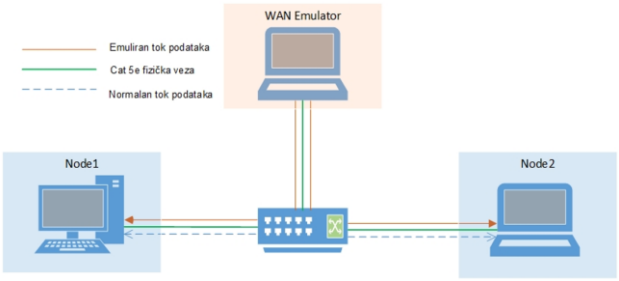
MQTT [2] je protokol za razmenu poruka aplikativnog sloja, razvijan od strane IBM-a krajem 1990-ih, no danas je otvoreni protokol. Razvijan je za komunikaciju preko nepozdanih satelitskih veza, sa udaljenim lokacijama, gde je propusni opseg mreže ograničen. Koristi se u okruženjima sa malim kapacitetom programske i radne memorije. Bazira se na objavi-pretplati komunikacionom obrascu i dobar je za sisteme koji se odlikuju nestabilnom mrežom sa prekidima veze, gubicima paketa, malim brzinama prenosa i velikim kašnjenjem. MQTT je ustaljen, stabilan protokol sa velikom podrškom na tržištu. MQTT se oslanja na TCP transportni protokol što omogućava pouzdanost u komunikaciji. MQTT klijenti se karakterišu ulogama objavljivača odnosno pretplatnika na poruke na određenim temama, dok je broker centralna komponenta koja upravlja razmenom poruka u objavi-pretplati komunikaciji. Na nivou pojedinačne specificira QoS koji se

odnosi na garanciju isporuke poruke. Razlikuju se QoS 0 koji garantuje isporuku poruka najviše jednom, QoS 1 za garanciju da će bar jedna poruka doći na odredište i QoS 2 koji garantuje da će poruka biti isporučena tačno jednom bez duplikata.

3. EKSPERIMENT

U eksperimentu je određivano zauzeće komunikacionog kanala u zavisnosti od veličine poruke na koju uticaj imaju: format sadržaja, odabrani protokol (MQTT, CoAP), upotreba mehanizama bezbednosti. Takođe je određivana i latencija kao funkcija veličine razmenjivane poruke, i kao posledica nepovoljnih uslova mreži.

U testnom scenariju (slika 1) se transportuju podaci od čvora Node1, koji ima ulogu agregatora na nižem nivou, do čvora Node2 koji je na višem nivou u hijerarhije u IoT rešenju. Razmena podataka između čvorova Node1 i Node2 se vrši protokolima MQTT, odnosno CoAP. Za kontrolu uslova u mreži korišćen je WANem alat emulator mreže širokog područja.



Slika 1. Postavka testnog okruženja

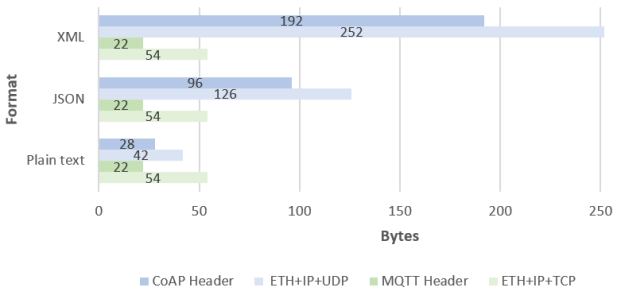
Sadržaj poruke predstavlja agregaciju merenja temperature i vlažnosti vazduha sa 3 emulirana uređaja sloja opažanja (kratke/male poruke) i sa 30 uređaja (dugačke/velike poruke). Latencija je određivana tokom slanja paketa po povratnoj putanji (RTT).

Za testiranje MQTT protokola je potrebna implementacija brokera i klijenta. Zbog lakoće podešavanja i korišćenja, odabran je **mosquito** broker koji je u testnom scenariju instaliran na čvoru Node1, dok je **M2Mqtt** biblioteka upotrebljena u .NET Core MQTT klijentskim aplikacijama koje se izvršavaju na čvorovima Node1 i Node2. Jedina funkcionalna CoAP .NET Core/.NET Framework implementacija u trenutku izvršavanja eksperimenta je **Com.AugustCellar/CoAP** implementacija. Opcija koja omogućava transferovanje velikih reprezentacija se zove block-wise transfer. U ovoj implementaciji je uočen problem sa block-wise transferom enkriptovanih poruka.

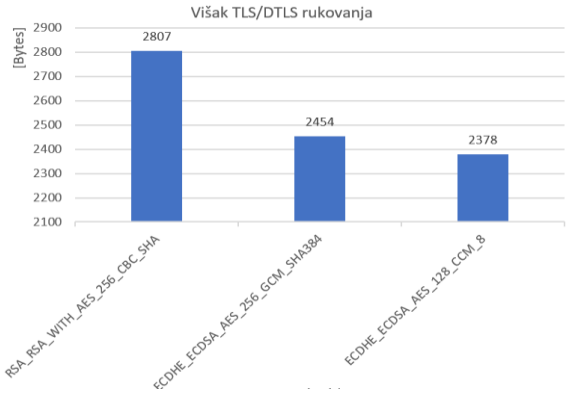
3.1. Analiza rezultata

Veličina sadržaja je ista za oba protokola, ali zavisi od formata poruke. Za prenos informacije preko male poruke, u Plain tekst, JSON, XML formatima je potrebno 39, 121, 401 bajtova, a za prenos informacije velike poruke se koristi 392, 1227 i 2563 bajtova.

Na grafiku 1. je prikazan ukupan višak koji korišćenje MQTT i CoAP protokola unosi u saobraćaj. Vidi se da prilikom transfera većih poruka protokoli nižeg nivoa MQTT steka unose po 54 bajta, i MQTT zaglavlje unosi 22 bajta. Protokoli koji se koriste uz CoAP unose veće opterećenje u saobraćaj, jer se velike poruke šalju u blokovima.

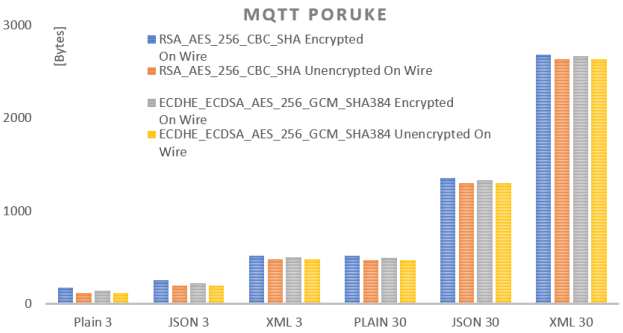


Grafik 1. *Ukupan broj prenetih bajtova u zavisnosti od protokola, bez prikazivanja bajtova sadržaja*

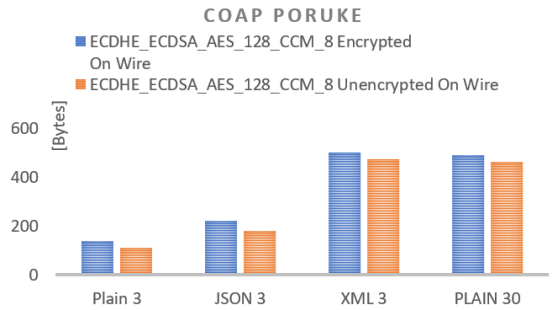


Grafik 2. *Višak TLS i DTLS rukovanja*

Grafik 2. pokazuje da je uočljivo da upotreba eliptičnih kriva (ECDSA) doprinosi manjem višku protokola rukovanja u odnosu na RSA.



Grafik 3. *Poređenje veličine šifrovanih i nešifrovanih MQTT poruka*



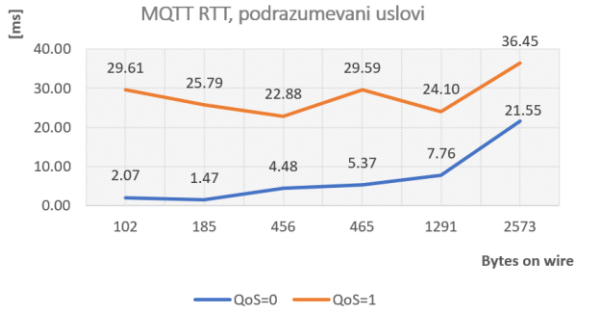
Grafik 4. *Poređenje veličine šifrovanih i nešifrovanih CoAP poruka*

Određivan je višak (Grafik 3, 4) na nivou pojedinačne poruke, za svaku od malih i velikih poruka prethodno korišćenih u eksperimentu. Na nivou razmenjivanih poruka za MQTT/TLS višak je skoro zanemarljiv, i iznosi prosečno 50 bajtova na nivou poruke za RSA_AES_256_CBC_SHA, dok za ECDHE_ECDSE_AES_256_GCM_SHA384 iznosi 29 bajtova. Višak na nivou poruke za

CoAP/DTLS za ECDHE_ECDSE_AES_128_CCM_8 iznosi 30 bajtova. Za CoAP/DTLS su prikazane samo 4 poruke, zato što su to poruke koje su dovoljno male da se ne šalju u blokovima i za njih korišćene implementacije funkcionišu.

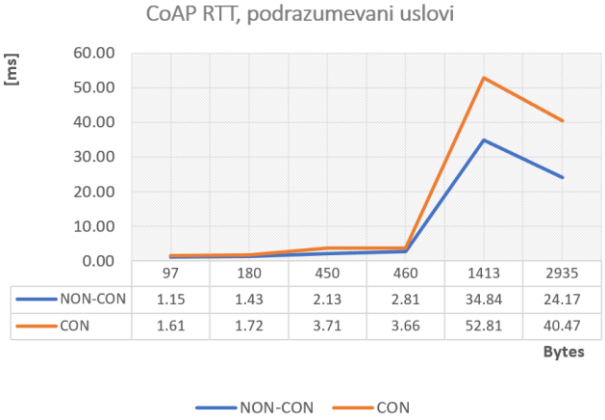
Latencija je merena kao funkcija veličine prenošene poruke, za oba protokola. Podrazumevani uslov je opseg od 85Mbps, kao i 0% gubitaka paketa. Za dobijanje rezultata, experiment je izvršavan 5 puta (N=5).

Grafik 5. pokazuje rezulate latencije za MQTT poruke različite veličine, u slučaju postavljanja vrednosti kvaliteta servisa na 0, odnosno na 1. Uočljivo je da je za QoS=1 latencija veća od QoS=0 za ~23ms. Skok na kraju kod prenošenja velike XML poruke je uzrokovan time što se na nižem nivou poruka prenosi iz dva IP paketa jer je MTU 1500 bajtova, dok je veličina prenošenog sadržaja veća.



Grafik 5. *Latencija MQTT poruka*

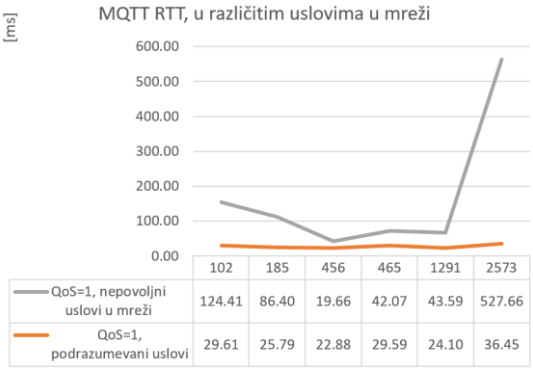
Uočava se i da na početku prenošenja poruka sa QoS=1 ne prati linearan trend kao kod QoS=0. Dolazi se do zaključka da implementaciji klijenta odnosno M2Mqtt biblioteki treba „vreme" da inicijalno počne da upravlja porukama sa višim kvalitetom servisa, učita potrebne module i/ili startuje procedure.



Grafik 6. *Latencija CoAP poruka*

Na grafiku 6. su predstavljeni rezultati za CoAP latenciju za slanje CoAP poruka koje ne očekuju potvrdu prispeća (NON-CON) i onih koje očekuju (CON) što odgovara MQTT kvalitetima servisa QoS 0 odnosno QoS 1 respektivno. U podrazumevanim uslovima, latencija je niska i za NON-CON i za CON poruke, do momenta slanja velike poruke koja mora da se fragmentuje i šalje u blokovima. Očekivano je duža latencija za NON-CON poruke. Takođe, bibilioteka ima problem sa procesiranjem enkriptovanih blokova. Izvršeno je po 5 iteracija za NON-CON i CON poruke, i trend je takav da je latencija duža za prvu poruku koja se transferuje iz blokova, u našem slučaju 1413 bajtova, nego za sledeću, duplo dužu poruku.

Latencija kao posledica nepovoljnih uslova u mreži je prikazana na graficima 7. i 8. Kvalitet servisa poruka je postavljen za MQTT QoS=1, CoAP CON, da bi se osigurali da će poruke sigurno doći na destinaciju. Uslovi su podešeni WANem-om tako da stopa gubitaka paketa u mreži bude 20% i da dostupni opseg bude 40 Mbps, što predstavlja oko 47% polaznog, podrazumevanog opseg 85Mbps. Oba grafika prikazuju, kao referentnu tačku, i latenciju u podrazumevanim uslovima, sa istim nivoom kvaliteta servisa.

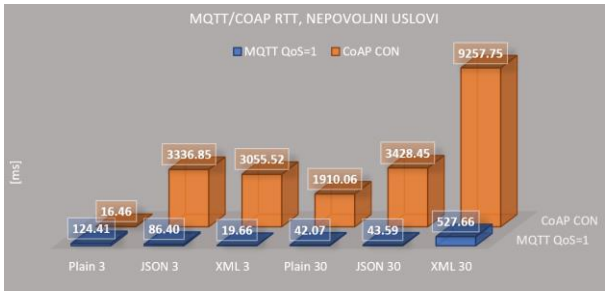


Grafik 7. *MQTT latencija u podrazumevanim i nepovoljnim uslovima u mreži*

Uočava se da je za MQTT (grafik 7), latencija porasla u uslovima mreže sa gubljenjem paketa (ima retransmisija) i smanjenjem opsega, pa tako imamo latenciju malo veću od 1/2s za velike XML poruke.

Grafik 8. *CoAP latencija u podrazumevanim i u mreži sa nepovoljnim uslovima*

Grafik 8. prikazuje latenciju kao posledicu nepovoljnih uslova u mreži, za CoAP poruke. Uočava se izuzetno visoka latencija, posebno za velike CON poruke, što se pripisuje gubljenju velikog broja paketa (20%).



Grafik 9. *MQTT i CoAP latencija u mreži sa nepovoljnim uslovima*

Na kraju (Grafik 9) je predstavljeno poređenje latencija za MQTT QoS=1, CoAP CON poruke, u mrežama sa nepovoljnim uslovima. Uočava se da je za velike poruke,

latencija za CoAP izuzetno visoka. Aplikativni sadržaj se deli na blokove od 512 bajtova, što znači da velika XML poruka zauzima >5 blokova , svaki se pakuje u zaseban UDP datagram i očekuje se potvrda dostavljanja. Sa stopom gubitaka od 20%, u smanjenom opsegu, ima puno retransmisija.

4. UPOTREBA IOT KOMUNIKACIONIH TEHNOLOGIJA ZA NADZOR BANDERA

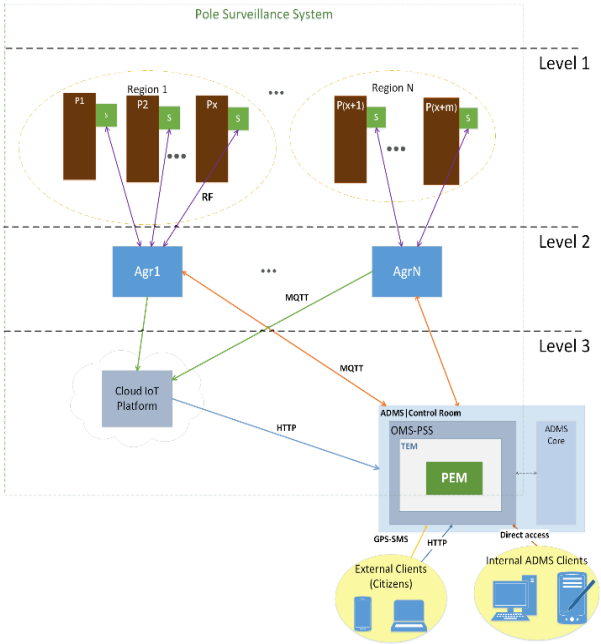
Rizik po bandere predstavljaju saobraćajne nesreće, oluje (sneg, vetar, led), poplave, zemljotresi, sadnice/drveće neodgovarajuće veličine u neposrednoj blizini, životinje (insekti, ptice). Padovi ili naginjanja bandera, zajedno sa propratnom opremom, predstavljaju incidente u mreži koji se mogu kategorisati na ispade (eng. *Outage*) koji dovode do deenergizacije i incidente koji predstavljaju smetnje/poremećaje u. Postoji rizik da čovek pokuša sam da popravi banderu, ili priđe srušenoj banderi sa svom njenom opremom smatrajući je deenergizovanom i na taj način se dovede u opasnost od strujnog udara. Zbog nepredvidive dinamike ljudskog faktora u okruženju bandera nužno je osigurati ih.

U pametnim mrežama se koristi sistem za upravljanje ispadima (eng. *Outage Management System* OMS), kao jedna od ključnih komponenti naprednog sistema za upravljanje distributivnom mrežom (eng. *Advanced Distribution Managament System* ADMS), za upravljanje svim planiranim i neplaniranim radovima u mreži, uključujući ispade, manipulisanje prekidačkom opremom, saniranje opasnosti, održavanje.

4.1. Predlog rešenja

Predlaže se sistem za online nadgledanje bandera (eng. *Pole Surveillance System* PSS) baziran na IoT tehnologijama, za rano detektovanje promena položaja bandera, kao i promena u okruženju odnosno potencijalnih incidenata/ispada u mreži uzrokovanih problemima sa banderama. Sistem je predložen kao proširenje tipičnih SG OMS sistema i namenjen je da se koristi integrisan sa ostalim komponentama rešenja za upravljanje ispadima. Fokus je na komponenti za upravljanje događajima sa bandera (eng. *Poles Events Manager* PEM) čija osnovna ideja je opametnjivanje bandera tako da kroz online nadzor, u slučaju naginjanja odnosno rušenja bandere, kontrolni centar/OMS može da bude obavešten/upozoren. Rešenje je hijerarhijsko i raspodela entiteta u rešenju je tronivojska. Na slici 2. je prikazan dijagram OMS-PSS sistema sa označenim komunikacionim tehnologijama koje se koriste između određenih komponenti.

Na nivou grada postoji veliki broj bandera, grupisanih u regije, na koje se instaliraju merni čvorovi. Za podskup geografski bliskih bandera, postoji nadređen agregator. Za agregatore je nadređen ADMS/OMS odnosno kontrolni čvor. Prisutna je opciona Cloud IoT platforma. Merni čvor omogućuje detekciju: nepogoda (požara, zemljotresa), pozicije (nagib/srušenost, lokacija) bandere. Slanje podataka sa mernih čvorova ka agregatorima se vrši modulima radio tehnologije (RF) modulima koji mogu da imaju domet i do 5km. Agregator dobavlja podatke sa mernih čvorova, obrađuje ih i potom prosleđuje u kontrolni centar upotrebom LTE, WiMAX, WiFi ili žične tehnologije. Za primanje podataka sa podređenih mernih čvorova koristi RX RF modul. Kontrolna soba ima potpuni uvid u stanje svih ispada u mreži. Web aplikacija OMS-PSS sistema omogućava korisnicima da pristupe ažurnim informacijama o trenutnim problemima. Preko istog portala je moguće i prijavljivanje problema u mreži.



Slika 2. – Globalni prikaz OMS-PSS sistema

Komunikacija između mernih čvorova i agregatora je dvosmerna, odnosno postoje dva RF modula (RX i TX). Za komunikaciju se koristi UART. Komunikacija između agregatora i kontrolnog centra ili clouda, se dešava preko WiFi, LTE, ili WiMAX tehnologije. Aplikativni protokol je MQTT.

5. ZAKLJUČAK

Zaključak je da su arhitekture IoT rešenja kombinovane i na određenim nivoima komunikacije su uposleni protokoli bazirani na drugačijim komunikacionim modelima. U vezi sa eksperimentalnim delom, u daljem radu je poželjno testirati protokole podešavanjem veličine MTU-a, tako da odgovara različitim protokolima nižih nivoa. Implementirati custom enkripciju za MQTT i koristiti OSCORE za CoAP. Cilj je temeljnije se baviti problemom bezbednosti.

6. LITERATURA

[1] Z. Shelby, K. Hatke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC Editor, 2014

[2] [mqtt-v3.1.1] MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS Standard., 2014.

Kratka biografija:

Miljana Vujaković je rođena 14.09.1994. godine u Novom Sadu. Fakultet tehničih nauka u Novom Sadu je upisala školske 2013/2014. godine. Diplomirala je 2018. godine na odseku za Elektrotehniku i računarstvo, smer „Primenjeno softversko inženjerstvo", i iste godine je upisala master studije na smeru „Primenjeno softversko inženjerstvo".

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.38:620.9
DOI: https://doi.org/10.24867/10BE12Ivanovic

VERIFIKACIJA MODULACIONIH STRATEGIJA MAŠINA SA OTVORENIM KRAJEVIMA NAMOTAJA PRIMENOM FPGA I HIL UREĐAJA

VERIFICATION OF MOULATION STRATEGIES FOR OPEN END WINDING MACHINES USING FPGA AND HIL DEVICES

Aleksandar Ivanović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu su prezentovani rezultati verifikacije raspregnutih modulacionih strategija za upravljanje dva invertora koji ostvaruju dva nivoa izlaznog napona i napajaju mašinu sa otvorenim krajevima namotaja. Prvo je data teorijska analiza i način implementacije predloženih modulacionih strategija. Zatim je vršena eksperimentalna verifikacija prezentovanih modulacionih strategija. Eksperimentalna verifikacija je vršena pomoću FPGA i HIL uređaja.*

Ključne reči: *Modulacione strategije, Mašina sa otvorenim krajevima namotaja, FPGA, HIL*

Abstract – *In this paper verification results, of modulation strategies for open-end winding machine supplied by two two-level inverters, are presented. Firstly theoritical basics and implementation of used modulation strategies is presented. Modulation startegies were then tested on the real system, using FPGA and HIL devices.*

Keywords: *Modulation strategies, Open-end winding, FPGA, HIL*

1. UVOD

Tema ovog rada jeste eksperimentalna verifikacija modulacionih strategija na mašinama sa otvorenim krajevima namotaja. Kontrolni algoritam će se izvršavati na *Field Programmable Gate Array* (*FPGA*) uređaju - *ZedBoard*, dok će se mašina sa otvorenim krajevima namotaja simulirati preko *Typhoon HIL600* uređaja. Ovi uređaji komuniciraju preko prilagodne kartice koju je bilo neophodno projektovati. Mašina sa otvorenim krajevima namotaja će se napajati obostrano sa invertorom koji ostvaruje dva nivoa izlaznog napona. Za ovaj pogon će se vršiti poređenje predloženih modulacionih strategija. Konačno, harmonijske performanse predloženih modulacionih strategija datog pogona biće prezentovane.

1.1 Mašine sa otvorenim krajevima namotaja

Mašine sa otvorenim krajevima namotaja, kako im ime kaže, umesto formiranja konfiguracije sa prevezivanjem statorskih namotaja, imaju otvorene krajeve namotaja sa obe strane. Samim tim ove mašine se napajaju sa obe strane namotaja. Principska šema mašine sa otvorenim krajevima namotaja prikazana je na slici 1.

Slika 1. Principska šema mašine sa otvorenim krajevima namotaja

Pokazano je da napajanjem mašine sa otvorenim krajevima namotaja sa dva dvonivovska invertora se može postići rad sa više naponskih nivoa [1]. Kod mašina sa otvorenim krajevima namotaja, ukupan broj nivoa faznog napona, ne zavisi samo od broja naponskih nivoa individualnih invertora, već i od odnosa napona dc kola koja napajaju te invertore. Kod mašine koja je napajana sa po dva dvonivovska invertora, jednakih dc napona, ukupan broj naponskih nivoa jednak je kao i kod mašine napajane sa jedne strane tronivovskim invertorom. Ukoliko je odnos dc napona jednak 2:1, ukupan broj naponskih nivoa je jednak kao i kod invetora sa četiri nivoa. Upravo to predstavlja i prednost mašina sa otvorenim krajevima namotaja, iste performanse *multilevel* rada se postižu sa jednostavnijim i robusnijim topologijama, što dovodi do povećanja pouzdanosti.

U [2] su predstavljene dve raspregnute modulacione strategije, kada su invertori napajani sa dc naponima različitih odnosa, 2:1. U prvoj se naponske reference dele u odnosu 2:1 dok im je frekvencija prekidanja ista, dok se kod druge strategije, frekvencija prekidanja deli u odnosu 2:1. Dobar pregled modulacionih metoda za mašine sa otvorenim krajevima namotaja se može naći u [3].

Takođe, pokazano je da se kod ovih pogona, na račun broja naponskih nivoa može postići eliminacija *common-mode* napona i balansiranje kondenzatora dc kola. Ovo je od posebnog značaja za smanjenje gubitaka koje *common-mode* napon stvara. U [4] mašina sa otvorenim krajevima namotaja se napaja sa dva petonivovska invertora, koji su sačinjeni od kaskadne veze tronivovskog i dvonivovskog invertora, gde se u potpunosti eliminiše *common-mode* napon.

1.2. Primena *FPGA* u energetskoj elektronici

Danas i dalje, najveći broj digitalnih kontrolera je zasnovan na *Digital Signal Processing* (*DSP*) procesorima ili mikrokontrolerima. Međutim ova rešenja imaju dva glavna nedostatka, to su vreme izvršavanja kontrolnog algoritma i nedostatak fleksibilnosti u povezivanju sa analognim okruženjem. Oba ova nedostatka se mogu prevazići upot-

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Stevan Grabić , vanr. prof.

rebom *FPGA* i pažljivim deljenjem algoritamskih zadataka između hardverskih i softverskih komponenti. Pored toga, lako se rekonfigurišu, imaju manju potrošnju energije i relativno brži proces razvoja [5].

Pokazano je da kontroler zasnovan na *FPGA* može biti efikasan odabir za visoko zahtevne aplikacije, gde su frekvencije prekidanja veće od $100kHz$, ali i za aplikacije sa gde su prisutne niže frekvencije prekidanja. U prvu grupu spadaju aplikacije gde je priustna potreba za specifičnim hardverom, aplikacije u kojima postoji visok nivo paralelizma kao i *HIL* aplikacije. Što se tiče druge grupe, može doći do značajnih poboljšanja u vremenu potrebnom za izvršavanje algoritma, a samim tim i do boljih performansi pogona. Detaljniji opis primene u ovoj oblasti se može naći u [6]. Neki od primena *FPGA* u industriji, kao što su *HIL* simulatori, napredni kontrolni algoritmi, kao i eksperimentalni rezultati predstavljeni su u [7].

Međutim, i pored brojnih pogodnosti i mogućih primena, *FPGA* digitalni sistemi se retko koriste u energetici. Još uvek, glavno ograničenje je cena. Pored toga, korišćenje *FPGA* podrazumeva dobro znanje i hardvera i softvera, zatim znanje potrebno za integraciju, kao i efikasnu podelu algoritamskih zadataka između ova dva domena. Ovo, zajedno uz činjenicu da se inženjeri često drže poznatih rešenja, predstavlja prepreku za veću primenu *FPGA*.

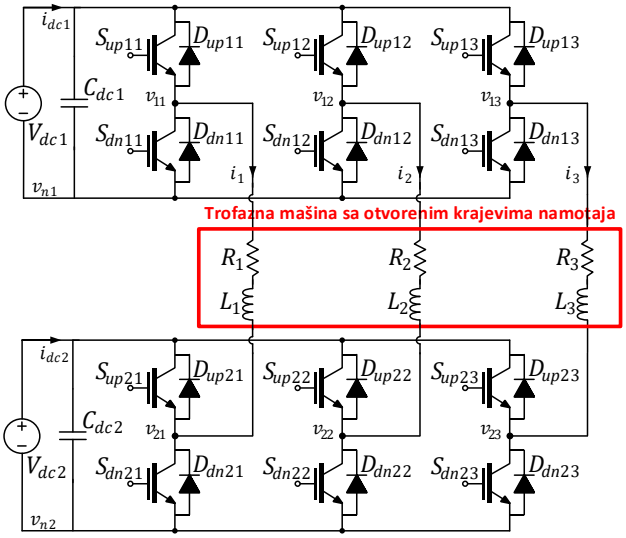
1.3 *Hardware in loop*

HIL simulacija, najopštije rečeno, je tehnika koja se koristi za razvoj i testiranje kompleksnih *embedded* sistema u realnom vremenu. Iz perspektive kontrolera, ne postoji razlika između realnog i simuliranog hardvera. Samim tim ova tehnika omogućava testiranje kontolnog algoritma a da pri tom ni prototip konvertera nije izrađen. Performanse simulacije zavise od mogućnosti *HIL* uređaja ali i od kompleksnosti matematičkih modela hardvera koji se simuliraju sa *HIL*-om.

HIL tehnika se danas koristi u sve više industrija. Najčešće primene su u oblastima energetske elektronike i energetskih sistema [8]. Današnji *HIL* uređaji se sve više fokusiraju na primenu u oblasti elektroenergetskih mikromreža. Takođe, sve više je prisutna primena *HIL* uređaja u automobilskoj industriji, gde *HIL* uređaji mogu da simuliraju više *Electronic Control Unit* (*ECU*) jedinica od jednom i služe za testiranje kompleksnog električnog sistema automobila.

2. TROFAZNA MAŠINA SA OTVORENIM KRAJEVIMA NAMOTAJA NAPAJANA SA DVA INVERTORA SA SVA NIVOA

Topologija koja se razmatra za početak je prikazana na slici 2. Sastoji se od dva invertora sa dva nivoa, koji su napajani sa dva dc izvora (V_{dc1} *i* V_{dc2}) i trofazne električne mašine sa otvorenim krajevima namotaja. Izlazni naponi invertora su označeni sa v_{1k} i v_{2k}, gde broj k predstavlja broj faze. Fazne struje su označene sa i_k. Komponente na šemi su S_{pojk} i D_{pojk}, i predstavljaju prekidače i diode. Supskript "po" označava poziciju komponente u grani invertora, "up" za gore i "dn" za dole. I na kraju, "j" označava o kom se invertoru radi (1 ili 2) i "k" predstavlja broj faze mašine (1, 2 ili 3).



Slika 2. Trofazna šema korišćene topologije

Za razvinjanje modulacione strategije, naponski invertori se najčešće posmatraju kao pojačavači *PWM* signala, gde je *IGBT* i dioda struktura posmatrana kao idealni bidirekcioni prekidač. U većini slučajeva ovaj pristup je sasvim opravdan. Analiza se može značajno pojednostaviti korišćenjem prekidačkih stanja, koja predstavljaju *IGBT* i diodu kao par, umesto da se svaka od ovih poluprovodničkih komponenti posmatra posebno.

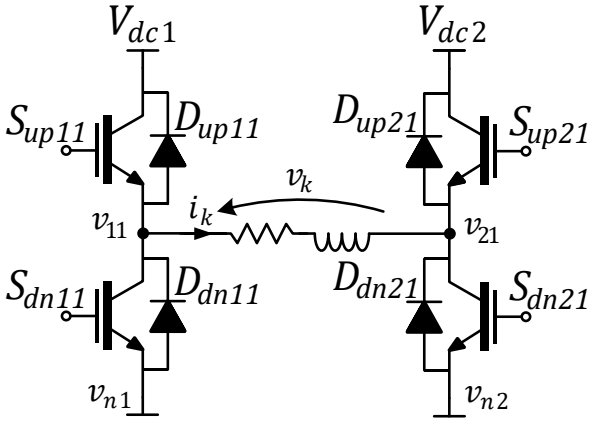
Konačna formula za fazni napon predstavlja razliku izlaznih napona dva invertora i *common-mode* napona:

$$v_k = v_{1k} - v_{2k} - v_{n2n1}, \qquad (1)$$

gde se *common-mode* napon računa kao:

$$v_{n2n1} = v_{n2} - v_{n1} = \frac{1}{3}\sum_{k=1}^{3}(v_{1k} - v_{2k}). \qquad (2)$$

Na ovaj način, napon na namotaju i *common-mode* napon su izraženi kao funkcija izlaznih napona invertora koji su u potpunosti upravljivi. Jedna grana invertora može da proizvede dva različita napona, 0 i V_{dc} što odgovara stanjima 1 i 0. Dolazi se do zaključka da postoje četiri moguće kombinacije prekidača po fazi. Nadalje, kompleksni model invertora sa mašinom će se posmatrati kao jednostavniji, tako što će se posmatrati samo jedna faza, što je prikazano na slici 3**Error! Reference source not found.**.



Slika 3. Jednofazna šema korišćene topologije

Pod pretpostavkom da su donji potencijali dc kola isti ($v_{n2} = v_{n1}$), može se doći do naponskih nivoa faznog napona. Kako bi se generalizovala i pojednostavila analiza, uvodi se smena:

$$r = \frac{V_{dc1}}{V_{dc2}}, \tag{3}$$

dok se ukupni napon dc kola može napisati kao:

$$V_{dc} = V_{dc1} + V_{dc2}. \tag{4}$$

Ovi naponi su dostupni u tabeli 1.

N	S1	S2	V_{1k}	V_{2k}	V_k
1	1	1	$r/(r+1)$	$1/(r+1)$	$(r-1)/(r+1)$
2	1	0	$r/(r+1)$	0	$r/(r+1)$
3	0	1	0	$1/(r+1)$	$-1/(r+1)$
4	0	0	0	0	0

Tabela 1

3. RASPREGNUTE METODE MODULACIJE

Jedan od pristupa prilikom razvoja modulacione strategije za pogon mašine sa otvorenim krajevima namotaja, je da se dva naponska invertora posmatraju kao zasebni moduli. S obzirom da je mašina napajana sa dve strane i uzimajući u obzir jednačinu (1), moguće je ostvariti fazni napon sa više naponskih nivoa. Ideja je da se upravljanje celog sistema razloži na dva podsistema, manje kompleksnosti, tako što će se ukupna referenca podeliti na dve reference, po jedna za svaki invertor.

Ukoliko bi se referenca napona delila na osnovu ukupnog modulacionog indeksa govori se o metodama nejednakog deljnja reference, na engleskom *Unequal Voltage Sharing (URS)* [9]. Pimenom *URS* modulacije, postoje dve oblasti upravljanja invertorima. Kada je $M \leq M_{max}/2$, gde je M modulacioni indeks, jednim invertorom se upravlja, dok drugi služi da formira zvezdu. U tok slučaju pogon je identičan kao kada je mašina napajana jednostrano, sa jednim invertorom. U drugom slučaju, kada je $M \geq M_{max}/2$, jedan invertor se drži na $M = M_{max}$, dok se drugim invertorom upravlja. Tada je prisutan multilevel rad.

Referenca faznog napona u slučaju V/f *open-loop* kontrole se kreira na osnovu sledeće jednačine:

$$v_k^* = \frac{M}{2}\sin\left(M \cdot \omega_n \cdot t - \frac{2 \cdot \pi}{2} \cdot k\right) \tag{5}$$

Zajedničko za sve raspregnute metode je deljenje originalne naponske reference na dva dela, kako bi se ona podelila između dva naponska invertora. Modulacija se vrši kasnije, posebno za v_{1k}^* i v_{2k}^*, koje predstavljaju reference za izlazne napone naponskih invertora.

Pogon se može upravljati sa samo jednim invertorom za indeks modulacije $M < M_{max}/(r+1)$. Označimo sa 1 invertor koji radi u niskim modulacionim indeksima, dok je sa 2 označen invertor koji radi u *PWM* modu, za $M > M_{max}/(r+1)$. Na osnovu ovoga može se zapisati na koji način će se računati reference:

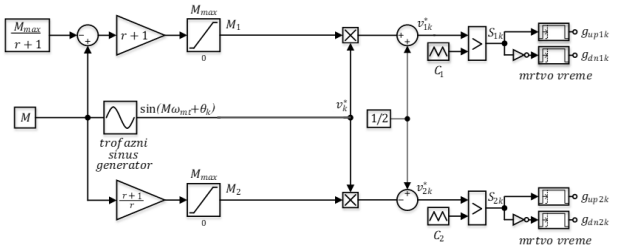
$$M_1 = \begin{cases} 0, & 0 \leq M \leq \frac{M_{max}}{(r+1)} \\ (r+1)\left(M - \frac{M_{max}}{(r+1)}\right), & \frac{M_{max}}{(r+1)} \leq M \leq M_{max} \end{cases} \tag{6}$$

$$M_2 = \begin{cases} (r+1) \cdot M, & 0 \leq M \leq \frac{M_{max}}{(r+1)} \\ M_{max}, & \frac{M_{max}}{(r+1)} \leq M \leq M_{max} \end{cases}$$

Dobijeni indeksi modulacije se koriste za dobijanje para referenci za dva naponska invertora. M_1 i M_2 predstavljaju samo amplitude za reference. Što se tiče frekvencije referenci, ona se podešava zasebno i figuriše u M u jednačini (5). Konačan izraz za reference v_1^* i v_2^* glasi:

$$\begin{aligned} v_1^*(t) &= \frac{1}{2} + \frac{M_1}{M} \cdot v_k^*(t) \\ v_2^*(t) &= \frac{1}{2} + \frac{M_2}{M} \cdot v_k^*(t) \end{aligned} \tag{7}$$

Dobijene jednačine (6) i (7) predstavljaju metodu nejednakog deljenja referenci – *URS*. Blok dijagram ove metode nalazi se na slici 4. Sa C_1 i C_2 su označeni noseći signali za invertor 1 i 2. Poređenjem sa referencom, generišu se upravljački signali za prekidače (g_{up1k}, g_{dn1k}, g_{up2k}, g_{dn2k}) i dodaje mrtvo vreme.

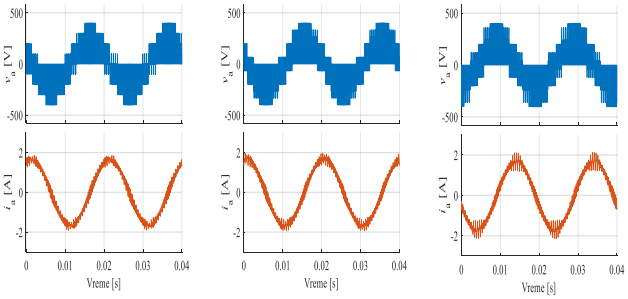


Slika 4. Blok dijagram *URS* modulacije

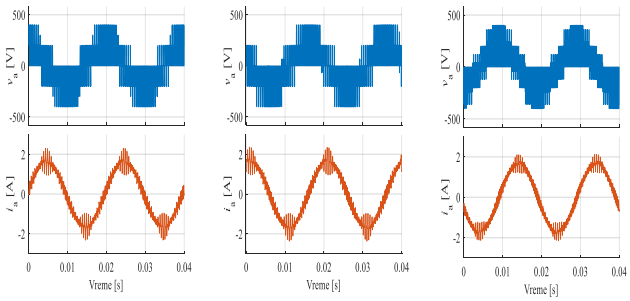
Kod ove metode, u zavisnosti od toga da li su noseći signali u fazi ili ne, postoje dve modulacione strategije. Kod jedne su noseći signali u fazi (*URS1*), dok su kod druge noseći signali pomereni za pola *PWM* periode (*URS2*).

3. VERIFIKACIJA MODULACIONE STRATEGIJE NA FPGA I HIL UREĐAJU

Eksperimentalna postavka se sastoji od *FPGA* (*ZedBoard*), *HIL600* i prilagodne kartice koja povezuje ova dva sistema. *ZedBoard* služi za izvršavanje *open-loop* algoritma i izračunavanje potrebnih faktora ispune za prekidače u invertorima. Na *HIL600* se nalazi model mašine sa otvorenim krajevima namotaja i dva invertora koji napajaju mašinu sa obe strane. Dobijeni rezultati za različite odnose dc napona ($r = 1, r = 2, r = 4$) i za modulacioni indeks jednak jedinici, primenom strategija *URS1* i *URS2* prikazani su na slikama 5 i 6.

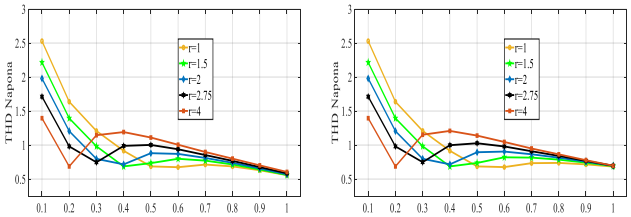


Slika 5. Fazni naponi i struje dobijeni primenom *URS1*

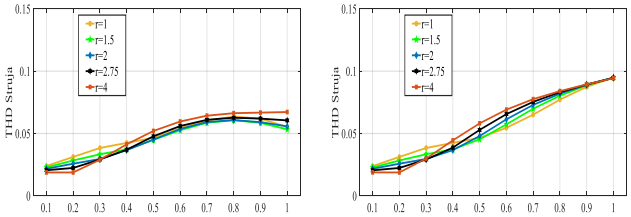


Slika 6. Fazni naponi i struje dobijeni primenom *URS2*

Harmonijske performanse pogona, u odnosu na dc napone invertora, biće sagledane za *URS1* i *URS2* metode. Sa slika 7 i 8 vidi se da prilikom tranzicije sa rada sa dva naponska nivoa na rad sa više naponskih nivoa, dolazi do povećanja harmonijskog izobličenja. Ovo je posebno izraženo za veće r. Dodatno na slikama 7 i 8 su prikazane harmonijske analize za još dva različita odnosa dc napona, $r = 1.75$ i $r = 2.75$. Pri $r = 1$, granična vrednost modulacionog indeksa pri kojoj se prelazi na rad sa više nivoa izlaznog napona, iznosi $M = 0.525$. Kada je $M > 0.525$, drugi invertor, koji ima napon dc kola $300V$ počinje sa radom sa preostalim modulacionim indeksom, koji je za slučaj kada je $M = 0.6$, jednak $M_2 = 0.15$. Do harmonijskog izobličenja dolazi usled efekta mrtvog vremena i malog modulacionog indeksa.



Slika 7 Odnosi *THD* napona za *URS1* i *URS2*



Slika 8 Odnosi *THD* struja za *URS1* i *URS2*

Kada su u pitanju *URS1* i *URS2* metode, ono što se može videti sa slika 5, 6, 7 i 8 je to da se niži *THD* napona i struja ostvaruje primenom *URS1* metode. Što se tiče odabira optimalnog odnosa dc napona, može se zaključiti da je najbolji odabir od prikazanih metoda, za $r = 2$.

3. ZAKLJUČAK

U ovom radu je analizirana topologija gde je mašina sa otvorenim krajevima namotaja napajana sa dva invertora koji ostvaruju dva nivoa izlaznog napona. Prvo je topologija modelovana, svođenjem na jednu fazu, a potom i analizirana. Predložene su URS1 i *URS2* raspregnute modulacione strategije zajedno sa odgovarajućim jednačinama za njihovo ostvarenje.

Performanse pogona verifikiovane su na eksperimentalnoj postavci, korišćenjem *HIL*-a, *ZedBoard*-a i projektovane prilagodne kartice. Obe metode su analizirane u otvorenoj povratnoj sprezi, korišćenjem U/f upravljanja. Kao stepen slobode uzet je odnos dc napona dva invertora. Iz prikazanih rezultata vidi se da *URS1* modulaciona strategija unosi manje harmonijsko izobličenje, tj. da su *THD* struja i napona manji nego u slučaju *URS2* modulacione strategije Takođe, iz dobijenih rezultata se nameće da je optimalni odnos napona dc kola invertora $r = 2$.

4. LITERATURA

[1] E.G. Shivakumar ; K. Gopakumar ; S.K. Sinha ; A. Pittet ; V.T. Ranganathan, *"Space vector PWM control of dual inverter fed open-end winding induction motor drive"*, Sixteenth Annual IEEE Applied Power Electronics Conference and Exposition, APEC 2001.

[2] Barry Venugopal Reddy ; Veeramraju Timurala Somasekhar ; Yenduri Kalyan, *" Decoupled Space-Vector PWM Strategies for a Four-Level Asymmetrical Open-End Winding Induction Motor Drive With Waveform Symmetries "*, IEEE Transactions on Industrial Electronics, Volume: 58 , Issue: 11, Nov. 2011.

[3] Darijevic, M., "Modulation and control strategies for multilevel five-phase open-end winding drives", PhD. thesis, Liverpool John Moores University, 2016

[4] Gopal Mondal, K. Gopakumar, P. N. Tekwani, Emil Levi, "A Reduced-Switch-Count Five-Level Inverter With Common-Mode Voltage Elimination for an Open-End Winding Induction Motor Drive" IEEE Transactions on Industrial Electronics, Volume: 54 , Issue: 4 , Aug. 2007

[5] Concettina Buccella, Carlo Cecati, Hamed Latafat, "Digital Control of Power Converters—A Survey", IEEE Transactions on Industrial Informatics, Volume: 8, Issue: 3, Aug. 2012

[6] Eric Monmasson, Lahoucine Idkhajine, Mohamed Wissem Naouar "FPGA-based Controllers", IEEE Industrial Electronics Magazine, Volume: 5 , Issue: 1 , March 2011

[7] E. Monmasson, L. Idkhajine, I. Bahri, M-W- Naouar, L. Charaabi, "Design methodology and FPGA-based controllers for Power Electronics and drive applications", 2010 5th IEEE Conference on Industrial Electronics and Applications, 15-17 June 2010

[8] Simon Abourida, Christian Dufour, Jean Bélanger, "Hardware-In-the-Loop Simulation of Power Drives with RT-LAB", Power Electronics and Drives Systems, International Conference on, Volume: 2, 2005

[9] Bodo, N., "PWM strategies for open-end winding multiphase drives", PhD. thesis, Liverpool John Moores University, 2013

Kratka biografija:

Kontakt:
acikaca@gmail.com

Aleksandar Ivanović rođen je u Šapcu 1993. godine. Godine 2012 upisuje osnovne akademske studije na studijskom programu Energetika, elektronika i telekomunikacije. 2016 godine završava osnovne i upisuje master akademske studije na istom studijskom programu

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.738.1:766
DOI: https://doi.org/10.24867/10BE13Bokic

SISTEM ZA DETEKCIJU PLAGIJARIZAMA U DOKUMENTIMA PISANIM NA SRPSKOM JEZIKU

SYSTEM FOR DETECTION OF PLAGIARISM IN DOCUMENTS WRITTEN IN SERBIAN

Stefan Bokić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Rad opisuje arhitekturu i implementaciju sistema za proveru naučnih radova (diplomski, master, doktorat). U sistemu je moguće dodavati naučne radove i proveravati da li postoje slični radovi u kolekciji. Ako postoje slični radovi, koliko su slični. Takođe nastavnici imaju mogućnost da provere da li ima previše poklapanja uvidom u naučne radove.*

Ključne reči: *Elasticsearch, plagijarizam, naučni radovi*

Abstract – *This paper describes the architecture and implementation of the system for checking scientific papers (graduate, master, doctoral). It is possible to add scientific papers to the system and check if there are similar papers in the collection. If there are similar papers, how similar are they. Professors also have the opportunity to check if there are too many matches by looking at scientific papers.*

Keywords: *Elasticsearch, plagiarism, scientific papers*

1. UVOD

Razvoj tehnologija uslovio je njihovu sve veću integraciju u ljudske živote. Takođe sa razvojem interneta dolazi do transformisanja načina na koji funkcionišu komunikacioni sistemi. S obzirom da je internet dostupan svima, a na internetu se nalazi veliki broj informacija, dolazi se do situacije da je jednim klikom moguće kopirati informacije sa interneta. Zbog toga je od izuzetne važnosti razlikovati verodostojne naučne radove i plagijate. Na taj način se doprinosi kredibilitetu naučnih radova

2. TEORIJSKE OSNOVE

Plagijarizam predstavlja skup algoritama koje omogućavaju proveru radova, tj. pokazuje da li jedan rad predstavlja kopiju drugog. Zašto je važno u današnjem svetu razlikovati prave radove od njihovih kopija, plagijata?! Danas plagijarizam predstavlja veliki problem u profesionalnom i edukativnom svetu. S obzirom da je internet dostupan svima, i da se veliki izvor informacija nalazi na internetu, vrlo je lako kopirati te informacije. Plagijarizam se takođe odnosi na upotrebu tuđih informacija, jezika ili pisanja, upotrebljenih bez odgovarajuće potvrde originalnog izvora [1, 2].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dragan Ivanović, vanr. prof.

Plagijarizam u okviru tekstualnih dokumenata može se pojaviti u nekoliko oblika [1, 2]:

- plagirani tekst se može kopirati jedan na jedan
- odlomci se mogu kopirati u većem ili manjem obimu
- odlomci se mogu prevesti sa stranih jezika

Uspeh postojećih pristupa u otkrivanju plagijata teksta zavisi od načina plagijarizma, tj. koji od tri navedena načina je upotrebljen. Postoje tri glavne ideje koje se mogu iskoristi za traženje plagijata [3]:

- *substring matching* – podudaranje delova teksta
- *keyword similarity* – sličnost ključnih reči
- *fingerprint analysis* – generisanje digitalnog otiska

Podudaranje delova teksta pokušava da identifikuje maksimalno poklapanja delova teksta plagijata i originalnog teksta [3].

Kod pristupa sličnosti ključnih reči, ideja je da se identifikuju značajne reči iz dokumenta i da se uporede sa značajnim rečima drugih dokumenata. Ako sličnost premaši zadati prag, dokument predstavlja plagijat.

Dokument je podeljen u delove koji se rekurzivno porede. Ovaj algoritam pretpostavlja da se plagijarizam dešava u dokumentima sa sličnim temama [3].

Kod pristupa generisanje digitalnog otiska, koristi se kriptografska heš funkcija. Dokumenti su podeljeni u delove teksta. Heš funkcija služi za generisanje digitalnih otisaka dela teksta (niza termova), koji se zatim upoređuje sa bazom originalnog teksta. S obzirom da heš funkcija identifikuje niz reči jedinstveno, to znači da kvalitet ovog pristupa zavisi od veličine delova teksta i odstupanja od originalnog teksta [3].

3. KORIŠĆENE TEHNOLOGIJE

Java je programski jezik opšte namene koji je zasnovan na klasama i objektno orijentisanom programiranju. Pored toga Java je osmišljena da ima što je moguće manje zavisnosti od implementacije. Dakle, glavna ideja tvoraca Jave je da kod jednom napisan može da se pokrene sa bilo koje mašine.
To znači da kod iskompajliran u Javi može da se pokrene na bilo kojoj platfomi koja podržava Javu [5].

Spring radni okvir je platforma koja pruža sveobuhvatnu infrastrukturnu podršku za razvoj Java aplikacija. Spring upravlja sa infrastrukturom aplikacije, tako da se programeri mogu skoncentrisati na poslovnu logiku svoje aplikacije. Ovo je omogućeno kroz korišćenje objekata (engl. *Plain old Java objects* - POJO), anotacija i konfiguracijonih fajlova. Spring je zamišljen kao lagan radni okvir. Dva glavna koncepta na kojima se bazira Spring su [6]:

- injektovanje zavisnosti (engl. *dependency injection – DI*) [7]
- aspekti orijentisanog programiranja (engl. *aspect-oriented programming* - AOP) [8]

Elasticsearch je pretraživač otvorenog koda koji nudi pretragu u gotovo realnom vremenu, mogućnost pretraživanja celokupnog teksta, kao i RESTful API. Dakle, *Elasticsearch* je pretraživač celokupnog teksta napisan u Javi, koji je osmišljen da bude distributivan, skalabilan i da vraća rezultate pretrage u realnom vremenu. Takođe, *Elasticsearch* server je jednostavan za instalaciju, može se podići bez ikakvog podešavanja, mada većina korisnika želi da podešava funkcionalnosti što je isto moguće kroz podešavanje raznih parametara. Pokrenuta instanca *Elasticsearch*-a se naziva čvor, dva ili više čvorova formiraju klaster. Da bi se postavio klaster potrebno mu je samo zadati ime. *Elasticsearch* će se sam pobrinuti za otkrivanje čvorova na mreži i povezati ih u klaster [9].

Angular predstavlja radni okvir za dizajniranje aplikacija i razvojnu platformu za kreiranje efikasnih i sofisticiranih jednostraničnih aplikacija (engl. *single page aplication* [12]). Angular je poput mnogih drugih veb radnih okvira baziran na komponentama. To znači da su komponente glavni gradivni blokovi u Angular-u. Blokovi mogu da prikazuju informacije, renderuju šablone i izvršavaju akcije nad podacima. Najbolje prakse sugerišu da bi komponente trebale da se sastoje iz tri dela [10, 11]:

- HTML fajl za šablone
- CSS fajl za stilove
- *TypeScript* fajl za kontrolu

4. SPECIFIKACIJA

Sistem za detekciju plagijata je veb aplikacija namenjena za više korisnika koji uz pomoć aplikacije mogu da unose seminarske i završne radove različitih nivoa studija (diplomski, master ili doktorski rad) u sistem i proveravaju poklapanja sa drugim radovima. Sistem za detekciju plagijata olakšava i ubrzava provere prepisivanja sa drugim radovima. Kako radi aplikacija:

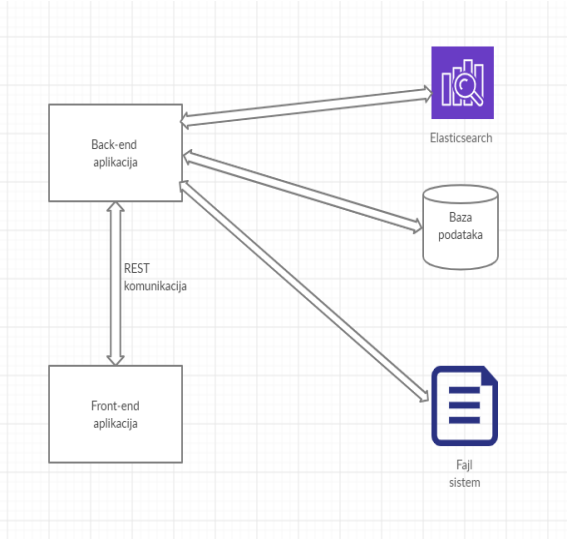
- Potrebno je da korisnik napravi svoj nalog
- Nakon toga je moguće logovanje na aplikaciju
- Moguće je otrpemanje fajlova koji ne prolaze kroz proveru plagijarizma, tj. oni su već objavljeni
- Nakon napunjene baze sa naučnim radovima, je moguće vršiti proveru na plagijate
- Po otpremanju novog fajla, dobija se rezultat provere na kom su prikazani najsličniji rezultati
- Ukoliko korisnik želi može da pristupi detaljima određenog rada gde su prikazane sličnosti između dva dokumenta po delovima
- Postoji mogućnost preuzimanja oba rada kako bi korisnik ručno proverio poklapanja i napisao komentar za ta dva rada

4.1. Arhitektura sistema

Sistem se sastoji iz sledećih delova:

- *Backend* aplikacija
- *Frontend* aplikacija
- Baza podataka
- Fajl sistem
- *Elasticsearch*

Na slici 1. je prikazana arhitektura sistema koja je prethodno opisana.



Slika 1. *Arhitektura sistema*

Backend aplikacija predstavlja glavni deo ovog sistema u kojoj se nalazi sva poslovna logika. Takođe ona obezbeđuje servise svim prethodno navedenim delovima aplikacije. Realizuje se kao servisno orijentisana veb aplikacija. *Backend* aplikacija je implementirana pomoću Spring radnog okvira.

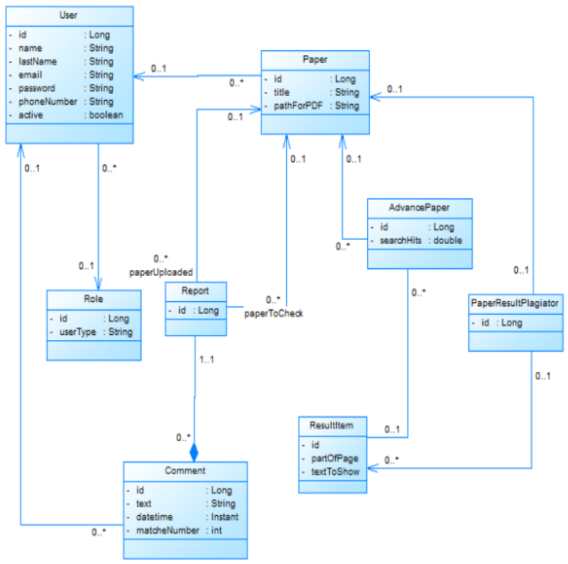
Frontend aplikacija obezbeđuje interfejs i funkcionalnosti koje su neophodne korisnicima. *Frontend* aplikacija je implementirana pomoću Angular radnog okvira.

Baza podataka se koristi za skladištenje podataka vezanih za korisnike, metapodatke o naučnim radovima, kao i putanje na fajl sistemu gde su uskladišteni otpremljeni naučni radovi.

Fajl sistem se koristi za skladištenje naučnih radova.

Elasticsearch se koristi za skladištenje metapodataka o naučnim radovima, pretprocesiranih tekstova izvučenih iz naučnih radova, njihovu obradu i odgovor na upite koji su vezani za sličnost između naučnih radova.

Na slici 2. je prikazan model klasa sistema koji je zadužen za fizičku reprezentaciju tabela u bazi podataka.



Slika 2. *Dijagram klasa sistema*

5. IMPLEMENTACIJA

Ono što je od najvećeg interesa je pitanje na koji način se podaci čuvaju u *Elasticsearch*-u. U okviru podešavanja *Elasticsearch*-a je podešen analizator (engl. *analyzer*) koji je zadužen za analiziranje teksta koji stiže na *Elasticsearch* server. U okviru njega je ubačen *Shingle token filter* [13]. Koji je podešen da od teksta pravi niz reči od veličine četiri pa do veličine šest reči. Parametri koji su zaduženi za to su *min_shingle_size* i *max_shingle_size*. Takođe pored toga je podešeno da ne želimo da se dodatno prave reči od veličine jedan. Šta to znači? Pogledajmo na sledećem primeru. Ako imamo rečenicu *Danas je lep i sunčan dan*. Ova rečenica se sastoji od šest reči. Ako bi se primenio *Shingle token filter* koji je podešen izlaz bi bili sledeći nizovi reči:

- Danas je lep i
- Danas je lep i sunčan
- Danas je lep i sunčan dan
- je lep i sunčan
- je lep i sunčan dan
- lep i sunčan dan

Iz ovoga zaključujemo da rezultat predstavljaju svi podstringovi veličine četiri, pet i šest. Podešavanja su prikazana u listingu 1.

Naravno nije jedini cilj da izdelimo tekst na jednake delove i da se nadamo da će se baš ti identični isečci naći u drugim radovima. Naš cilj je da postignemo da svi radovi koji su pisani na sprskom jeziku, bilo da su napisani na ćirilici, latinici, bez dijakritika, u drugim padežima, licima, rodovima, budu prepoznati sa tim isečćima ako predstavljaju isti tekst.

To se postiže dodavanjem *plugin-a* za srpski jezik koji se bavi rešavanjem prethodno navedenih problema.

```
settingsBuilder =
    XContentFactory.jsonBuilder()

    .startObject()
      .startObject("analysis")
        .startObject("filter")
          .startObject(
                 "filter_shingle")
          .field("type","shingle")
          .field(
            "min_shingle_size", 4)
          .field(
            "max_shingle_size", 6)
          .field(
         "output_unigrams", false)
          .endObject()
        .endObject()
        .startObject("analyzer")
          .startObject(
          "ShingleAnalyzer")
          .field("type","custom")
          .field("tokenizer",
                  "standard")
          .array("filter",
                "lowercase",
                "filter_shingle")
          .endObject()
        .endObject()
      .endObject()
    .endObject();
```

Listing 1. *Podešavanja analizatora*

Definisan je parametar koji govori na koliko termova će se deliti rad prilikom kreiranja segmenata naučnog rada. Na ovaj način je postignuto otkrivanje plagijata koji su posledica dva ili više naučnih radova. Na primer naučni rad radX je nastao tako što je autor kopirao prvu trećinu iz rad1, drugu iz rad2 i treću iz rad3. Nakon što je rad podeljen na segmente, taj segment prolazi kroz definisani analizator i šalje se upit ka *Elasticsearch* serveru. Kao rezultat tog upita se dobija *searchHits* za svaki naučni rad. *SearchHits* je jedan realan broj koji govori o tome koliko je taj upit uspeo da pogodi taj rad.
U prevodu provere se svi radovi i za svaki rad se dobije jedan realan broj. Rezultati su vraćeni sortirano. Međutim taj broj ne znači ništa sam po sebi, pošto *searchHits* zavisi od broja termova u radu i od sadržaja samog rada.
Međutim mi možemo izračunati koliki bi mogao da bude maksimalan *searchHits*. To se uradi tako što se proveri koliki bi bio *searchHits* da se rad poredi sa samim sobom. Kada podelimo ta dva broja dobijamo procenat sličnosti ta dva naučna rada. Jako je važno napomenuti da u zavisnosti od pogođenog terma u radu se *searchHits* povećava.
Konfiguracije *shingle filter tokena* utiče na to da povećavanje *searchHits* broja bude veće ukoliko se u radu nađu jedni pored drugih nizovi termova kreiranih prilikom otpremanja rada, nego kad se pogodi pojedinačan term.

6. ZAKLJUČAK

U radu je predstavljen sistem za proveru plagijarizama. Nakon toga je objašnjeno šta to predstavlja plagijarizam i na koje načine se može detektovati. Zatim su opisane tehnologije koje su korišćene za implementaciju sistema. Nakon toga, predstavljen je kompletan opis sistema, kao i njegove komponente i moduli. Detaljno su opisani moduli, funkcionalnosti modula i komunikacija između istih. Pored toga je prikazan model klasa sistema koji je zadužen za fizičku reprezentaciju tabela u bazi podataka.

U nastavku je prikazana implementacija sistema. Obrazloženi su zanimljivi delovi koda u kojima je implementirana provera plagijarizma, odnosno oni delovi koda koji su zaduženi za pretprocesiranje delova teksta, pripremu naučnih radova za proveru na plagijarizam.

Cilj ovog rada je bio da pronađe radove koji dosta liče, imaju dosta napisanog sličnog teksta i da ih prikaže korisniku kako bi on mogao detaljnije da to pregleda i utvrdi da li su stvarno plagijati. Međutim i dalje postoje neki nedostaci. Na primer ukoliko bi dva naučna rada bila napisana na istu temu i poseduju isti fond reči, Sistem za detekciju plagijata bi rekao da su dosta slični čak iako ne predstavljaju plagijate. Tako da je to jedna od stvari koja bi mogla da bude unapređena. Druga stvar, kada Sistem za detekciju plagijata da rezultate on može samo da nabroji najsličnije naučne radove i da eventualno po delovima kaže koji su delovi najsličniji. To znači da bi jedan od pravaca nastavka razvoja ovog rada bio da se nakon liste najsličnijih radova koje vrati Sistem za detekciju plagijata, na svakom od njih označi koji je to deo tačno kopiran.

7. LITERATURA

[1] Zu Eissen, Sven Meyer, and Benno Stein. "Intrinsic plagiarism detection." In *European conference on information retrieval*, 2006.

[2] Zu Eissen, Sven Meyer, Benno Stein, and Marion Kulig. "Plagiarism detection without reference collections." In *Advances in data analysis*, 2007.

[3] Benno Stein and Sven Meyer zu Eißen. Near Similarity Search and Plagiarism Analysis. In Proc. 29th Annual Conference of the GfKl Springer, 2006.

[4] https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf (pristupljeno u avgustu 2020.)

[5] Java 8, *Java dokumentacija*, https://docs.oracle.com/javase/8/docs/ (pristupljeno u avgustu 2020.)

[6] Webb, Phillip, Dave Syer, Josh Long, Stéphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, and Sébastien Deleuze. "*Spring boot reference guide*." *Part IV. Spring Boot features* 24 (2013).

[7] Prasanna, Dhanji R. "Dependency injection." (2009).

[8] Irwin, John, Gregor Kickzales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, and J. Loingtier. "Aspect-oriented programming." *Proceedings of ECOOP, IEEE, Finland* (1997)

[9] Kononenko, Oleksii, Olga Baysal, Reid Holmes, and Michael W. Godfrey. "Mining modern repositories with elasticsearch." In *Proceedings of the 11th working conference on mining software repositories*, pp. 328-331. 2014.

[10] Angular dokumentacija, https://angular.io/docs (pristupljeno u avgustu 2020.)

[11] Wohlgethan, Eric. "Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue. js." PhD diss., Hochschule für Angewandte Wissenschaften Hamburg, 2018.

[12] Mikowski, Michael, and Josh Powell. *Single page web applications: JavaScript end-to-end*. Manning Publications Co., 2013.

[13] Shingle token filter, https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-shingle-tokenfilter.html (pristupljeno u avgustu 2020.)

Kratka biografija:

Stefan Bokić rođen je 02.10.1996. godine u Novom Sadu. Osnovnu školu „Žarko Zrenjanin“ završio je 2011. godine. Gimnaziju „Jovan Jovanović Zmaj“ u Novom Sadu završio je 2015. godine. Iste godine upisao se na Fakultet tehničkih nauka, odsek Računarstvo i automatika. Školske 2019. godine je diplomirao sa prosekom 9.97. Iste godine upisao je master studije, smer Računarstvo i automatika, modul Elektronsko poslovanje. Položio je sve ispite predviđene planom i programom.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.31
DOI: https://doi.org/10.24867/10BE17Spasojevic

ANALIZA PRORAČUNA OPTIMALNE TOPOLOGIJE PRENOSNIH MREŽA ZA POTREBE ELIMINACIJE PREOPTEREĆENJA

ANALYSIS OF OPTIMAL TOPOLOGY CALCULATION FOR RESOLVING OVERLOADS IN TRANSMISSION NETWORKS

Ana Spasojević, Savo Đukić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu su opisani osnovni pojmovi vezani za optimalnu promjenu topologije prenosnih mreža (OPTPM). Takođe, teorijski su opisana dva algoritama OPTPM-a. Oba algoritma su verifikovana nad testnom mrežom i izneti su zaključci o dobijenim rezultatima.*

Ključne riječi: *Optimalna promjena topologije, Preopterećenje elemenata, Prenosna mreža.*

Abstract – *The paper describes basic concepts about the optimal topology calculation for transmission networks. Also, two algorithms for optimal topology calculation for transmission networks are described theoretically. Both algorithms are verified using test network and obtained conclusions are provided.*

Keywords: *Optimal topology change, Element overload, Transmission network.*

1. UVOD

Nakon narušavanja ograničenja opterećenja elemenata, da bi se eletroenergetski sistem (EES) preveo u stabilan režim rada, operatori posežu za upravljačkim akcijama. Dostupne upravljačke akcije su promjena proizvodnje, rasterećenje potrošnje i akcije nad prenosnom mrežom [1].

Posmatranjem rada operatora primjećeno je da oni koriste rekonfiguraciju mreže za potrebe poboljšanja stanja u mreži. Na osnovu iskustva određuju da li promjena topologije može dovesti do poboljšanja stanja u mreži. Dakle, ukoliko se već promjena topologije aktivno koristi u svrhe razrješenja preopterećenja, postoji osnova za uvođenjem proračuna koji bi omogućili da se donesu bolje odluke prilikom upravljanja [2].

U drugom poglavlju ovog rada su predstavljena dva algoritma OPTPM-a. U trećem poglavlju su upoređeni rezultati oba algoritma nad testnom mrežom. Na kraju rada je dat zaključak, kao i korišćena literatura.

2. OPTPM

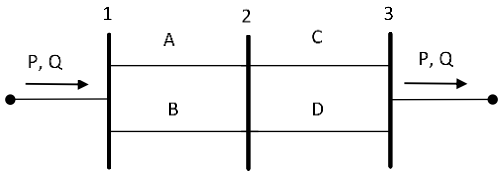
Proračun OPTPM-a predstavlja način da se promjenom uklopnog stanja elemenata mreže popravi stanje sistema.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Savo Đukić, docent.

Prilikom proračuna se mora voditi računa da, tokom izvršavanja proračunate sekvence manipulacija rasklopnom opremom, svako novo stanje mora biti stabilno i ne smije dovoditi do novih narušavanja sigurnosti sistema [3].

2.1. Uticaj izmjene topologije na tokove snaga

Analizira se isječak mreže prikazan na Slici 2.1.1. Isječak ove mreže se sastoji od četiri grane od kojih su grane označene sa D i C paralelne, kao i grane označene sa A i B. Navedene grane su povezane sabirnicama označenim brojevima 1, 2 i 3. Pretpostaviće se da je u grani C došlo do preopterećenja.



Slika. 2.1.1. – Dio EES-a

Isključenjem grane D bi dovelo do povećanja toka kroz granu C i dovelo bi do još većeg preopterećenja te grane jer bi ona morala prenositi sav tok od sabirnice 2 ka sabirnici 3. Međutim, ukoliko bi isključili granu B, povećala bi se impedansa između čvorova 1 i 2 i smanjila bi se vrijednost struje od čvora 1 do čvora 2.

Ovaj primjer je veoma jednostavan i nije teško zaključiti koje grane bi mogle biti isključene da bi došlo do rasterećenja opterećene grane. Međutim, EES su veoma komplikovani i u njima je teško na ovakav način zaključiti kojim granama je potrebno promjeniti uklopno stanje [4].

2.2. Proračuni OPTPM

Postoji više pristupa optimizaciji promjene topologije prenosne mreže. Oni se mogu klasifikovati po tome [3] [4]:

- koji elementi se uzimaju kao resursi (transformatori, sekcije, prekidači spojnog polja, otočni elementi poput kondenzatora i potrošača),
- da li su zasnovani na provjeri svih mogućih stanja, izmjeni matrice admitansi (impedansi) ili na simuliranju isključenja pomoću injektiranja struja (snaga), i

- da li kao resurse posmatraju elemente iz cijele mreže ili su resursi samo izabrani elementi.

U narednom poglavlju će biti predstavljena dva proračuna koja traže moguća razrješenja prekoračenja strujnih ograničenja, a kao resursi se koriste grane.

2.3. Proračun OPTPM zasnovan na tokovima snaga

Ovaj algoritam se zasniva na simulaciji promjene uklopnog stanja svakog resursa pojedinačno, a zatim se vrši upoređivanje broja prekoračenja nakon izmjene uklopnog stanja svakog od resursa.

Upoređivanjem broja prekoračenja za svaki od resursa, bira se onaj resurs koji će dovesti do najvećeg poboljšanja stanja. Kada se izabere, mreža sa promjenjenim stanjem izabranog resursa postaje nova početna mreža.

Za novu početnu mrežu se vrši simulacija promjene uklopnih stanja ostalih resursa i ponovo se bira resurs koji dovodi do najvećeg poboljšanja.

Ovaj proces se vrši iterativno sve dok se ne dođe do mreže bez prekoračenja ograničenja ili do stanja iz kojeg se promjenom uklopnog stanja elemenata ne može preći u stanje sa manjim brojem prekoračenja ograničenja.

Funkcija cilja je definisana sledećim izrazom:

$$\min\{\sum_{i=1}^{N}\left(\frac{I_i^{calc}}{I_i^{nom}}\right)^2+\sum_{j=1}^{M}\left(\frac{S_j^{calc}}{S_j^{nom}}\right)^2\} \qquad (2.3.1)$$

uz ograničenje:

$$Viol_p < Viol_k \qquad (2.3.2)$$

gdje su:

$$Viol_b=\sum_{l=1}^{L}Viol_{b,l}^{Sect}+\sum_{r=1}^{R}Viol_{b,r}^{Node}+\sum_{t=1}^{T}Viol_{b,t}^{GenQ}, b\in p,k. \qquad (2.3.3)$$

N- broj vodova,

I_i^{calc}- vrijednost modula struje voda *i*,

I_i^{nom}- vrijednost nominalne struje voda *i*,

M- broj transformatora,

S_j^{calc}- vrijednost modula prividne snage transfomatora *j*,

S_j^{nom} - vrijednost nominalne prividne snage transformatora *j*,

$Viol_p$- broj prekoračenja ograničenja u početnom stanju,

$Viol_k$- broj prekoračenja ograničenja u novom stanju,

L- broj svih grana,

p, k- indeksi početnog i novog stanja, respektivno,

$Viol_b$- broj prekoračenja ograničenja,

$Viol_{b,l}^{Sect}$- funkcija koja ima vrijednost 1 ukoliko je prekoračeno ograničenje grane *l*, a vrijednost 0 ukoliko nije,

R- broj svih čvorova,

$Viol_{b,r}^{Sect}$ - funkcija koja ima vrijednost 1 ukoliko su prekoračena ograničenja čvora *r*, a vrijednost 0 ukoliko nisu,

T- broj svih generatora, i

$Viol_{b,t}^{Sect}$ - funkcija koja ima vrijednost 1 ukoliko su prekoračena ograničenja reaktivne snage generatora *t*, a vrijednost 0 ukoliko nisu.

Prednost ovog proračuna predstavlja to što se može koristiti za proračun boljeg stanja i u mreži u kojoj ima veći broj prekoračenja. Mane su što optimum do kog ovaj algoritam dolazi ne mora biti globalni. Takođe, vrijeme izvršavanja je veliko jer se za svaki od resursa simulira promjena topologije i vrši proračun tokova snaga, i tako u svakoj iteraciji u kojoj se pronađe novo, bolje stanje mreže.

2.4. Proračun OPTPM zasnovan na strujnim distribucionim faktorima

Cilj ovog algoritma je razrješenje prekoračenja opterećenja jedne grane EES. Proračun koristi strujne distribucione faktore kako bi došao do informacije kakav će uticaj na stanje sistema imati izmjena topologije. Strujni distribucioni faktori prikazuju približne promjene vrijednosti struja po granama pri promjeni konfiguracije prenosne mreže [1].

Cilj proračuna je da se iz topologije A pređe u topologiju B koja se od A razlikuje samo za uklopno stanje jednog elementa. Prelaženjem iz jedne u drugu topologiju koja je najbolja od svih mogućih potencijalnih topologija se ide ka optimalnom rešenju.

U ovom proračunu se koristi „bazna" mreža. To je imaginarna mreža u kojoj su sve grane EES u uključenom stanju. Prije samog proračuna neophodno je proračunati matricu admitansi bazne mreže kao i matricu strujnih distribucionih faktora.

Početni korak algoritma predstavlja proračun tokova snaga kako bi se dobile injektirane struje i naponi svih čvorova. Nakon toga je neophodno proračunati injektirane struje nove mreže. S obzirom da će se vršiti samo jedna prekidačka akcija, uvodi se pretpostavka da će injektirane struje u obe mreže biti iste.

Iz matrice distribucionih faktora bazne mreže (koja je već proračunata) se proračunavaju matrice distribucionih faktora za sve topologije u koje se iz trenutne topologije mreže može preći jednom prekidačkom akcijom. S obzirom da je uvedena pretpostavka za vrijednost injektiranih struja nove topologije, mogu se izračunati struje elemenata nove topologije, kao i naponi čvorova i ispitati da li su ograničenja napona čvorova i struja elemenata ispoštovana za ponuđene topologije mreže. Topologija koja nudi najveća poboljšanja se proglašava novom početnom i proračun se nastavlja sve dok se ne dođe do topologije iz koje se ne može preći u bolje stanje.

S obzirom na uvedene pretpostavke, ne može se sa sigurnošću reći da su dobijeni rezultati tačni. Zato se moraju izvršiti i tokovi snaga za izabranu topologiju kako bi se potvrdilo da su rezultati dobri.

Ako rezultati nisu dobri, bira se druga topologija za koju se opet računaju tokovi snaga. Ako su rezultati dobri, ta topologija predstavlja početnu topologiju i algoritam se ponavlja. Nakon izvršenog algoritma postoji sekvenca prekidačkih akcija koja se može izvršiti da bi se prešlo u bolje stanje [3].

Funkcija cilja je definisana izrazom:

$$\min\{I_{opt}^{calc}\} \qquad (2.4.1)$$

uz ograničenje:

$$Viol_p < Viol_k \qquad (2.4.2)$$

gdje je I_{opt}^{calc} izračunata vrijednost struje preopterećene grane.

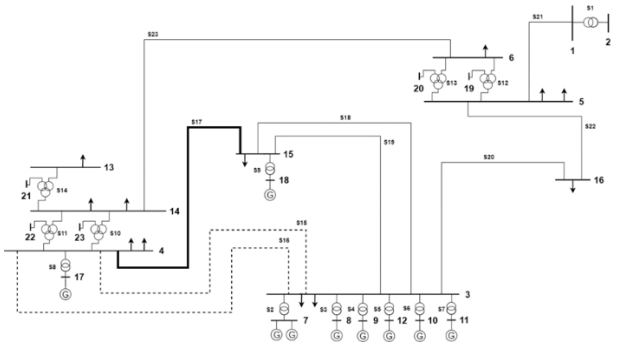
Opisani algoritam uzima u obzir ograničenja nejednakosti koja modeluju ograničenja vrijednosti napona čvorova i struja grana i vrši proračun na brz i sistematičan način. Međutim, potrebno je naglasiti da algoritam OPTPM zasnovan na distribucionim faktorima ne garantuje optimalnost rješenja. Takođe, veliki nedostatak je i to što se proračun vrši za razrješenje preopterećenja samo jedne grane, a ne za razrješenje preopterećenja svih grana u mreži.

3. VERIFIKACIJA OPISANIH ALGORITAMA

U ovom poglavlju prikazani su rezultati dobijeni izvršavanjem opisanih algoritama nad testnom mrežom (slika 3.1.1). Algoritam koji koristi strujne distribucione faktore je implementiran programskim jezikom C++ i poređen sa već implementiranim algoritmom zasnovanim na tokovima snaga. Proračun OPTPM zasnovan na tokovima snaga će u nastavku biti nazivan „kontrolni" proračun, a proračun OPTPM zasnovan na strujnim distribucionim faktorima će biti nazivan „implementirani" proračun. Razmatrana su dva primjera za istu testnu mrežu, ali se impedansa preopterećene grane razlikuje u prvom i drugom primjeru.

3.1. Primjer 1

Proračun je testiran nad mrežom koja se sastoji od 23 čvora, 23 grane, 9 generatora i 12 interkonekcija. Grane *S15* i *S16* su deenergizovane i prikazane su isprekidanim linijama na slici 3.1.1. Stanje u mreži je takvo da je grana *S17* preopterećena (prikazana podebljanom linijom na slici 3.1.1).



Slika 3.1.1. *Razmatrana test mreža*

Kada je izvršen implementirani proračun, nije predložena nijedna grana za promjenu uklopnog stanja. Proračunato novo stanje nakon promjene uklopnog stanje bilo kog od resursa bi, prema implementiranom proračunu, dovelo do prekoračenja naponskih ili strujnih ograničenja.

Kontrolni proračun je kao rezultat predložio promjenu uklopnog stanja grane S15. Dakle, rezultat implementiranog proračuna se ne slaže sa kontrolnim proračunom.

Nakon poređenja stvarnih vrijednosti napona čvorova i struja (prividnih snaga) grana, koje se dobijaju ako grana S15 promjeni uklopno stanje, i vrijednosti dobijenih od strane implementiranog proračuna, detektovane su značajne razlike između stvarnih i proračunatih napona.

Početna pretpostavka je bila da će injektirane struje ostati približno jednake nakon manje promjene topologije, kao što je izmjena uklopnog stanja jednog elementa. Da bi se utvrdio uticaj razlike injektiranih struja, izvršen je implementirani proračun sa ispravnim (proračunatim od strane tokova snaga) injektiranim strujama za slučaj kada je grana S15 energizovana i dobijeni su rezultati koji se zanemaraljivo razlikuju od tačnih vrijednosti. Dakle, zbog razlike u injektiranim strujama, implementirani proračun daje loše rezultate.

3.2. Primjer 2

Prilikom analize rezultata prethodnog primjera, primjećeno je da su gubici nad opterećenom granom značajni i da iznose 30% ukupnih gubitaka mreže. Nakon što se razriješi preopterećenje te grane, gubici po njoj se značajno smanje. Zbog toga, ukupna snaga sistema se smanji i injektirane struje u čvorovima se značajno promjene. Da bi se analizirao uticaj promjene snage gubitaka na rezultate proračuna, smanjena je impedansa opterećene grane, a samim tim i gubici po njoj. Sa smanjenom impedansom, gubici na preopterećenoj grani će u obe situacije biti zanemarljivi.

Proračun je testiran nad istom test mrežom kao u prethodnom primjeru. Kada je izvršen implementirani proračun, nije predložena nijedna grana za promjenu uklopnog stanja. Proračunato novo stanje nakon promjene uklopnog stanje bilo kog od resursa bi, prema implementiranom proračunu, dovelo do prekoračenja naponskih ili strujnih ograničenja.

Kontrolni proračun je kao rezultat predložio promjenu uklopnog stanja grane S15. Dakle, rezultat implementiranog proračuna se ponovo ne slaže sa kontrolnim proračunom.

Nakon poređenja stvarnih vrijednosti napona čvorova i struja (prividnih snaga) grana, koje se dobijaju ako grana S15 promjeni uklopno stanje, i vrijednosti dobijene od strane implementiranog proračuna, ponovu su detektovane značajne razlike između stvarnih i proračunatih napona.

Da bi se utvrdio uticaj razlike injektiranih struja, izvršen je implementirani proračun sa ispravnim (proračunatim od strane tokova snaga) injektiranim strujama za slučaj kada je grana S15 energizovana i dobijeni su rezultati koji se zanemaraljivo razlikuju od tačnih vrijednosti. Dakle, zbog razlike u injektiranim strujama, implementirani proračun je opet dao loše rezultate.

3.3. Upoređivanje rezultata primjera

U tabeli 3.3.1 su prikazani rezultati proračuna za Primjer 1 i Primjer 2. Sivom bojom su označeni rezultati primjera za koji su razlike između stvarnih i proračunatih vrijednosti manje.

Tabela 3.3.1. *Pregled rezultata razmatranih primjera*

	Primjer 1	Primjer 2
Najveća razlika injektiranih struja [r.j]	0.0174	0.0007
Prosječna razlika injektiranih struja [r.j]	0.0014	0.0001
Najveća razlika napona čvorova [r.j]	0.1389	0.1606
Prosječna razlika napona čvorova [r.j]	0.1323	0.1539
Najveća razlika struja (prividnih snaga) grana [r.j]	0.0682	0.0849
Prosječna razlika struja (prividnih snaga) grana [r.j]	0.0242	0.0216

Na osnovu rezultata prikazanih u tabeli 3.3.1 može se zaključiti da je razlika vrijednosti injektiranih struja najmanja kada su gubici na preopterećenoj grani mali. Na ovaj način drastična promjena opterećenja ove grane ne utiče na proizvodnju. Iako je očekivano da će rezultati za sve veličine biti precizniji ukoliko su razlike između injektiranih struja prije i nakon optimizacije manje, razmatrani primeri su pokazali da nije tako. Štaviše, rezultati za napone su lošiji. Može se zaključiti da čak i mala razlika u injektiranim strujama dovodi do velike razlike u rezultatima.

4. ZAKLJUČAK

Na kraju rada se može zaključiti da rezultati koji se dobijaju algoritmom koji koristi distribucione faktore nisu dovoljno dobri da bi se algoritam mogao koristiti za proračun promene topologije prenosne mreže sa aspekta eliminacije preopterećenja. U proračunu je napravljena pretpostavka da će injektirana struja u čvorovima ostati približno ista ukoliko se napravi mala izmjena u topologiji poput promjene uklopnog stanja jedne grane.

Nakon analize primjera se došlo do zaključka da je promjena injektirane struje nakon izmjene uklopnog stanja jedne grane dovoljno velika da izazove značajne razlike u rezultatima koje daje proračun OPTPM zasnovan na strujnim distribucionim faktorima i proračun tokova snaga za režim koji se dobija kada ta grana izmjeni uklopno stanje.

Takođe, na osnovu Primjera 2 se može zaključiti da čak i mala razlika vrijednosti injektirane struje dovodi do velike greške u proračunu vrijednosti napona čvorova. Iako su dobijeni rezultati za struje vodova i prividne snage transformatora dobri, vrijednosti napona čvorova se previše razlikuju, pa se na osnovu dobijenih rezultata ne može zaključiti da li će novo stanje biti bez prekoračenja ograničenja napona ili će dovesti do novih.

Zbog toga, zaključak je da upotreba ovog algoritma nije moguća u savremenim softverima za upravljanje prenosnom mrežom.

5. LITERATURA

[1] A. Sarić, M. Ćalović i P. Stefanov: Eksploatacija elektroenergetskih sistema u uslovima slobodnog tržišta, Tehnički fakultet, Čačak, 2005.

[2] A. A. Mazi, B. F. Wollenberg and M. H. Hesse: Corrective control of power system flows by line and bus-bar switching, *IEEE Transactions on Power Systems*, Volume 1, Issue 3, Aug. 1986.

[3] R. Bacher and H. Glavitsch: Network Topology Optimization with Security Contraints, *IEEE Transactions on Power Systems*, Volume 1, Issue 4, Nov. 1986.

[4] E. B. Makram, K. P. Thorton and H.E. Brown: Selection of lines to be swithed to eliminate overloaded lines using a Z-matrix method, *IEEE Transactions on Power Systems*, Volume 4, Issue 2, May 1989.

Kratka biografija:

Ana Spasojević rođena je u Bijeljini 1993. godine. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Elektroenergetski sistemi odbranila je 2017. godine.

Savo Đukić rođen je u Novom Sadu 1983. godine. Doktorsku disertaciju odbranio je 2014. godine na Fakultetu tehničkih nauka iz oblasti Elektroenergetski sistemi.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.31
DOI: https://doi.org/10.24867/10BE19Jokovic

RAZVOJ KONCEPTA VEKTORSKOG UPRAVLJANJA ASINHRONOM MAŠINOM U POGONU OPŠTE NAMENE

DEVELOPMENT OF CONCEPTUAL VECTOR CONTROL ALGORITHM FOR INDUCTION MACHINE DRIVE

Marko Joković, Darko Marčetić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu je prikazana univerzalna procedura za podešavanje vektorskog kontrolera u pogonu opšte namene sa asinhronom mašinom. Polazeći od matematičkog modela vektorskog kontrolera, definiše se skup koraka za sintezu softversko/hardverskih modula koje uključuje pomenuti kontroler za pravilnu orijentaciju polja. Eksperimentalna verifikacija metode je izvršena na pogonu opšte namene sa AM upravljanog putem mikroprocesorskog sistema baziranog na DSPIC30F4011.*

Ključne reči: *Vektorsko upravljanje, Asinhrona mašina, Regulacija*

Abstract *The paper presents the universal procedure for synthesis of vector controller within the IM drive. Starting from the mathematical model of vector controller, the guideline steps for setting the relevant modules of field oriented controller are given. Performance evaluation of microcontroller based IM drive is given on DSPIC30F4011 control platform.*

Keywords: *Vector control, induction machine, regulation*

1. UVOD

Visoke performanse upravljanja u naizmeničnim elektromotornim pogonima sa *AM* su moguće jedino upotrebom koncepta vektorske kontrole. Ovakav vid upravljanja se danas sve više primenjuje čak i u okviru pogona opšte namene koji su se tradicionalno bazirali na jednostavnijim upravljačkim algoritmima (skalarno upravljanje u otvorenoj sprezi, kontrola klizanja…). Tome značajno doprinosi razvoj energetske i upravljačke elektronike zasnovane na jeftinim i komercijalno dostupnim mikroprocesorima.

Pogoni opšte namene se proizvode u masovnim serijama i često imaju za zahtev smanjenje broja raspoloživih senzora. Time se umanjuje cena pogona koji postaje konkurentan na tržištu proizvoda. Stožiji zahtevi tržišta po pitanju performansi rada nameću potrebu za definisanjem jednostavne ali univerzalne procedure za kalibraciju kontrolera vektorskog upravljanja.

U ovom radu, jedna takva procedura je predstavljena i detaljno definisana.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Darko Marčetić, red. prof.

2. VEKTORSKI MODEL AM

Posmatra se matematički model AM u sinhronorotirajućem *dq* sistemu osa dat u [1]. Ukoliko se unesu sledeća ograničenja unutar kontrolera:

$$\psi_{rd} = |\boldsymbol{\psi_r}|, \qquad \psi_{rq} = 0 \tag{1}$$

dq sistem se orijentiše na fluks rotora i postiže se pozicioniranje na njegov ugao.

Dobija se redukovani vektorski model *AM* trećeg reda, koji prikazan u prostoru stanja ima oblik:

$$\frac{d}{dt}\begin{bmatrix} i_{sd} \\ i_{sq} \\ \psi_{rd} \end{bmatrix} = \begin{bmatrix} -\frac{R_\sigma}{L_\sigma} & \omega_{dq} & \frac{L_m}{T_r L_r L_\sigma} \\ -\omega_{dq} & -\frac{R_\sigma}{L_\sigma} & -\frac{\omega L_m}{L_r L_\sigma} \\ \frac{L_m}{T_r} & 0 & -\frac{1}{T_r} \end{bmatrix} \begin{bmatrix} i_{sd} \\ i_{sq} \\ \psi_{rd} \end{bmatrix} + \begin{bmatrix} \frac{1}{L_\sigma} & 0 \\ 0 & \frac{1}{L_\sigma} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_{sd} \\ u_{sq} \end{bmatrix} \tag{2}$$

$$(L_m/T_r) i_{sq} - (\omega_{dq} - \omega)\psi_{rd} = 0 \tag{3}$$

Orijentacijom polja po 1), dinamička jednačina za *q* komponentu fluksa rotora postaje algebarska odakle se uočava redukcija reda modela.

Posledica pravilne orijentacija polja jeste raspregnuta kontrola momentom m_{el} i fluksom *AM* $\psi_{rd} = \psi_r$ koji se računaju kao:

$$m_{el} = \frac{3}{2} P \frac{L_m}{L_r} \psi_{rd} i_{sq} = \frac{3}{2} P \frac{L_m^2}{L_r} i_{sd} i_{sq} \tag{4}$$

$$\frac{d}{dt}\psi_{rd} = -\frac{1}{T_r}\psi_{rd} + \frac{L_m}{T_r} i_{sd} \xrightarrow{d/dt=0} \psi_{rd} = L_m i_{sd} \tag{5}$$

Praktična realizacija orijentacije polja *FOC* (*eng. Field Oriented Controller*) se vrši indirektno računom klizanja na osnovu algebarske relacije 3):

$$\begin{aligned} \omega_k &= \frac{L_m}{T_r}\frac{i_{sq}}{\psi_{rd}} = \frac{1}{T_r}\frac{i_{sq}}{i_{sd}} \\ \omega_{dq} &= \omega + \omega_k \\ \theta_{dq} &= \int \omega_{dq} dt \end{aligned} \tag{6}$$

Za potrebe regulacije struje kao drugog preduslova za vektorsku kontrolu *AM*, strujne jednačine u (2) se prepisuju u pogodan oblik za željenu analizu:

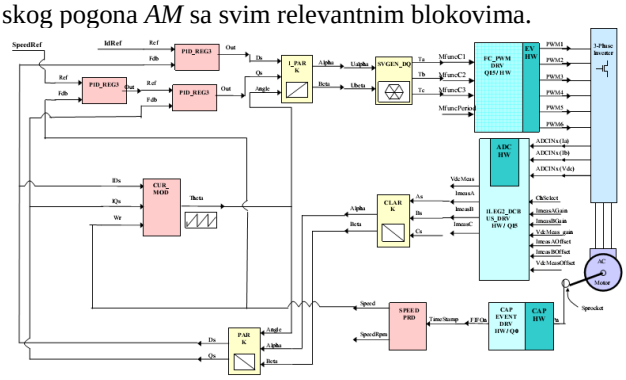
$$u_{sd} = R_s i_{sd} + L_\sigma \frac{di_{sd}}{dt} - \omega_{dq} L_\sigma i_{sq} \tag{7}$$

$$u_{sq} = R_s i_{sq} + L_\sigma \frac{di_{sq}}{dt} + \omega_{dq} L_\sigma i_{sd} + \omega_{dq} \frac{L_m}{L_r} \psi_{rd} \tag{8}$$

Relacije 1) –8) su polazna osnova za definisanje procedure za kalibrisanje vektorski upravljanog pogona *AM*.

3. PROCEDURA ZA PODEŠAVANJE VEKTORSKOG KONTROLERA AM

Na *Slici 1* prikazan je kompletan blok dijagram vektorskog pogona *AM* sa svim relevantnim blokovima.



Slika 1. *Kompletan blok dijagram vektorski upravljanog pogona AM*

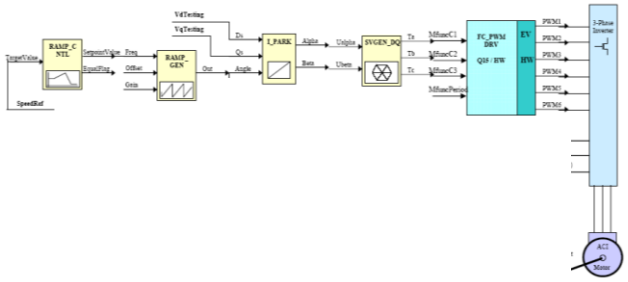
Testiranje vektorskog kontrolera *AM* po predloženoj proceduri se sprovodi kroz nekoliko faza:

- Test *VSI* – test naponskog izvora
- Test *CRVSI* – strujno regulisani naponski izvor, metoda za podešavanje pojačanja PI regulatora struje
- Test *FOC* – podešavanje orijentacije polja
- Test *Brzinska petlja* – metoda podešavanje pojačanja PI regulatora brzine

3.1. Test VSI

Prva faza testa vektorskog kontrolera *AM* u osnovu obuhvata testiranje mehanizma za pravljenje vektora napona unutar pogona sa invertorskim pretvaračem.

Na *Slici 2* prikazan je deo pogona kojim se ostvaruje prva faza testa – impulsno širinska modulacija putem prostornih vektora *SVPWM* (*eng. Space Vector Pulse Width Modulation*).



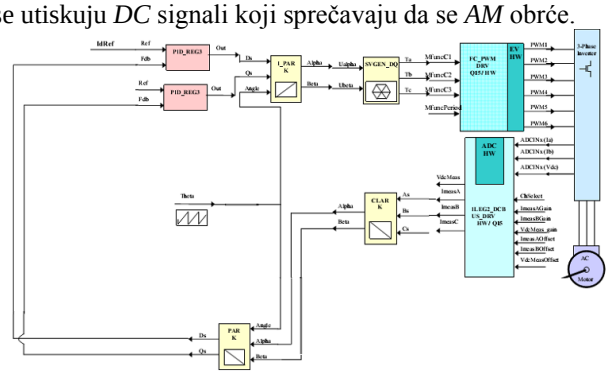
Slika 2. *Blok dijagram za testiranje faze I - VSI*

Blokovima *RAMP_CTRL* i *RAMP_GEN* se zadaju vrednosti učestanosti napajanja *Freq* i naponskih referenci *VdTesting* i *VqTesting*. Blokovima *I_PARK* i *SV_GEN* se procesuiraju ulazni signali i računaju vremena vođenja za *PWM* periferiju.

3.2. Test CRVSI

Vektorska kontrola je specijalan vid strujne kontrole kod koje se upravlja pored amplitude i faznim stavom struje u odnosu na referentni fluks rotora. Iz tog razloga jako je važno obezbediti strujnu regulaciju u pogonu u slučaju da se energija dobija iz naponskog *VSI* pretvarača.

Koncept strujno regulisanog naponskog izvora tzv. *CRVSI* se može obezbediti ukoliko dinamika uvedene strujne petlje u okviru kontrolnog algoritma značajno ubrza odziv struje u odnosu na prirodnu dinamiku uspostavljanja iste. Celokupna konfiguracija za testiranje *CRVSI* je prikazana na *Slici 3*. U odnosu na konfiguraciju sa *Slike 2* uvodi se povratna sprega po struji sa svim elementima za procesuiranje dok se signali napona dobijaju kao izlaz iz strujnih regulatora. Učestanost napajanja iznosi nula, čime se utiskuju *DC* signali koji sprečavaju da se *AM* obrće.



Slika 3 *Blok dijagram za testiranje faze II - CRVSI*

CRVSI posmatra napone kao upravljačke veličine kojima se potrebno upravljati na način da se utiskuje željeni vektor struje AM. U tom pravcu, koriste se relacije 7) i 8) koje opisuju zakonitosti uspostavljanja struje pri pobudnim naponskim signalima. Izvodi se funkcija prenosa od napona do struje sa zanemarenjem sporopromenljivih članova usled obrtnih elektromotornih sila kao i efekata sprezanja osa usled utiskivanja *DC* signala kada je $\omega_{dq} = 0$, [2]. Funkcije prenosa strujnog podsistema se opisuju filtarskom strukturom niskopropusnika opsega prvog reda:

$$i_{sd}(s) = \frac{1/R_s}{1 + T_\sigma s} u_{sd}(s) \qquad 9)$$

$$i_{sq}(s) = \frac{1/R_s}{1 + T_\sigma s} u_{sq}(s) \qquad 10)$$

Ukoliko se raspolaže informacijom o parametrima otpora namotaja statora R_s i ekvivalentne vremenske konstante namotaja mašine T_σ može se iskoristiti kao pogodan *Dahlin*-ov algoritam za sintezu parametara regulatora [3]. Algoritam uvažava u obzir diskretni rad kontrolera sa periodom T kao i kašnjenje pretvarača τ i posmatra funkciju prenosa objekta u vidu filtra prvog reda sa transportnim kašnjenjem:

$$G_{ob}(s) = \frac{K_{inv} \cdot 1/R_s}{1 + sT_\sigma} \cdot e^{-\tau s} \qquad 11)$$

gde je $K_{inv} = u_{DC}/\sqrt{3}$ pojačanje pretvarača.

Optimalni parametri po *Dahlin* algoritmu su:

$$K_p = \frac{1 - e^{-\lambda T}}{K \cdot \left(e^{\frac{T}{T_1}} - 1\right) \cdot [1 + N \cdot (1 - e^{-\lambda T})]} \qquad 12)$$

$$K_i = K_p \frac{T}{T_i} = K_p \left(e^{\frac{T}{T_1}} - 1\right) \qquad 13)$$

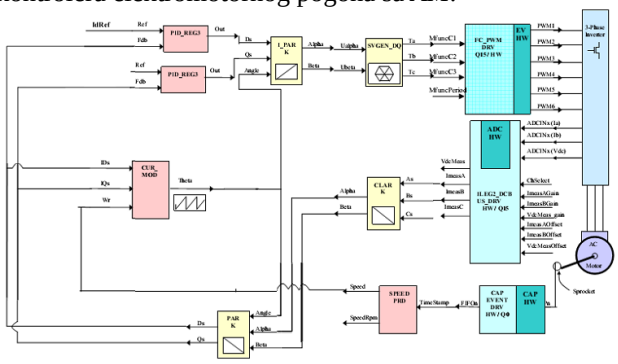
gde su redom:

- λ - parametar koji definiše širinu propusnog opsega strujne regulacione konture (brzinu uspostavljanja odziva struje),
- N - broj perioda transportnog kašnjenja,

3.3. Test FOC

Treća test faza vektorskog pogona *AM* podrazumeva kalibraciju bloka za orijentaciju polja dq koordinatnog

sistema na fluks rotora. Pokazano je da se orijentisanjem sistema na fluks rotora dobija raspregnuta kontrola fluksa i momenta u pogonu putem upravljačkih komponenti vektora struja statora. Uz prethodno testiran *CRVSI* ovim je omogućena minimalna konfiguracija vektorskog kontrolera elektromotornog pogona sa *AM*.



Slika 4 *Blok dijagram za testiranje faze III - FOC*

Test *FOC* pored *CRVSI* konfiguracije uključuje dodatni blok *CUR_MOD* koji je zanosnovan na 6). Na osnovu komponenti vektora struje statora i vremenske konstante rotora T_r blok daje na izlazu signale ω_{dq} i θ_{dq} koji se koriste u okviru blokova za transformacije koordinata *PARK* i *IPARK*.

Ukoliko se parametar inverzne vremenske konstante rotora u mikrokontroleru poklopi sa istim parametrom u stvarnom pogonu, postići će se 1) odnosno 4) i 5). Promena struje statora d ose će rezultovati samo promenom fluksa rotora u mašini, promena q komponente struje statora će direktno menjati momenat *AM*. Neophodno je napomenuti da je ovaj test potrebno raditi pri ukočenom rotoru, tj.u stanju kratkog spoja pogona.

Ovim se onemogućuje obrtanje rotora *AM* te ne postoji opasnost da se mašina usled značajnih vrednosti postignutom momenta pokrene. Treći ulaz u blok *CUR_MOD Speed* može se izjednačiti sa nulom u tom slučaju te je učestanost napajanja jednak učestanosti klizanja u pogonu.

Koraci za kalibraciju vremenske konstante rotora su redom nabrojani:

- Utiskivanje d komponente struje statora u *AM*
- Zatim se daje profil skokovite reference komponente struje po q osi u ekvidistantnim vremenskim intervalima
- Na osnovu podataka o vrednostima q komponente struje rotora, izračunatih odgovarajućih klizanja kao i momenata AM neophodno je sačiniti linearnu korelaciju
- Mera odstupanja od linearne korelacije (standardna devijacija, modus/medijana...) treba da obezbedi mehanizam za adaptaciju parametra inverzne vremenske konstante na način da se ponovnim ponavljanjem prethodnih koraka ista smanji
- U slučaju pronalaska kalibracionog parametra inverzne vremenske konstante rotora kojom je mera odstupanja linearne zavisnosti momenta od klizanja minimalna postupak se prekida a vrednost parametra se proglašava za referentnu

3.4. Test Brzinska petlja

Poslednji test vektorskog pogona *AM* upravljanog po brzini obrtanja rotora jeste zatvaranje povratne sprege po brzini. Blok dijagram pogona za ovaj test je već prikazan na *Slici 1*.

Signali na ulazu u regulatore struja *dq* osa se ne zadaju ručno već se dobijaju kao izlaz iz PI regulatora brzine.

Test brzinske petlje najpre podrazumeva sprovođenje metode za merenje mehaničke inercije J pogona. Metode zaustavljanja, zaleta sa konstantnim momentom m_e itd pri neopterećenom stanju u pogonu mogu da posluže u svrhe proračuna ovog parametra.

$$J = m_e / \frac{d\omega}{dt} \approx m_e / \frac{\Delta\omega}{\Delta t} \qquad 14)$$

Nakon merenja momenta inercije moguće je nekom od analitičkih metoda za proračun parametara regulatora brzine dobiti vrednosti proporcionalnog i integralnog pojačanja kojim se obezbeđuju željeni zahtevi pogona.

Koristi se procedura za pronalaženje parametara po kriterijumu minimizacije sume greške [4]:

$$K_p = 0.2027 \frac{2J}{T_\omega} \qquad 15)$$

$$K_i = 0.03512 \frac{2J}{T_\omega} \qquad 16)$$

gde je T_ω definisana perioda rada brzinske petlje.

Ovom metodom se obezbeđuje kritično aperiodičan odziv na odskočnu pobudu. Metoda uvažava u obzir diskretan rad regulatora brzine a kao ulazni podatak koristi određeni parametar J po 14).

4. EKSPERIMENTALNA VERIFIKACIJA REZULTATA PREDLOŽENE PROCEDURE

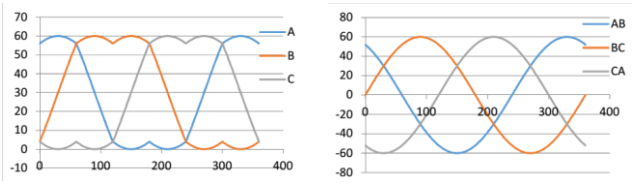
Prototip pogona se sastoji od dve identične *AM*, mehanički povezane u motor-generator grupu za svrhu opterećivanja u pogonu. Energetska elektronika se sastoji od trofaznog invertora koji pretvara DC u AC pomoću PWM upravljačke strategije. Frekvencija PWM nosioca je podesno izabrana na vrednost $f_{PWM} = 15151\,Hz$. Digitalna kontrola koja uključuje vektorski kontroler, sa opisanom procedurom je izvedena putem digitalnog mikrokontrolera zasnovanog na *dsPIC30F4011*. Ovaj mikroprocesor zadovoljava sve kriterijume koji se postavljaju pred jedan moderni pogon visokih performansi za primenu u oblasti električne vuče i transporta. Ova postavka je povezana na DC napajanje koje se sastoji od četiri 12V baterije povezane na red kako bi formirale stabilan 48V izvor napajanja za pretvarač. Postavka je prikazana na *Slici 5*.

Slika 5 *Postavka pogona za eksperiment*

Električni parametri korišćene *AM* su: $R_s = 5.5\,m\Omega$, $L_{\gamma s} = L_{\gamma r} = 28\,\mu H$, $T_r = 255\,ms$, $R_{Fe} = 3.2\,\Omega$

Observacijom signala vremena vođenja *A*, *B* i *C* kao izlaza iz *SVPWM* bloka u funkciji ugla modulacije i međufaznih signala na *Slici 6* ze test *VSI* može se zaključiti:

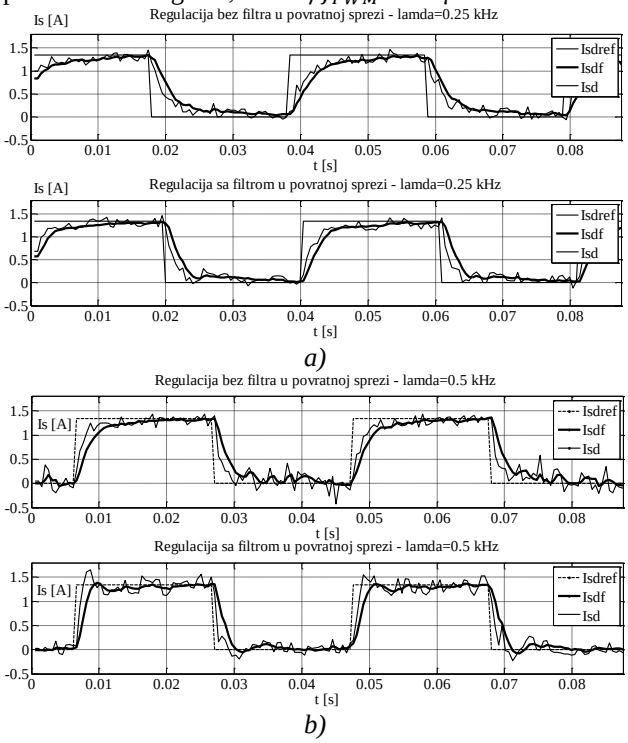
- (Analogni) Signali vremena vođenja faza pretvarača iz SVPWM predstavljaju *offset*-ovane kvazi sinusoidalne signale
- Treći harmonik signala koji je nultog redosleda simetrije je prisutan jedino u faznom signalu, međufazni signal ima samo osnovni harmonik
- *Offset* signal nije prisutan u međufaznim naponima pretvarača



a) *b)*

Slika 6. *Izgled a) SVPWM vremena vođenja faza invertora i b) međufaznih napona pretvarača*

Na *Slici 7* prikazani su eksperimentalni odzivi struja na odskočni profil reference po *Dahlin*-ovom kriterijumu za više vrednosti propusnog opsega λ. Uočava se aperiodičan odziv struje koji je u saglasnosti sa *Dahlin* algoritmom. Odziv regulacione strukture je brži od prirodnog odziva *AM* definisanog sa prirodnom učestanošću $1/T_\sigma = 100\,Hz$. Perioda rada strujne petlje odgovara periodi *PWM* signala, $T = 1/f_{PWM} = 66\mu s$.



a)

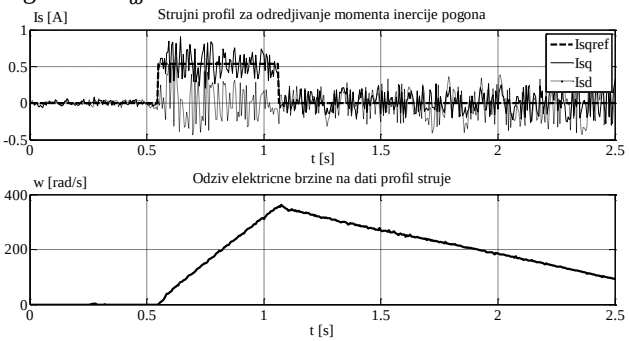
b)

Slika 7. *Eksperimentalni odzivi struja na odskočni profil reference po Dahlin-ovom kriterijumu a) pri $\lambda = 250\,Hz$ i b) $\lambda = 500\,Hz$ korišćenog AM*

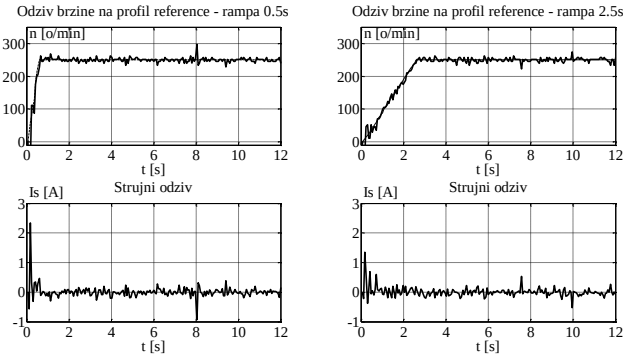
Na *Slici 8* prikazan je eksperimentalni test za određivanje momenta inercije pogona. Zadaje se profil q struje odskočni profil struje trajanja $\Delta t = 0.5s$ kojim se uspostavlja elektromagnetni momenat. Brzina pri konstantnom momentu se promeni za iznos $\Delta\omega = 350\,rad/s$, što daje vrednost za momenat inercije:

$$J = \frac{m_e \Delta t}{\Delta\omega} = 0.0041 kgm^2 \qquad 16)$$

Na kraju, prikazani su i eksperimentalni rezultati odziva brzine (gornji) i struje (donji grafici) pogona *AM* na profil brzine, referenca brzine *250 o/min* tipa rampa funkcije trajanja 0.5s, desno, i 2.5s, levo. Sinteza parametara je izvršena na osnovu 15)–16) sa periodom rada brzinskog regulatora $T_\omega = 20\,ms$.



Slika 8. *Momentni profil za određivanje parametra J*



Slika 9. *Odzivi brzine (gornji) i struje (donji grafici) pogona na profil brzine, referenca brzine 250 o/min tipa rampa funkcije trajanja 0.5s, desno, i 2.5s, levo*

5. ZAKLJUČAK

U ovom radu je prikazana praktična procedura za kalibraciju vektorskog kontrolera pogona *AM* opšte namene. Dat je detaljan opis svih relevantnih koraka sinteze blokova koji sačinjavaju ovaj kontroler. Na primeru digitalnog pogona *AM*, prikazani su eksperimentalni rezultati testova koji potvrdjuju validnost predložene praktične procedure.

6. LITERATURA

[1] Darko Marčetić: „*Mikroprocesorsko upravljanje energetskim pretvaračima*“, Fakultet tehničkih nauka, Novi Sad, 2014.

[2] V. Vučković, „*Električni pogoni*“, Akademska misao, 2002.

[3] Dahlin E.B. „*Designing and tuning digital controllers, Part I and II*“, Instrum. Control. Systms, vol. 41, pp. 77-83 and pp. 87-91, Jun./Jul. 1968.

[4] Slobodan Vukosavić: „*Digitalno upravljanje električnim pogonima*“, Akademska misao, Beograd, 2003.

Kratka biografija:

Marko Joković rođen je u Kladovu 1996. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Energetska elektronika i električne mašine odbranio je 2020.god.

Darko Marčetić je redovni profesor na Fakultetu tehničkih nauka u Novom Sadu, na Katedri za Energetsku elektroniku i pretvarače. Oblast interesovanja su mu digitalno upravljanje elektromotornim pogonima i optimalna kontrola.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE20Orcic

UPOTREBA MAŠINSKOG UČENJA ZA OBUČAVANJE ROBOTA ZA ŠPRICANJE JAGODA U UNITY 3D OKRUŽENJU

USE OF MACHINE LEARNING FOR TRAINING A STRAWBERRY SPRAYING ROBOT IN A UNITY 3D ENVIRONMENT

Timotej Orčić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Cilj ovog istraživanja jeste prezentovanje idejnog rešenja rapidnog razvoja veštačko-inteligentnih agenata putem simuliranja u Unity grafičkom okruženju. Oblast primene konkretno razvijanog agenta jeste poljoprivreda. Istreniran je model za detekciju objekata sa slike, kao i Reinforcement Learning model za ciljano kretanje agenta u okruženju. Podaci i okruženje su samostalno kreirani. Svi upotrebljeni modeli i algoritmi su evaluirani.*

Ključne reči: *veštačka inteligencija, veštačke neuronske mreže, prepoznavanje objekata sa slike, Reinforcement Learning, Unity 3D, Unity ML Agents*

Abstract – *The aim of this research is to present a conceptual solution for rapid development of artificially-intelligent agents through simulation in a Unity graphical environment. The area of application of the specifically developed agent is agriculture. An object detection model was trained, as well as a Reinforcement Learning model for targeted agent movement in the environment. Data and the environment are self-created. All models and algorithms used were evaluated.*

Keywords: *artificial intelligence, artificial neural networks, object detection, Reinforcement Learning, Unity 3D, Unity ML Agents*

1. UVOD

Naučno-tehnološka ekspanzija u prethodnim godinama i decenijama otvorila je prostor za automatizaciju velikog broja ljudskih delatnosti, pa tako i poljoprivrednih. San o autonomnim poljoprivrednim sistemima, koji zahtevaju minimalnu ljudsku interakciju sve nam je bliži zahvaljujući enormnom razvoju robotike i kompjuterske inteligencije.

Tema ovog rada je simulacija robota koji šprica jagode (konkretno zrele) obučenog putem *Reinforcement Learning*-a, a data simulacija se odvija koristeći *Unity 3D* grafički *engine* [1]. *Reinforcement Learning* je način obučavanja veštačke inteligencije da izvršava određeni zadatak putem nagrađivanja uspešnih (željenih) akcija i kažnjavanjem neuspešnih, tj. neželjenih akcija veštačko-inteligentnog agenta.

NAPOMENA:

Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Kovačević, van. prof.

Na taj način agent, u našem slučaju robot, može relativno brzo da savlada zadate izazove i "nauči" da izvršava željeni zadatak. Simulacija u grafičkom *engine*-u veoma je pogodna jer isključuje kreiranje stvarnog robota i sve izazove datog postupka, a takođe donosi ogromnu uštedu na vremenu obučavanja veštačke inteligencije jer omogućava paralelizovano treniranje. Odabran je *Unity 3D engine* jer je izuzetno pogodan za brz razvoj kvalitetnih 3D okruženja, a uz to poseduje eksterni skup alata - *Unity ML Agents Toolkit* [2] koji je kreiran baš za namene obučavanja veštačko-inteligentnih agenata u 3D grafičkom svetu.

Kreirana je 3D scena u *Unity*-ju sa tri reda jagoda kroz koje će robot kasnije prolaziti. Nakon toga je napravljen skup od 64 slike datih jagoda koji je podeljen na trening (58 slika) i test skup (8 slika). Jagode na datim slikama su zatim anotirane koristeći *labelImg* kompjuterski program, te je na kraju anotirano ukupno 264 jagode, od kojih je 232 pripadalo trening skupu a 32 test skupu. Zatim je, pomoću datog anotiranog skupa odrađen *fine-tuning* nad *ssd_mobilenet_v1_coco* konvolutivnoj neuronskoj mreži koristeći *Google*-ov *Object Detection API* [3], gde je uvedena klasa *strawberry* (tj. jagoda), kako bi jagode na našoj sceni mogle biti prepoznate od strane robota. Trening je pokrenut na *Google Colab* [4] platformi radi brzine (*GPU*) i trajao je 200,000 koraka. Robotu su postavljene dve kamere, jedna sa desne i jedna sa leve strane, čije su slike direktan ulaz u našu *fine-tune*-ovanu neuronsku mrežu, koje vraćaju poziciju detektovanih jagoda sa datih slika (ukoliko postoje), a nad datim izlazom se još sitnim matematičkim transformacijama zaključuje da li robot treba da pali prskalicu sa te strane ili ne, te mu se to prosleđuje kao krajnja informacija (kroz *Unity* okruženje). Kretanje robota je sprovedeno kroz sistem odvojen od prskanja, gde je u *Unity*-ju, koristeći pogodnosti *ML Agents* paketa isprogramirana cela *RL* logika našeg agenta uz pomoć tri *C#* programske skripte. Nakon još nekih sitnih podešavanja datog *Unity* dela celo okruženje je eksportovano kao aplikacija. Tako eksportovano okruženje učitano je unutar *python* skripte, unutar koje je vršeno obučavanje veštačko-inteligentnog *RL* modela za upravljanje kretanja robota. Obučavanje datog modela vršeno je korištenjem *Asynchronous Advantage Actor Critic RL* algoritma [5], koji omogućava treniranje nad više instanci datog 3D okruženja. Dato obučavanje vršeno je na privatnom računaru (*CPU* sa 4 jezgra) te je učeno na 4 instance okruženja u paraleli. Nakon desetak dana (oko 250 sati) algoritam je načinio 1500 koraka i pokazao dobre rezultate - konkretno agent je naučio da prati prvi red jagoda, da napravi "polukružno skretanje" i da se uspešno kreće između prvog i drugog

reda jagoda, te je obučavanje obustavljeno a model sačuvan kao *Tensorflow* [6] zamrznuti graf (eng. *frozen graph*).
Ovo rešenje može predstavljati osnovu za razvoj stvarnog poljoprivrednog robota koji šprica različite tipove voća ili povrća određenim supstancama, te umnogome olakšati dati deo procesa uzgajanja, a uz dodatni trud može predstavljati osnov za razvoj raznih drugih tipova poljoprivrednih robota, posebno koristeći modul za kretanje.

2. METODOLOGIJA

U ovom poglavlju prezentovane su primenjene metodologije za razvoj pomenutog veštačko-inteligentnog agenta. Dato poglavlje podeljeno je u nekoliko faza, koje prate sam tok razvoja projekta.

2.1. Kreiranje Unity ML okruženja

S obzirom da u *RL* okruženje predstavlja okosnicu dešavanja, na početku je napravljeno *Unity* okruženje, koje sadrži jednu scenu i u koje je uključen *ML Agents* paket. Na datu scenu je ubačen teren (podloga sa teksturom zemlje), osvetljenje i 3D model našeg robota (*RL* agenta). Datom robotu je dodato sve što je neophodno za njegovo kretanje po sceni i njegovu "fiziku".
S obirom da se projekat sastoji iz dva dela (prepoznavanje objekata i *RL*) na taj način je razvijana i *Unity* scena, te je na početku ručno postavljeno par redova 3D modela jagoda i jedna kamera sa desne strane robota, koja je korištena za pravljenje skupa slika opisanog u sledećem delu poglavlja.

2.2. Kreiranje i anotiranje skupa slika jagoda

Napravljeno je 64 slike ručno postavljenih jagoda u našoj sceni, koje će kasnije biti iskorištene za obučavanje modela za detekciju objekata. Treba napomenuti da su date slike jagoda u punoj rezoluciji, a da se robotska kamera služi smanjenom rezolucijom (300X300) radi brzine upotrebe algoritma za detekciju objekata (u realnom vremenu). Anotiranje jagoda na slikama obavljeno je korištenjem *labelImg* alata i anotirano je ukupno 264 jagode (ploda). Nakon toga, dati skup podataka je podeljen na trening (58 slika i 232 jagode) i test skup (8 slika i 32 jagode) i od datih slika i anotacija kreirana su 2 *csv* dokumenta (*training.csv* i *test.csv*) korištenjem male *python* skripte. Zatim su, od datih csv dokumenata kreirana 2 *tfrecord* dokumenta (*train.record* i *test.record*) korištenjem druge *python* skripte, te je ovime završen postupak kreiranja skupa podataka za obučavanje modela detekcije objekata.

2.3. Finetuning object detection modela

Da bismo na najbrži mogući način mogli da naučimo neku mrežu da prepoznaje jagode iz našeg skupa podataka, poslužili smo se *finetuning*-om [7]. Za dati zadatak smo odabrali *ssd_mobilenet_v1_coco* neuronsku mrežu koja je prethodno istrenirana na *coco* skupu podataka. *Coco* [8] skup podataka sadrži 80 klasa, a takođe i nekoliko vrsta voća, što pogoduje našem zadatku. *Finetuning* postupak koji smo uradili jeste obučavanje date mreže da detektuje novu klasu *strawberry* i na taj način omogući prepoznavanje jagoda iz našeg 3D okruženja. Za izvršavanje datog zadatka poslužili smo se *Google*-ovim *API*-jem za treniranje i rad sa *object detection* modelima - *Object Detection API*, koji je razvijen nad *Tensorflow* bibliotekom. Da bismo dobili na brzini i računarskoj snazi (*GPU*), dati zadatak smo izvršili na *Google Colab* platformi, pokrećući par jednostavnih skripti. Da bi to bilo omogućeno bilo je neophodno kopirati *Object Detection API* paket na *Google Drive*, te zatim kroz jednu jednostavnu *Colab* skriptu aktivirati dati paket. Zatim smo na dati *Google Drive* preuzeli istrenirani *SSD* model i raspakovali ga, putem druge jednostavne *Colab* skripte. Na kraju smo, pokretanjem poslednje *Colab* skripte inicirali *finetuning* datog modela za 200,000 koraka koje je teklo veoma brzo (oko 5 sati) jer je odabrani model veoma "lagan" a naš skup podataka nije velik. Na kraju je bilo potrebno eksportovati graf zaključivanja (eng. *inference graph*) kako bi model mogao da se stavi u upotrebu.

2.4 Nameštanje Unity ML okruženja za RL

U prethodnom delu datog poglavlja dali smo kratak uvod u izgled *Unity* okruženja koje smo razvijali, tj. prvog dela razvoja datog okruženja potrebnog za prvi deo projekta. Nakon realizacije prvog dela projekta preostao nam je komplikovaniji deo, *Reinforcement Learning*, tj. obučavanje našeg robota da se kreće u datom okruženju (uz određena pravila) i da postigne određeni cilj.
Za početak je bilo bitno podesiti *Unity* okruženje da bude pogodno za *RL*. To je postignuto korištenjem *Unity ML Agents* paketa, koji pruža svu potrebnu podršku za realizaciju *RL* okruženja i programiranje istih putem *C#* skripti (koje su standardni deo *Unity engine*-a). S obzirom da ključni delovi *Unity ML Agents* implementacije predstavljaju 3 komponente - *Academy*, *Agent* i *Brain*, rasparčaćemo razvoj ovog dela projekta na te 3 komponente. Uz te 3 osnovne komponente, bitno je napomenuti na koji način je razvijeno samo okruženje, tj. na koji način su programski postavljeni određeni elementi na scenu datog okruženja.

2.4.1 RobotArea (oblast u kojoj će se robot kretati)

U prethodnom delu o *Unity* implementaciji spomenuto je da je na scenu našeg okruženja postavljen teren, osvetljenje i 3D model robota. Kako bi postavka terena mogla više puta da se iskoristi dati teren i robot su spakovani u jedan *Unity*-jev *GameObject*, tj. u zajedničkog roditelja, kako bi mogli zajedno da se koriste i pomeraju, te je od datog *GameObject*-a napravljen takozvani *Prefab*. Osim navedenog, napisana je jedna *C#* skripta (*RobotArea.cs*) u kojoj je implementirana sva programska logika potrebna za generisanje određenih objekata na sceni i koja će kasnije poslužiti ostalim delovima *ML Agents* arhitekture da programski pristupaju datim generisanim objektima.

2.4.2 RobotAcademy (Academy - glavna ML Agents komponenta)

Napravljen je *Prefab* za akademiju našeg projekta pomoću koje se kontrolišu osnovna podešavanja našeg okruženja i za koju se vezuju *Brain* elementi *Unity ML*-a. Osim toga, napisana je i kratka *C#* skripta koja kontroliše šta se dešava prilikom resetovanja akademije, a u našem slučaju to će samo inicirati resetovanje *RobotArea* instanci koje postoje u našem okruženju (konkretno jedna).

2.4.3 RobotAgent (RL Agent)

Napravljen je *Prefab* za našeg agenta koji sadrži 3D objekat robota i kamere vezane za njega. Dati *Prefab* ima dodat *rigidbody* element koji mu dodaje potrebnu "fiziku" i naravno *C#* skriptu kojom se kontroliše njegova logika

kretanja, a u koju je ugrađena sva *RL* logika pomoću koje će dati agent naučiti da se pravilno kreće u datom okruženju. Na datu *C#* skriptu se kroz *Unity editor* vezuju konkretne instance kamera, a i *Brain* objekat, koji će u stvari upravljati datim agentom.
Akcije koje naš *RL* agent može da preuzme prilikom kretanja su sledeće:

1) Ne radi ništa (ne preduzimaj nikakvu akciju)
2) Skreći levo
3) Skreći desno

Nagrade koje okruženje vraća agentu u odnosu na njegove akcije su sledeće:

1) Kreće se po dobrom putu spram jagoda **+0.15**
2) Kreće se previše blizu/daleko od jagoda **-0.2**
3) Kreće se po dobrom putu spram jagoda u ćošku (prilikom skretanja) **+0.3**
4) Kreće se previše daleko od jagoda u ćošku **-0.4**
5) Kreće se previše blizu jagoda u ćošku **-0.2**
6) Gazi preko jagoda **-5**
7) Ušao je u pogrešan region **-50**
8) Otišao je predaleko od područja s jagodama **-50**
9) Isteklo mu je zadato vreme **-50**
10) Stigao je na cilj **+100**

Dato *RL* okruženje se resetuje u sledećim slučajevima:

1) Agent je ušao u pogrešan region
2) Istekao je zadati vremenski period
3) Agent je otišao predaleko od polja jagoda
4) Agent je stigao na cilj (u toku zadatog vremena)

2.4.4 Brain

Brain (mozak) predstavlja jedinicu kojom se upravlja agentima na koje je povezan. U našem projektu implementirane su 2 vrste mozga za našeg agenta: *RobotPlayeBrain* i *RobotLearningBrain*. Prvi mozak je služio da se postave i testiraju pravila u okruženju i testiraju agentske akcije uz pomoć tastera na tastaturi (poput igrice), dok je drugi mozak kreiran da vrši komunikaciju između *Unity* okruženja i *python RL* koda kroz koji će se naš agent obučavati da uspešno reši zadati zadatak.

Dati agentski mozak nema nikakve vektorske obzervacije, već ima nekoliko **vizuelnih obzervacija**, konkretno, prednju kameru dimenzija 160x120 piksela, crno-belu, koja će se slati na ulaz *python RL* modela i dve kamere sa strane dimenzija 300x300 piksela u boji, pomoću kojih će se aktivirati prskalice prilikom detekcije jagoda. Kao odgovor na dati ulaz prednje kamere, *python RL* model će vraćati vektor dimenzija 1x1 (sa 3 opcije), koji će predstavljati koja je sledeća **akcija** koju agent treba da izvrši u *Unity* okruženju. Dakle, isto ono što smo pomoću *PlayerBrain*-a radili tastaturom, sada će *RL* model učiti da radi sam.

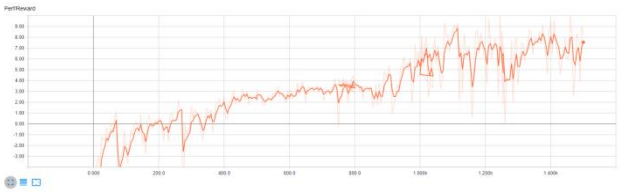
Na kraju je bitno napomenuti da je od datog *Unity* okruženja bilo potrebno napraviti (eng. *build*) program koji može da se izvršava na određenom operativnom sistemu (.exe, .x86_64, ...) kako bi dato okruženje moglo da bude pokrenuto putem *python* koda, te da se izvrši prethodno pomenuto obučavanje *RL* modela.

2.5 Obučavanje RL modela - kretanje robota

Naš agent je "učio" da se kreće koristeći jednu *python* skriptu (*A3C_CPU.py*) u kojoj je kreirana globalna mreža (model) i 4 radnika (eng. *worker*) pomoću kojih se instanciralo naše *Unity* okruženje u 4 paralelne instance (broj *CPU* jezgara privatnog računara).

Za svakog od instanciranog radnika kreirana je lokalna mreža nad kojom se agent obučavao i slao naučene težine u globalnu mrežu, te je tako postignuto deljenje naučenog znanja. Obučavanje je vršeno 1500 koraka i trajalo je oko 10 dana, za kojih je naš agent naučio da solidno prati liniju jagoda i da u većini slučajeva napravi "polukružno skretanje" i isprati drugi red jagoda do kraja.

Na slici 1 prikazan je graf napretka vrednosti nagrade (eng. *reward*) koje je jedan od radnika ostvario u toku obučavanja.



Slika 1. *Graf napretka vrednosti nagrade A3C radnika*

Nakon obučavanja bilo je potrebno napraviti "zamrznuti graf" od obučenog modela, a to je postignuto učitavanjem celokupnog modela (globalni i 4 lokalna), te izdvajanjem i čuvanjem samo globalnog modela putem *tensorflow variable scope*-a, i na kraju "zamrzavanjem" tako izdvojenog globalnog modela, nakon ponovnog učitavanja.

2.6 Povezivanje RL i OD modela u konačan sistem

Nakon obučavanja agenta da se kreće putem *RL* metode, ostalo je samo da se poveže *OD* model za detekciju jagoda sa agentom i na taj način napravi podloga za aktiviranje prskalica sa leve i desne strane robota. U *Unity*-ju su na našeg agenta ugrađene prskalice putem *ParticleSystem GameObject*-a, pomoću kojih je vizualizovana animacija prskanja jagoda. Kao što smo već videli u prethodnom delu datog poglavlja, na naš *LearningBrain* osim prednje kamere robota spojene su i leva i desna kamera.

Kao odgovor na ulaze sa datih kamera *python* skripta za simuliranje našeg agenta vraćaće vektor dimenzija 1x1 (sa 2 opcije) koji će signalizirati da li treba paliti ili gasiti povezanu prskalicu. Data logika je programski povezana kroz *RobotAgent.cs* skriptu.

Krajnje zaključivanje i simulacija sprovedeni su u python skripti *Full_Inference.py*, gde su učitana oba zamrznuta *tensorflow* grafa (*RL* i *OD*), te se za svaki frejm iz datog *Unity* okruženja (prednje kamere) izvršavala *RL* predikcija za upravljanje kretanja robota, a za svaki šesti frejm pozivala detekcija jagoda za levu i desnu kameru, gde se proveravalo da li su detektovane jagode u prvoj trećini ulazne slike i da li su detektovane sa preciznošću većom od 60%.

Ukoliko je dati kriterijum ispunjen, vraćen je signal *Unity* okruženju da se upali prskalica sa odgovarajuće strane.

3. EKSPERIMENTALNA EVALUACIJA I REZULTATI

Izvršena je eksperimentalna evaluacija radi određivanja performansi korištenih algoritama za detekciju jagoda i kretanje robota u prostoru. Performanse algoritma za detekciju jagoda evaluirane su upotrebom test skupa, dok su performanse *A3C* algoritma evaluirane empirijski, tj. posmatranjem ponašanja obučenog *RL* modela. Metrike performansi algoritama za detekciju jagoda koje su korišćene u ovom radu su preciznost, odziv i F_1-mera.

U tabeli 1 prikazani su rezultati *SSD Object Detection* modela pomoću kojeg smo detektovali jagode za različite *IoU* i različite veličine oblasti detekcije.

Tabela 1. *Rezultati OD modela*

IoU/oblasti	Preciznost	Odziv	F1-mera
0.50:0.95/sve	0.745	0.200	0.315
0.50/sve	1.000	0.775	0.873
0.75/sve	1.000	0.775	0.873
0.50:0.95/male	-1.000	-1.000	-1.000
0.50:0.95/srednje	0.722	0.767	0.744
0.50:0.95/velike	0.800	0.800	0.800

Iz tabele se vidi da jedino sitne jagode ne bivaju detektovane, tj. sve metrike za male oblasti imaju vrednost -1.000, dok za srednje i velike oblasti detekcija biva poprilično uspešna.

Evaluacija *A3C RL* algoritma izvršena je empirijski, posmatranjem kretanja koje je robot naučio da savlada. Kao što je pomenuto u prethodnim poglavljima, za 1500 koraka naš robot je naučio da prati prvi red jagoda, da napravi "polukružno skretanje" i da se uspešno kreće između prvog i drugog reda jagoda.

4. ZAKLJUČAK

U ovom radu predstavljeno je jedno rešenje za razvoj veštačko-inteligentnog robota za prskanje jagoda, koje je postignuto simuliranjem u *Unity 3D engine*-u. Kreirana je 3D scena u datom *engine*-u u koju su smešteni redovi jagoda kroz koje će robot kasnije prolaziti, te su napravljene 64 slike datih jagoda koje smo podelili na trening (58 slika, 232 jagode) i test skup (8 slika, 32 jagode). Uz pomoć datog skupa podataka odrađen je *fine-tuning ssd_mobilenet_v1_coco* konvolutivne neuronske mreže, gde je uvedena klasa *strawberry* (tj. jagoda). Dati *OD* model je kasnije iskorišten za prepoznavanje zrelih jagoda sa ulaza leve i desne kamere našeg robota, na osnovu koga reaguje sistem za aktivaciju prskalica (leve i desne). Kretanje robota obučeno je korištenjem *A3C RL* algoritma, uz pomoć pogodnosti koje nudi *Unity ML Agents* paket i njegova mogućnost povezivanja sa *python*-om.

OD model je pokazao zadovoljavajuće rezultate, s obzirom na ne preterano velik skup podataka. Evaluacija datog modela vršena je nad test skupom, a korištene su sledeće metrike: preciznost, odziv i F1-mera. Najveća preciznost od 1.000 postignuta je za *IoU* 0.5 i 0.75 nad svim oblastima, najveći odziv od 0.800 za *IoU* 0.50:0.95 nad velikim oblastima, a najveća F1 mera od 0.873 vezana je za pomenutu grupu sa najvećom preciznošću. *A3C* model je pokazao solidne rezultate i evaluiran je empirijski, gde uočavamo da je robot u većini slučajeva uspešno paralelno pratio prvi red jagoda, zatim najčešće uspešno napravio "polukružno skretanje" i na kraju se uspešno kretao između prvog i drugog reda jagoda.

Osim datih rezultata, bitno je napomenuti da je u ovom radu dokazano da je moguće obučiti veštačko-inteligentni model bez velikih novčanih investicija i bez kreiranja bilo kakvog *hardware*-a, putem simuliranja u besplatnim *Unity* alatima. Dati rad predstavlja dobru osnovu ne samo za dalji razvoj datog "virtuelnog" 3D agenta, već i za razvoj stvarnog robota koji bi izvršavao takav ili slične zadatke u poljoprivredi.

5. LITERATURA

Svim navedenim linkovima je pristupljenu u junu 2020.

[1] https://unity.com/

[2] A. Juliani, V-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, D. Lange. Unity: A General Platform for Intelligent Agents, 2018

[3] https://github.com/tensorflow/models/tree/master/-research/object_detection

[4] https://colab.research.google.com/notebooks/-intro.ipynb

[5] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K., 2016, June. Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937).

[6] https://www.tensorflow.org/

[7] http://wiki.fast.ai/index.php/Fine_tuning

[8] http://cocodataset.org/

Kratka biografija:

Timotej Orčić rođen je u Novom Sadu 1996. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Neuronske mreže i Metode poslovne inteligencije odbranio je 2020. god.
kontakt: timotej.orcic@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 007:004.056
DOI: https://doi.org/10.24867/10BE21Indjic

ALAT ZA MERENJE PERFORMANSI OPENMP PROGRAMA

A TOOL FOR MEASURING THE PERFORMANCE OF OPENMP PROGRAMS

Vladimir Inđić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Alat hpcrun je prilagođen tako da koristi OMPT interfejs u cilju olakšane analize performansi izvornog koda OpenMP programa. Upotreba OMPT funkcija povratnog poziva može dovesti do drastičnog povećanja vremena izvršavanja prilikom merenja performansi OpenMP programa sa fino granulisanim paralelizmom.*

Ključne reči: *OpenMP, OMPT, merenje performansi, analiza performansi, fino granulisani paralelizam*

Abstract – *The hpcrun has been extended with OMPT interface support in order to facilitate performance analysis of OpenMP programs' source code. The usage of OMPT call back functions may lead to drastic increase in overhead while measuring the performance of OpenMP programs with fine-grained parallelism.*

Keywords: *OpenMP, OMPT, performance measurement, performance analysis, fine-grained parallelism.*

1. UVOD

Česta upotreba *OpenMP* interfejsa [1] za programiranje višenitnih aplikacija složenih naučnih simulacija, dovela je do potrebe za razvojem kvalitetnih alata za merenje performansi u cilju boljeg razumevanja relativno novog koncepta paralelnog programiranja. Cena programske rutine predstavlja univerzalnu metriku za iskazivanje performansi računarskih arhitektura, poput utrošenog procesorskog vremena, memorijskog zauzeća, itd. S obzirom na to da su cene rutina zavisne od konteksta nastanka njihovog poziva, profajliranje putanji poziva (engl. *call path profiling*) je postao opšte prihvaćen metod za merenje performansi.

Kako bi se smanjio višak vremena uzrokovan profajliranjem putanji poziva (engl. *overhead*), savetuje se upotreba profajlera (engl. *profilers*) zasnovanih na statističkom uzorkovanju [2]. U trenutku prijema uzorka, pomenuti tip profajlera odmotava stek poziva (engl. *call stack*) kako bi formirao kontekst poziva aktivne rutine koji se sastoji od frejmova prisutnih na pomenutnom steku. Formirani konteksti odgovaraju izvršnom kodu *OpenMP* programa, koji u mnogome zavisi od implementacije samog sistema za izvršavanje. Ovo dovodi do skretanje pažnje s analize performansi izvornog koda na analizu performansi implementacije sistema za izvršavanje.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Goran Sladić, vanr. prof.

Kako bi se pomenuti problem prevazišao, neophodno je formirati kontekst poziva rutine koji direktno oslikava izvorni kod. Da bi se to postiglo, potrebno je prevazići problem distribuiranog konteksta nastanka paralelnih regiona [3], a ujedno i sakriti frejmove koji odgovaraju sistemu za izvršavanje iz konteksta poziva rutina. Ovo je bila glavna motivacija za razvoj *OMPT* interfejsa u okviru *OpenMP 5.0* standarda [1]. Pomenuti interfejs je namenjen za upotrebu od strane alata za merenje performansi *OpenMP* programa. Zadatak pomenutog interfejsa je da alatima pruži informacije o trenucima nastanka paralelnih regiona, kao i da ukaže na frejmove sistema za izvršavanje.

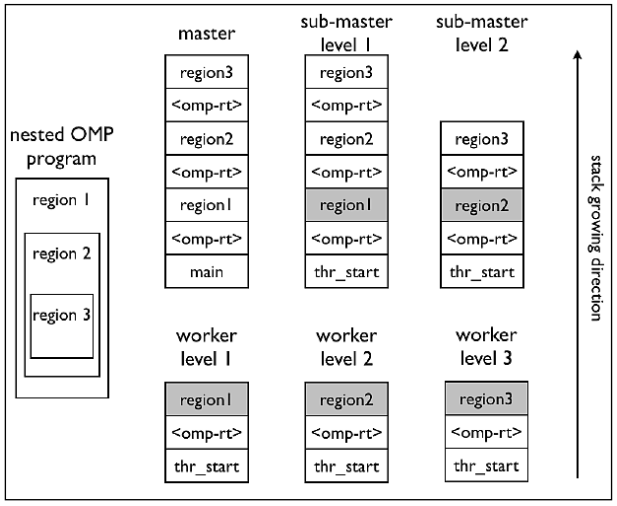
2. Upotreba *OMPT* interfejsa unutar *hpcrun* alata

S obzirom na to da se izvršavanje izvornog koda mahom dešava unutar *OpenMP* zadataka [1], *hpcrun* [4] je proširen podrškom za manipulaciju informacijama o pomenutim zadacima. Zadatak sistema za izvršavanje je da na zahtev profajlera, dostavi informacije koje odgovaraju trenutno aktivnom zadatku, tj. zadatku čiji se kod izvršava u trenutku slanja zahteva. Informacije su predstavljene u vidu 64-bitne reči dodeljene zadatku za vreme njegovog trajanja, kao i para pokazivača na frejmove sistema za izvršavanje koji predstavljaju tačku ulaza u sistem za izvršavanje, odnosno tačku izlaza iz njega [1].

2.1. Formiranje konteksta nastanka paralelnih regiona

Na primeru izvršavanja *OpenMP* programa koji sadrži tri ugnježđena paralelna regiona (slika 1) biće prikazan postupak formiranja konteksta nastanka regiona koji ne sadrži frejmove sistema za izvršavanje. Pomenuti postupak se zasniva na pristupu predloženom od strane *Liu* i ostalih [5]. Unapređenje njihovog pristupa se ogleda u tome što se konteksti nastanka regiona ne skladište u globalnoj *look-up* tabeli, već u 64-bitnoj reči dodeljenoj paralelnom regionu od strane sistema za izvršavanje [1]. Pristup kontekstu regiona više ne zahteva upotrebu složenih sinhronizacionih mehanizama zaduženih da omoguće korišćenje globalne *look-up* tabele od strane više niti konkurentno.

Kako bi se dobile informacije o trenutku nastanka regiona, *hpcrun* mora da registruje funkciju povratnog poziva (engl. *call back function*) koju će sistem za izvršavanje pozvati prilikom kreiranja paralelnog regiona. Pomenutu funkciju izvršiće master niti regiona 1, 2 i 3 prikazanih na slici 1. Njihov zadatak jeste da unutar pomenute funkcije formiraju potpun kontekst nastanka regiona koga kreiraju, pri čemu u obzir moraju uzeti i to da on može biti distribuiran na nekoliko stekova različitih niti.



Slika 1. *Stekovi poziva inicijalne master, podmaster niti i niti radilica tokom izvršavanja OpenMP programa sa tri nivoa ugnježdenog paralelizma [5].*

Naime, prilikom kreiranja regiona 1, inicijalna master nit (*master* na slici 1) odmotava stek poziva i preuzima informacije o implicitnom zadatku. Na osnovu pomenutih informacija pronalazi frejm sistema za izvršavanje koji predstavlja tačku ulaza u pomenuti sistem. Uklanja sve frejmove sistema za izvršavanje koji se nalaze iznad njega (frejmovi sa slike 1 iznad *main* frejma). Na osnovu frejmova koji se nalaze ispod tačke ulaza (*main* frejm na slici 1) formira kontekst nastanka regiona 1, koga čuva u 64-bitnoj reči dodeljenoj ovom regionu.

Tokom kreiranja regiona 2, podmaster nit nivoa 1, odmotava stek poziva i dobavlja informacije o implicitnom zadatku regiona 1. Pomenuta nit je nit radilica regiona 1. Na osnovu navedenih informacija, nit određuje tačke ulaza u sistem za izvršavanje, odnosno izlaza iz njega. Uklanja sve frejmove sistema za izvršavanje koji se nalaze iznad pomenute tačke ulaza (frejmovi sa slike 1 iznad frejma *region1*). Nakon toga, izdvaja frejmove regiona 1 koji su prisutni na njenom steku poziva, a koji se nalaze iznad pomenute tačke izlaza (frejm *region1* sa slike 1). Od izdvojenih frejmova formira delimičan kontekst nastanka regiona 2. Od sistema za izvršavanje traži 64-bitnu reč dodeljenu regionu 1, te iz nje preuzima potpun kontekst nastanka pomenutog regiona. Dodavanjem delimičnog konteksta nastanka regiona 2 na potpun kontekst nastanka regiona 1, formira se potpun kontekst nastanka regiona 2. Potonji biva smešten u 64-bitnu reč dodeljenu pomenutom regionu.

Sličan zadatak ima i podmaster nit nivoa 2 prilikom kreiranja regiona 3. Naime, ona će prvo odrediti delimičan kontekst nastanka regiona 3, koga će pridodati potpunom kontekstu nastanka regiona 2. Na taj način će formirati potpun kontekst nastanka regiona 3, koga smešta u odgovarajuću 64-bitnu reč.

Početak izvršavanja implicitnih, odnosno eksplicitnih zadataka u okviru nekog paralelnog regiona inicira se pozivom pojedinih rutina sistema za izvršavanje. Upravo zato, kao kontekst poziva implicitnog, odnosno eksplicitnog zadatka koristi se kontekst nastanka regiona u okviru koga se pomenuti zadatak izvršava. Javlja se potreba za preuzimanjem konteksta nastanka regiona iz odgovarajuće 64-bitne reči i smeštanjem istog u 64-bitnu reč dodeljenu instanci zadatka. Pomenuti proces obavlja se u funkciji povratnog poziva koja će biti pozvana od strane sistema za izvršavanje prilikom instanciranja implicitnog, odnosno eksplicitnog zadatka.

Kontekst poziva inicijalnog zadatka nema potrebe posebno izdvajati. Naime, on odgovara kodu kojim se započinje izvršavanje programa, te se samim tim nalazi na steku poziva inicijalne master niti.

2.2. Formiranje konteksta poziva rutina

Osnov za formiranje konteksta poziva rutine predstavlja kontekst nastanka regiona, tj. kontekst poziva zadatka u okviru koga se pomenuta rutina izvršava. Potrebno je razmotriti dva slučaju koji se odnose na to da li je uzorak nastao za vreme izvršavanje rutine izvornog koda, odnosno rutine sistema za izvršavanje.

Ukoliko je uzorak nastao za vreme izvršavanja rutine izvornog koda, tada je potrebno formirati kontekst poziva rutine koji se sastoji isključivo od frejmova koji odgovaraju rutinama izvornog koda. U tom slučaju, nit je dužna da odmota stek poziva, kao i da preuzme informacije o aktivnom *OpenMP* zadatku od sistema za izvršavanje. Na osnovu pomenutih informacija, određuje frejm koji predstavlja tačku izlaza iz sistema za izvršavanje, a takođe i kontekst poziva zadatka. Sve frejmove rutina izvornog koda koji se nalaze iznad pomenute tačke izlaza potrebno je iskoristiti kako bi se napravio delimičan kontekst poziva trenutno aktivne rutine izvornog koda. Ovaj kontekst biva pridodat kontekstu poziva zadatka u cilju formiranja potpunog konteksta poziva rutine.

Postavlja se pitanje šta uraditi ukoliko uzorak nastane za vreme izvršavanje rutine sistema za izvršavanje. To se može desiti ukoliko nit čeka da bude upošljena ili ukoliko izvršava određeni posao unutar samog sistema, a koji se tiče kreiranja regiona, kreiranja eksplicitnog zadatka, redukovanja rezultata, pokušaja ulaska u kritički region, itd. Slučaj čekanja niti unutar sistema biće obrađen u sledećem potpoglavlju.

Ukoliko nit obavlja određeni posao unutar sistema za izvršavanje, to znači da ona resurse troši na *overhead* izazvan implementacijom sistema za izvršavanje. Ideja je da se prikazivanjem frejmova sistema za izvršavanje koji se nalaze na vrhu steka, ukaže na neoptimalnosti u implementaciji samog sistema. Naime, nit će odmotati stek i preuzeti informacije o aktivnom *OpenMP* zadatku. Odrediće tačku ulaza u sistem za izvršavanje, kao i kontekst poziva zadatka. Izdvojiće frejmove sistema za izvršavanje koji se nalaze iznad pomenute tačke ulaza, te od njih stvoriti delimičan kontekst poziva aktivne rutine sistema. Delimičan kontekst biva pridodat na kontekst poziva *OpenMP* zadatka, u cilju formiranja potpunog konteksta poziva rutine sistema za izvršavanje.

2.3. Čekanje na poslednjoj implicitnoj barijeri

Ukoliko uzorak nastane dok nit radilica izvršava rutine sistema za izvršavanje kojima se reprezentuje čekanje na poslednjoj implicitnoj barijeri paralelnog regiona, tada se na njenom steku poziva nalaze samo frejmovi sistema za

izvršavanje. Oni će biti uklonjeni i zamenjeni jednim *idle* frejmom. Nakon toga, potrebno je odrediti kontekst čekanja niti. S obzirom na to da nit radilica ne zna da li region na čijoj barijeri čeka i dalje postoji [1], cene čekanja biće akumulirane u privatnoj promenljivoj niti (engl. *thread-private variable*). Ukoliko nit u određenom trenutku primi uzorak izvršavajući eksplicitni zadatak u okviru istog regiona, tada će se akumulirana cena čekanja dodeliti kontekstu nastanka istog. Na ovaj način se korisniku ukazuje da je čekanje izazvano nejednakom raspodelom posla, koja je izazvana izvornim kodom regiona. Ukoliko pak, nit radilica primi sledeći uzorak tokom izvršavanja rutina nekog drugog paralelnog regiona ili sama nit bude uništena od strane sistema za izvršavanje, tada će se akumulirana cena čekanja dodeliti globalnom kontekstu programa. Na ovaj način će korisniku biti ukazano da izvorni kod programa ne generiše dovoljno posla da uposli sve raspoložive resurse.

Inicijalna specifikacija *OMPT* interfejsa [1] nije omogućavala dovoljno informacija na osnovu kojih bi se moglo zaključiti da li nit čeka na poslednjoj implicitnoj barijeri. U cilju prevazilaženja ovog nedostatka, predložene su izmene koje su se ticale manipulacije tipovima *OpenMP* barijera. Od sistema za izvršavanje se zahteva da u svakom trenutku može odrediti na kojoj vrsti barijere unutar regiona nit čeka. Predložene izmene su prihvaćene u trenutnoj reviziji *OpenMP 5.1* standarda [6].

Ukoliko pak master nit čeka na poslednjoj implicitnoj barijeri, tada je postojanje regiona garantovano, te se cena čekanja može dodeliti konteksu nastanka paralelnog regiona. Slično je i sa čekanjem niti na drugim tipovima *OpenMP* barijera, te se njima neće posvetiti pažnja u ovom radu.

2.3. Evaluacija rešenja

Upotrebom *OMPT* interfejsa unutar *hpcrun* alata, cene izvršavanja rutina *OpenMP* programa dodeljene su kontekstima poziva koji odgovaraju izvornom kodu. Na ovaj načina olakšana je analiza performansi izvornog koda pomenutih programa.

Neophodno je izmeriti *overhead* unet upotrebom *OMPT* interfejsa. Ovo će biti urađeno putem dva eksperimenta, koji se zasnivaju na profajliranju nekoliko reper-programa (engl. *benchmarks*). Oba eksperimenta su obavljena na računaru koji ima *Intel Xeon Phi 7250* procesor radnog takta 1,4 GHz, sa 68 jezgara i sa po 4 hardverske niti na svakom jezgru. Od ukupnog broja niti, radi relevantnosti rezultata, korišćeno je 204 niti. Prilikom profajliranja, korišćen je *REALTIME* sistemski brojač sa periodom generisanja uzorka od 10 ms.

Prvi eksperiment se zasniva na profajliranju *benchmark*-a iz skupa *SPEC OMP 2012* [7]. Kako bi se odredio *overhead* profajliranja, svaki od *benchmark*-a je pokrenut po tri puta samostalno, nakon čega je određena srednja vrednost vremena izvršavanja (*avg_bench* u formuli (1)). Nakon toga, svaki od *benchmark*-a je izvršen još po tri puta, pri čemu je njegovo izvršavanje profajlirano upotrebom *hpcrun* alata. Određena su i srednja vremena profajliranih izvršavanja (*avg_hpcrun* u formuli (1)). Na kraju, *overhead* izazvan profajliranjem izračunat je po formuli (1). Radi preglednosti, *overhead* je izražen u procentima:

$$overhead = \left(\frac{avg\_hpcrun}{avg\_bench} - 1\right) * 100 \qquad (1)$$

Kao što se u tabeli 1 može videti, *hpcrun* unosi nizak kontrolisan *overhead* u granicama od 1% do 5%. Izuzetak predstavlja *benchmark 376.tree* koji generiše izuzetno veliki broj kratkih eksplicitnih zadataka. Razlog za to je funkcija povratnog poziva zadužena da odredi kontekst poziva eksplicitnog zadatka. Poziv pomenute funkcije je skuplji od količine posla koju svaki od zadataka obavlja, te dolazi do drastičnog porasta *overhead*-a.

Tabela 1. *Overhead iskazan u procentima unet od strane hpcrun alata prilikom profajliranja SPEC OMP 2012 skupa benchmark-a*

Benchmark	Overhead
350.md	2.62
351.bwaves	2.78
352.nab	1.68
357.bt331	1.40
358.botsalgn	1.50
359.botsspar	2.30
360.ilbdc	3.95
362.fma3d	0.97
363.swim	1.41
367.imagick	2.31
370.mgrid331	0.98
371.applu331	2.12
372.smithwa	1.50
376.kdtree	460.09

Kako bi se pomenuti problem rešio, trebalo bi izbeći registrovanje prethodno pomenute funkcije. Umesto toga, trebalo bi pokušati sa određivanjem konteksta poziva zadatka prilikom obrade uzorka nastalog za vreme izvršavanja zadatka. U tom slučaju, moglo bi se pokušati sa direktnim pristupom kontekstu nastanka regiona u okviru koga se zadatak izvršava, čime bi se izbegla potreba za njegovim kopiranjem u 64-bitnu reč dodeljenu zadatku.

Drugi eksperiment se zasniva na određivanju *overhead*-a nastalog profajliranjem *OpenMP* programa sa fino granulisanim paralelizmom. U tu svrhu, korišćen je *testpr benchmark EPCC* skupa [8]. Pomenuti *benchmark* je iskonfigurisan tako da generiše milion i po kratkih paralelnih regiona, pri čemu svaki od njih izvršava posao u trajanju od oko 1 μs. Pod pojmom kratkog paralelnog regiona, podrazumeva se region čije je trajanje kraće od perioda generisanja uzorka.

Za razliku od prethodnog eksperimenta, *tesptr* je pokrenut 10 puta samostalno, a zatim još 10 puta pri čemu je njegovo izvršavanje profajlirano *hpcrun* alatom. Značenje *avg_bench* i *avg_hpcrun* iz formule (1) promenjeno je tako da određuje srednje vreme potrebno za izvršavanje jedne instance kratkog paralelnog regiona, odnosno izvršavanje instance regiona profajlirano od strane *hpcrun* alata. Eksperiment je pokazao da *hpcrun* unosi *overhead* od 198%. Razlog za to je taj što su regioni veoma kratki, te proces formiranja konteksta nastanka regiona traje duže od njih samih. S obzirom na to da su regioni dosta kraći od perioda generisanja uzorka, velika je verovatnoća da niti neće primiti uzorak za vreme izvršavanja mnogih od njih. Samim tim, konteksti nastanka regiona za vreme

čijeg izvršavanja nisu nastali uzorci bivaju neiskorišćeni, te je njihovo formiranje suvišno.
Nizak *overhead* prilikom profajliranja je neophodan kako bi se dobila realna slika o iskorišćenosti modernih mikroprocesorkih arhitektura tokom izvršavanja fino granulisanog paralelizma [9]. Upravo zbog toga, neophodno je uvesti određene modifikacije pristupa formiranja konteksta nastanka regiona koji *hpcrun* koristi. Umesto formiranja konteksta prilikom kreiranja regiona, potrebno je odložiti proces formiranja do trenutka kada je on zaista neophodan. Odlaganja formiranja konteksta sa sobom nosi i nekoliko izazova.
Pre svega, neophodno je odrediti kada je zaista neki kontekst nastanka regiona neophodan. To znači da niti radilice moraju biti u stanju da obaveste master nit da su primile uzorak za vreme trajanja regiona, te da im je njegov kontekst neophodan. S druge strane, master nit mora biti u stanju da obavesti sve niti radilice o dostupnosti konteksta nastanka regiona u trenutku kada je formiran. Postavlja se pitanje kako formirati kontekst poziva rutine za vreme čijeg izvršavanje je nastao uzorak, ukoliko kontekst nastanka regiona još uvek nije formiran. Takođe, javlja se i problem prilikom formiranja konteksta nastanka ugnježdenih regiona u trenucima nedostupnosti konteksta spoljašnjih regiona. Razrešavanje pomenutih izazova predstavlja prostor za neophodno poboljšanje *hpcrun* alata u cilju primene nad *OpenMP* programima sa fino granulisanim paralelizmom.

3. ZAKLJUČAK

Alat *hpcrun* je modifikovan tako da koristi informacije o *OpenMP* zadacima upotrebom *OMPT* interfejsa. Na taj način, omogućeno je formiranje konteksta poziva rutina distribuiranih na nekoliko stekova različitiih niti, kao i uklanjanje nepotrebnih detalja implementacije. Formirani konteksti u potpunosti odgovaraju izvornom kodu, te olakšavaju analizu performansi istog. Informacije o sistemu za izvršavanje prikazuju se korisniku samo u situacijama kada je potrebno ukazati na *overhead* izazvan implementacijom pomenutog sistema.
Potrebno je posebno ukazati na čekanje niti na poslednjoj implicitnoj barijeri regiona, koje je uzrokovano izvornim kodom istog. Pomenuti kod ili ne generiše dovoljno posla da se iskoriste svi dostupni resursi ili posao neravnomerno raspoređuje na niti tima paralelnog regiona. Da bi se odredilo čekanje na pomenutoj barijeri, predložene su izmene trenutne specifikacije *OMPT* interfejsa koje se tiču manipulacije tipovima *OpenMP* barijera. Pomenute izmene su prihvaćene u trenutnoj reviziji *OpenMP 5.1* standarda.
Alat *hpcrun* uglavnom unosi nizak kontrolisan *overhead* prilikom profajliranja *OpenMP* programa. Izuzetak predstavljaju programi sa fino granulisanim paralelizmom. Kako bi se smanjio *overhead* nastao prilikom profajliranja pomenutih programa, neophodno je revidirati postupak formiranja konteksta nastanka regiona. Umesto formiranja pomenutog konteksta u trenutku kreiranja paralelnog regiona, predlaže se odlaganje formiranja konteksta do trenutka kada je pomenuti zaista neophodan.
Odlaganje unosi vremenski okvir u kome neka od niti može primiti uzorak izvršavajući rutine regiona čiji kontekst nije formiran. To dovodi do potrebe za novim vidom saradnje niti paralelnog regiona u cilju razmene informacija o potrebi konteksta, odnosno o njegovom formiranju. Istraživanje na temu ovakvog vida saradnje može doprineti značajnom poboljšanju kvaliteta merenja performansi *OpenMP* programa sa fino granulisanim paralelizmom.

4. LITERATURA

[1] https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf (pristupljeno u novembru 2018.)
[2] N. Froyd, J. Mellor-Crummey, R. Fowler. Low overhead call path profiling of unmodified, optimized code. *In Proceedings of the 19th annual international conference on Supercomputing*, pages 81–90, 2005.
[3] https://www.openmp.org/wp-content/uploads/ompt-tr2.pdf (pristupljeno u julu 2017.)
[4] L. Adhianto, et all. Hpctoolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.
[5] X. Liu, J. Mellor-Crummey, and M. Fagan. A new approach for performance analysis of openmp programs. *In Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 69–80, 2013.
[6] https://www.openmp.org/wp-content/uploads/openmp-TR9.pdf (pristupljeno u martu 2020.)
[7] https://www.spec.org/omp2012/ (pristupljeno u junu 2020.)
[8] https://www.epcc.ed.ac.uk/research/computing/-performance-characterisation-and-benchmarking/epcc-openmp-micro-benchmark-suite (pristupljeno u junu 2018.)
[9] A. E. Eichenberger, K. O'Brien. Experimenting with low-overhead openmp runtime on ibm blue gene/q. *IBM Journal of Research and Development*, 57(1/2):8–1, 2013.

Kratka biografija:

Vladimir Inđić rođen je u Šapcu 1995. godine. Dilpomirao je na Fakultetu tehničkih nauka 2019. godine iz oblasti Elektrotehnike i računarstva – Softversko inženjerstvo i informacione tehnologije sa prosečnom ocenom 10. Od jula 2017. godine radi na unapređenju *hpcrun* alata pod mentorstvom prof. dr *John Mellor-Crummey*, vođe *HPCToolkit* projekta (*Rice* Univerzitet, Hjuston, Teksas, SAD).

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.738.5:339
DOI: https://doi.org/10.24867/10BE22Ivanovic

AUTOMATIZACIJA TESTIRANJA SOFTVERA U OKVIRU MASTER-AGENT ARHITEKTURE

SOFTWARE TESTING AUTOMATION WITHIN MASTER-AGENT ARCHITECTURE

Bojana Ivanović, Branko Milosavljević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je opisan sistem za automatizaciju testiranja modelovan kroz master-agent arhitekturu, opisan je rad agentskog dela sistema unutar kontejnerskog okruženja. Pored implementacionih detalja istaknuti su uočeni nedostaci koji su doveli do proširenja sistema i dati su predlozi za buduće širenje istog.*

Ključne reči: *Automatizovano testiranje, Master-agent arhitektura, Jenkins, Allure, Docker*

Abstract – *Paper describes a test automation system modeled through the master-agent architecture and agent part of the system within container enviroment. In addition to implementation details, the observed shortcomings that led to the expansion of the system were highlighted with also suggestions for possible future expansion of the system.*

Keywords: *Test automation, Master-agent architecture, Jenkins, Allure, Docker*

1. UVOD

Automatizacija nam donosi velike prednosti u odnosu na manuelni rad, a to su: povećana produktivnost, efikasnija upotreba resursa, veći kvalitet proizvoda, poboljšana sigurnost, manja verovatnoća za ljudske greške, smanjenje obima posla koji obavlja čovek itd. U pogledu informacionih tehnologija, automatizacija je od ključnog značaja, pre svega zbog velikog interesa za smanjenjem verovatnoće za grešku koja dolazi sa čovekovim uključenjem u rad, ali i povećane efikasnosti i svih drugih prednosti automatizacije nad manuelnim radom.

Automatizacija je kao paradigma unela velike promene u savremeni razvoj softvera. Bilo da je taj softver deo sistema velike hidroelektrane, broda, aviona ili sistema bioskopa za rezervaciju karata, očekujemo da on radi ono što mu je i namena i to tako da radi bez grešaka. U praksi je gotovo nemoguće napisati *bug-free* softver te su prethodna očekivanja nerealna, ali je moguće napraviti dovoljno dobar softver time što će se broj problema u njemu svesti na minimum. Ovo možemo postići testiranjem softvera. Testiranje je obavezan zadatak tokom razvoja svakog softvera i onaj koji najteže pada svim programerima. Mnogi kažu da pisanje testova unosi greške u rešenje tj. softver, ali to nije ni blizu tačnog. Testiranje nam izvlači probleme na površinu, kako bismo njihovim otklanjanjem došli do „skoro savršenog“ softverskog rešenja.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.

Izvršavanje testova napisanih za neki softver traje prilično dugo. Pokretanje stotine ili hiljade napisanih testova za neki veliki sistem, kao što je sistem hidroelektrane ili makar jednog njenog segmenta traje dugo i potrebno je da efikasnije iskoristimo to vreme. Dodatno, ukoliko je testove potrebno pokrenuti u nekom definisanom momentu, nakon spremne nove verzije softvera, na zahtev vođe tima i sl. onda taj posao pokretanja testova nije nešto što je potrebno uraditi jednom, već svaki put kada je to potrebno, a to je u praksi prilično često. Ovaj proces se može ubrzati tako što će se testiranje automatizovati. Ideja iza automatizacije testiranja bi bila da možemo pokrenuti sve testove koji su nam potrebni odjednom, bez potrebe za pojedinačnim pokretanjem svakog od testova ili grupe testova. Ono što je prednost ovakvog testiranja jeste što ne zahteva čovekovo učešće - jednom kada je pokrenuto ono se izvršava dok ne izvrši svaki od zahtevanih testova. Kako bi se izvršilo automatizovano testiranje nekog softvera, treba sagledati prvo zahteve samog softvera, preferencije tima ili kompanije i na osnovu toga odabrati najbolji alat. Sve prethodno navedeno se tiče isključenja čoveka iz nekog rada i ubrzanjem posla koji inače čovek obavlja. Jednom kada su testovi pokrenuti bitno nam je gde će se ti testovi izvršavati tj. na kom računaru i na koji način, a o tome će biti reči u nastavku ovog rada.

Ovaj rad nastao je kao rezultat rada na ideji proistekloj iz potrebe za automatizovanim testiranjem tj. kreiranjem arhitekture koja bi dovela do ubrzanja testiranja i veće produktivnosti. Takođe, ideja za kreiranjem podstaknuta je i potrebom za lakšom distribucijom rešenja klijentima koje bi za njihov konkretan softver pružilo potpuno fleksibilno testno okruženje.

2. TEORIJSKI KONCEPTI I TEHNOLOGIJE

2.1. Automatizovano testiranje

Savremeno doba je u razvoju softvera dovelo do situacija gde su timovi koji rade na razvoju softvera pod konstantnim pritiskom rokova za isporuku softvera, kao i konkretnih zahteva od strane klijenata. Ono sa čime se timovi i menadžeri susreću svakoga dana jeste da, pored poštovanja rokova, treba softver razviti uz minimalnu potrošnju resursa, pre svega novca. Kao deo činjenice da treba pružiti što više uz što manju potrošnju resursa, javlja se i to da kompanije žele da adekvatno testiraju softver koliko god je to moguće brzo, ali i temeljno. Da bi se taj cilj postigao, potrebno je okrenuti se automatizovanom načinu testiranja [1]. Može se reći da automatizovano testiranje predstavlja upravljanje i izvršavanje testnih radnji, uključujući razvoj i izvršavanje skripte za testiranje kako bi se potvrdili zahtevi za testiranjem pri čemu se koristi neki od alata za

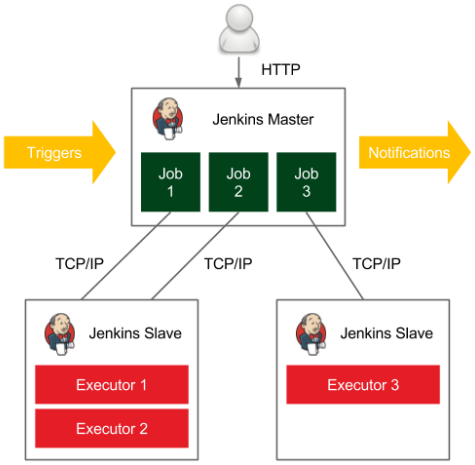
automatsko testiranje. Automatizacija testiranja svoje najveće prednosti pokazuje u situacijama gde se test skripte (ili njihovi delovi) izvršavaju često.

2.2. Master-agent arhitektura

Master-agent arhitektura (ranije poznata kao *master-slave*) predstavlja mehanizam koji se pre svega razvio za potrebe repliciranja baza podataka. Danas se ovaj koncept koristi u distribuiranim sistemima kao model komunikacije gde jedan uređaj ili proces (jedan od čvorova sistema) ima moć nad ostalim uređajima ili procesima. U toj komunikaciji master čvor se posmatra kao kontroler čija je jedina funkcija da vrši odlučivanje kom od potčinjenih čvorova da delegira neki posao. Posao potčinjenog čvora jeste da jednom kada dobije posao koji mu je dodeljen isti i izvrši i rezultate vrati master čvoru. Jednom kada dobije podatke, master čvor u zavisnosti od potreba servira rezultate klijentu ili ih šalje na dalju obradu ili čuvanje.

2.3. Jenkins

Jenkins je samostalni server otvorenog koda (engl. *open source*), koji se koristi za automatizaciju svih vrsta zadataka koji se odnose na izgradnju, testiranje, isporuku ili upotrebu softvera, olakšavajući kontinuiranu integraciju i isporuku softvera. Zasnovan je na serveru koji radi u servletskim kontejnerima kao što je Apache Tomcat.
Kod Jenkins-a master-agent koncept igra bitnu ulogu u delegiranju poslova. Slika 1 prikazuje delegiranje poslova od strane master čvora ka agentskim čvorovima. Kako bi se ovaj koncept implementirao, potrebno je definisati čvor koji će biti agent i kome će posao biti delegiran od strane mastera, gde agentskih čvorova može biti bezbroj. Kod agent čvora je potrebno definisati dodatne alate (engl. *plugin*) koje će agentski čvor direktno koristiti u obavljanju dodeljenog posla, ukoliko su mu oni potrebni. Pod master čvorom podrazumeva se Jenkins serverska aplikacija koja koordiniše definisanim agentskim čvorovima.



Slika 1. *Jenkins-ov master-agent koncept delegiranja poslova [3]*

Upotrebom Jenkins-a moguće je definisati različite poslove (engl. *job*) koji se mogu pokrenuti u bilo kom trenutku. *Job*-ovi su središte Jenkins *build* procesa. Na njih se može gledati kao na posebne zadatke ili korake celokupnog *build* procesa. Ovaj zadatak može da predstavlja samo kompajliranje izvornog koda ili pokretanje unit testova. S druge strane, *job* ne mora biti tako jednostavan. On može da meri pokrivenost koda testovima, primenjuje na izvornom kodu metrike kojima meri kvalitet napisanog koda, takođe može da generiše tehničku dokumentaciju ili čak izvrši *deploy* aplikacije na veb server [2].

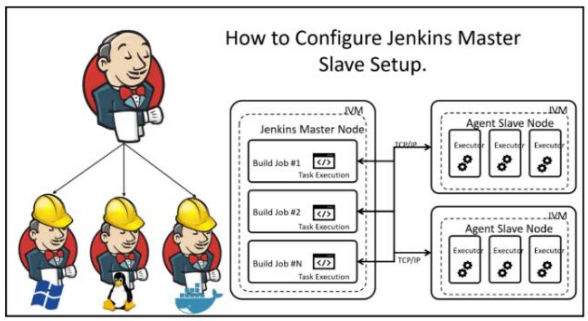
2.4. Allure

Allure radni okvir se razvija kao softver otvorenog koda i njegova uloga je da kreira izveštaj o testiranju koji bi bio jasan, ali sa dovoljno detalja o rezultatima izvršenih testova**Error! Reference source not found.**. Ovaj radni okvir je fleksibilan alat za generisanje izveštaja o testiranju, koji pruža koncizan izveštaj svega što je testirano i omogućuje svima koji rade na razvoju konkretnog softvera da izvuku što više korisnih informacija. Pored dosta korisnih informacija, Allure rezultate prikazuje i putem različitih grafičkih elemenata.

3. MODEL SISTEMA

Sistem za automatizovano testiranje baziran na master-agent arhitekturi sastoji se iz master čvora koji se nalazi unutar oblaka (engl. *cloud*), dok je agentski čvor fizička mašina. Ova arhitektura je implementirana upotrebom *Jenkins* tehnologije. Master čvor, delegira posao agentskim čvorovima, gde je za potrebe ovog rešenja bio potreban jedan agentski čvor. Sam agentski čvor izvršava dodeljeni posao i u slučaju ovog sistema taj posao izvršava se na fizičkom računaru koji komunicira sa masterom preko TCP/IP protokola. Na računaru gde će se obavljati posao koji dobije agentski čvor može biti bilo koji operativni sistem, što i prikazuje Slika 2.

U slučaju ovog konkretnog sistema za automatizaciju testiranja, na mašini koja obavlja posao nalazi se kontejnerizovana aplikacija. Kontejnerizacija je izabrana zato što je potrebno da automatizovano testiranje bude izvršeno u strogo kontrolisanom okruženju koje se može konfigurisati tako da nije potrebno vršiti nikakva dodatna podešavanja na mašini domaćinu. Ono što je velika prednost kontejnerizovane aplikacije, a što je jedan od razloga za njenu upotrebu, jeste činjenica da se takva aplikacija lakše distribuira korisnicima i u tu svrhu je upotrebljen Docker.

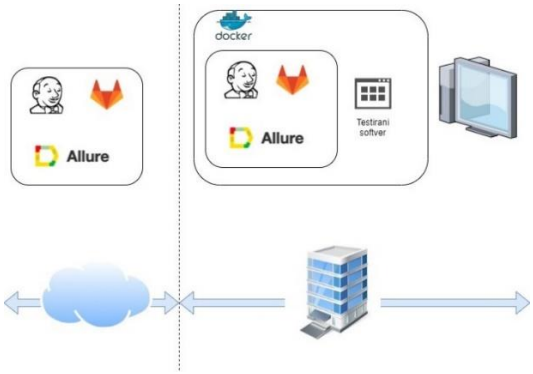


Slika 2. *Master-agent arhitektura sa prikazom različitih tipova agenata [4]*

4. IMPLEMENTACIJA

Sistem za automatizovano testiranje softvera u okviru master-agent arhitekture, implementiran za potrebe ovog rada, opisuje Slika 3, gde se može videti da je arhitektura takva da postoji *cloud* deo sistema, gde se nalazi Jenkins master čvor koji kontroliše obavljanje poslova koje vrši agent koji je predstavljen kao Docker kontejner. Tok testiranja je takav da master delegira posao za testiranje,

koji uključuje testove smeštene na Git repozitorijumu i ovi testovi se preuzimaju na agentsku mašinu gde se vrši testiranje. Jednom kada se testiranje završi, kao izlaz iz procesa generiše se izveštaj upotrebom Allure radnog okvira. Ovaj izveštaj se prosleđuje nazad do master čvora, gde korisnik može da ga pregleda.



Slika 3. *Arhitektura sistema za automatizovano testiranje softvera*

4.1. Kreiranje agentskog čvora

Kreiranje agentskog čvora moguće je izvršiti preko Jenkins aplikacije, postavljanjem osnovnih informacija o čvoru. Ono što se ističe jesu informacije potrebne za rad agenta, poput direktorijuma koji će se koristiti za rad samog Jenkins-a, zatim načina na koji će se koristi konkretan čvor, gde je u slučaju ove implementacije odabrana opcija da se on koristi maksimalno. Podešavanja agentskog čvora koja se tiču načina njegovog pokretanja zavise od načina povezivanja agentskog čvora sa masterom. U implementiranom rešenju odabran je način pokretanja agenta povezivanjem na master.
Ova opcija je zahvalnija za korišćenje u odnosu na preostale zbog manje potrebe za konfiguracijom, a pored toga je i preporučena među dostupnim opcijama za korišćenje. Pored načina pokretanja, ono što može biti važno za agentski čvor jesu alati koje on treba da koristi. Ono što je bilo potrebno za agentski čvor korišćen u ovom rešenju jeste Git, kako bi agentski čvor mogao da preuzme testove sa GitLab repozitorijuma. Da bi Git bio dostupan kao jedan od alata prilikom ove definicije, potrebno je da postoji kao jedan od *plugin*-a i da bude definisan kao globalni alat na nivou cele Jenkins aplikacije. Pored podrške za rad sa Git alatom, za potrebe ovog rešenja bitan je i rad sa Allure radnim okvirom radi generisanja finalnog izveštaja o testiranju, čiji je plugin takođe potrebno dodati na nivou cele Jenkins aplikacije.

4.2. Definisanje posla

Potreba za definisanjem zadatka u ovom rešenju bila je ta da se pokrene izvršavanje testova i kao izlaz tog testiranja servira Allure izveštaj o testiranju. Takođe, zadatak bi trebalo da pre pokretanja testova preuzme testove sa repozitorijuma gde se oni čuvaju. Postoji više načina za definiciju posla, a u implementiranom rešenju isprobani su *Pipeline* i *Free project.*

Prvo je isproban *Pipeline* mehanizam za definisanje poslova, ali se od istog odustalo. Ono što je loša strana ovog pristupa jeste skripta preko koje se definiše zadatak. Ona vremenom i širenjem zahteva raste i postaje nepregledna, te iako je reč o programskom kodu koji se piše unutar skripte, ipak nije jednostavno za održavanje na duže staze. Ono što se nametnulo kao dodatni zahtev koji bi dodatno automatizovao testiranje jeste da se pre pokretanja testova oni preuzmu sa repozitorijuma za čuvanje koda i smeste na neku lokaciju i nad tim preuzetim testovima pokrene izvršavanje. Ovaj zahtev nije mogao u potpunosti da se uklopi sa *Pipeline* mehanizmom, tj. Git kao alat nije bilo moguće integrisati u ovaj način rada.

Nakon isprobanog *Pipeline* mehanizma i uviđanja nedostataka za implementirano rešenje, izabran je *Freestyle project* koji predstavlja u Jenkins-u način definicije posla tj. zadatka sa mnogo širim setom mogućnosti nego što je bio slučaj kod *Pipeline*-a. Pored toga, ovaj tip zadatka tj. forma za njegovo kreiranje je jednostavnija i preglednija sa stanovišta korisničkog interfejsa, lakše je za rad i shvatanje čak i za osobe koje nisu programeri ili testeri.

Ono što je bila jedna od prepreka *Pipeline*-a jeste upotreba alata za kontrolu verziju i to je ono što *Source Code Management* sekcija rešava. Kako bi bilo moguće preuzeti kod sa repozitorijuma, potrebno je u okviru pomenute sekcije navesti putanju do repozitorijuma i izabrati način autentifikacije. Moguće je autentifikaciju vršiti koristeći GitLab kredencijale ili putem *SSH*, gde su u implementi-ranom rešenju isprobana oba ekvivalentna načina. Ostale sekcije definicije posla predstavljaju korake u kojima se definiše naredba za pokretanje testiranja i definisanje direktorijuma za smeštanje izveštaja o testiranju i one su identične u oba načina definicije posla.

4.3. Jenkins unutar Docker kontejnera

Kako je jedan od ključnih zahteva bio da sistem za automatizaciju testiranja radi unutar kontejnera, potrebno je omogućiti rad Jenkins-a unutar kontejnera. S jedne strane, da bi čvor imao ponašanje mastera potrebno je da sadrži celu Jenkins aplikaciju kako bi pružio sve funkcionalnosti. Ovo se može omogućiti pakovanjem Jenkins-a unutar Docker-a na dva načina: upotrebom Jenkins-a kao *base image*-a ili pakovanjem *jenkins.war*. Dok, ako je potrebno da se čvor ponaša kao agent unutar kontejnera, potrebno je obezbediti agentsko ponašanje, pakovanjem *agent.jar* unutar Docker-a.

Kada je potrebno da se čvor ponaša i kao agent i kao master tj. da bude u stanju da izvršava poslove kao agent, ali da bude master za druge agentske čvorove, potrebno je spojiti dva prethodna pri-stupa, te čvor mora posedovati i *jenkins.war* i *agent.jar*. Za potrebe implementiranog rešenja kreiran je jedan Docker *image* koji ima ponašanje i mastera i agenta, ali i jedan agentski *image* i oba Docker *image*-a su testirana tj. korišćena tokom evaluacije implementiranog rešenja.

4.4. Testiranje upotrebom Pytest radnog okvira

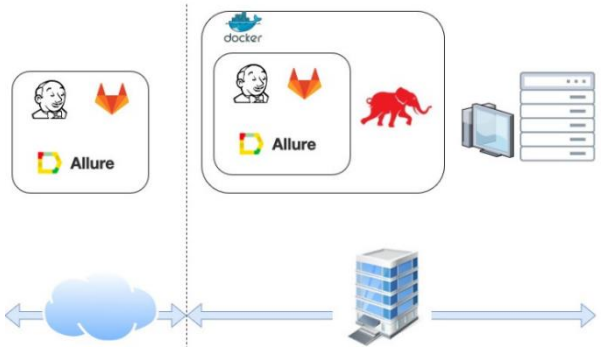
U ranoj fazi implementacije cilj je bio da u master-agent arhitekturi agentski čvor izvršava testove napisane kroz Pytest radni okvir i u početku je testirani set činilo tek nešto preko desetak testova. Svaki od tih testova bazirao se na jednostavnim tvrdnjama (asertacijama). Cilj ovih malobrojnih i jednostavnih testova bio je u tome da se

isproba ceo sistem i tok, od pokretanja posla na masteru do izvršavanja i generisanja izveštaja o istom.

4.5. Testiranje softvera u okruženju simulacije u realnom vremenu

Početna ideja ovog rada bila je napraviti okruženje za testiranje koje radi u master-agent režimu za konkretan softver kompanije Typhoon HIL i to unutar sistema čiji je model ranije opisan. Ideja je bila integrisati sistem za automatizovano testiranje da radi sa konkretnim softverom kompanije i bibliotekom TyphoonTest koju on koristi. Ova biblioteka je zasnovana na Pytest radnom okviru koji se koristio u početnoj fazi razvoja i predstavlja dodatak na pomenuti radni okvir. Kompanija koristi ovu biblioteku za izvršavanje testova koji su pisani za testiranje sistema u realnom vremenu. Da bi se testirali sistemi u realnom vremenu, ostvaruje se saradnja sa simulatorom gde će se praktično test izvršiti i vratiti rezultat. TyphoonTest je deo TyphoonTest IDE, a koji je deo THCC (*Typhoon HIL Control Center*) softvera. Takođe, kako je na kraju potrebno prikazati izveštaj o testiranju u vidu Allure izveštaja korišćen je i typhoon-allure koji je deo THCC-a i predstavlja *wrapper* oko samog Allure radnog okvira.

Implementirano rešenje radi u master-agent arhitekturi, gde se master čvor nalazi unutar oblaka. Kako je ovaj sistem kreiran za domenski softver kompanije Typhoon HIL, njihov kontrolni centar je spakovan unutar kontejnera koji predstavlja agentski čvor. Slika 4 prikazuje implementirani sistem, gde je kontejnerizovan THCC softver.



Slika 4. *Arhitektura sistema za automatizovano testiranje Typhoon HIL softvera*

4.6. Proširenje sistema

Tokom rada na implementaciji projektnog rešenja, došlo se do zaključka da je činjenica o tome da korisnik ne vidi nikakve rezultate sve dok se testiranje ne završi, dosta problematična. Samo testiranje može da traje nekoliko sekundi, ali i do nekoliko sati. Ono što je mana ovog sistema, kada je reč o prikazu rezultata, jeste to što su rezultati vidljivi samo na masteru. Master čvor tj. Jenkins aplikacija u okviru koje će biti serviran Allure izveštaj o testiranju uglavnom nije dostupna korisniku ili možda ne svim korisnicima, što znači da ukoliko korisnik nema pristup masteru ne može da vidi rezultate. Pored vizuelizacije rezultata, javlja se i problem čuvanja Allure izveštaja, koji zapremaju dosta memorijskog prostora, jer su sačinjeni od mnogobrojnih *.json* datoteka. Njihovo čuvanje na duže staze predstavlja ozbiljan problem.

Zbog pomenutih razloga celokupan sistem je proširen tako što je kreiran još jedan Docker kontejner u koji su smeštene dve aplikacije: Django aplikacija koja nadgleda testiranje i Angular aplikacija koja vrši prikaz pređašnjih rezultata testiranja i pruža korisniku bolje iskustvo, ali korisnik i dalje ne vidi u realnom vremenu rezultate što je mana. Takođe, kao serverska aplikacija koja radi sa podacima, podignuta je Spring boot aplikacija koja radi sa MySQL bazom podataka u kojoj se pamte informacije o testiranju i ova aplikacija se nalazi unutar oblaka.

5. ZAKLJUČAK

U radu je opisan sistem za automatizaciju testiranja modelovan kroz master-agent arhitekturu, gde je opisan rad agentskog dela sistema unutar kontejnerskog okruženja, kao i primena ovog sistema za automatizaciju testiranja sistema u realnom vremenu. U glavnom poglavlju o implementaciji je prikazan celokupan implementirani sistem.

Kao neki predlog budućeg razvitka ovog rada jeste omogućiti korisniku uvid u rad sistema u realnom vremenu. Takođe, ovo rešenje je isprobano, tako što su pokretani skupovi testova reda veličine do par stotina testova. Bilo bi interesantno ovo rešenje primeniti za testiranje nekog velikog sistema poput sistema automobila, aviona ili hidroelektrane sa količinom testova gde red veličine skupa testova ide do nekoliko hiljada testova i posmatrati ponašanje sistema.

6. LITERATURA

[1] Dustin, Elfriede, Jeff Rashka, and John Paul. *Automated Software Testing: Introduction, Management, and Performance: Introduction, Management, and Performance*. Addison-Wesley Professional, 1999.

[2] Smart, John Ferguson. *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. " O'Reilly Media, Inc.", 2011.

[3] https://www.oreilly.com/library/view/continuous-delivery-with/9781787125230/assets/162a6bf6-9b46-4604-93ec-d818fbfde1f9.png (pristupljeno u septembru 2020.)

[4] https://digitalvarys.com/how-to-configure-jenkins-master-slave-setup/ (pristupljeno u septembru 2020.)

Kratka biografija:

Bojana Ivanović rođena je u Novom Sadu 1996. god. Upisala je osnovne akademske studije na Fakultetu tehničkih nauka 2015. god. na smeru Softversko inženjerstvo i informacione tehnologije. Diplomirala je 2019. god. i iste godine je upisala master akademske studije na Fakultetu tehničkih nauka, smer Softversko inženjerstvo i informacione tehnologije, na usmerenju Elektronsko poslovanje.
kontakt: bojana.ivanovic@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.738.5:339
DOI: https://doi.org/10.24867/10BE23Mijatov

TESTIRANJE SOFTVERA U OKRUŽENJU SIMULACIJE U REALNOM VREMENU

SOFTWARE TESTING IN REAL-TIME SIMULATION ENVIRONMENT

Vanja Mijatov, Branko Milosavljević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu se opisuju principi testiranja hardver u petlji i kontejnerizacije aplikacije. Opisuje sistem za testiranje softvera u okruženju simulacije u realnom vremenu koristeći specijalizovan sistem za testiranje po principu hardver u petlji pri čemu je softverski deo sistema kontejnerizovan korišćenjem Docker tehnologije.*

Ključne reči: *testiranje softvera, softver u realnom vremenu, hardver u petlji, kontejnerizacija, TyphoonTest, Docker*

Abstract – *Paper describes principles of hardware-in-the-loop testing and application containerization. It describes system for software testing in real-time simulation environment by using specialized system for hardware-in-the-loop testing where software part of the system is containerized by using Docker technology.*

Keywords: *software testing, real-time software, hardware-in-the-loop, containerization, TyphoonTest, Docker*

1. UVOD

Testiranje softvera je proces validacije ponašanja softverskog sistema. Testiranjem se otkrivaju potencijalni bagovi i proverava se pravilnost rada sistema u različitim situacijama, kako u trenucima normalnog rada, tako i u ekstremnim situacijama ili situacijama kada je sistem pod opterećenjem. Takođe, proverava se da li softverski sistem podržava očekivane i potrebne funkcionalnosti.

Kod sistema u realnom vremenu (engl. *real-time system*) testiranje rada je kompleksnije zato što osim što sistem treba da ostvari očekivan rezultat, taj rezultat mora biti ostvaren u određenom vremenskom periodu, odnosno postoje vremenska ograničenja u kojima sistem mora izvršiti posao.

Za testiranje ovakvih sistema najčešće je potrebno obezbediti posebno softversko i hardversko okruženje i posebne alate koji omogućavaju izvršavanje testova.

Simulacija sa hardverom u petlji je tip simulacije u realnom vremenu i način da se putem simulacija testira kompleksan sistem u realnom vremenu. Umesto da se rad gotovog sistema testira tako što se pusti u privremen rad i na taj način rizikuju potencijalne nepredviđene i opasne situacije kao i veliki troškovi usled potencijalnog oštećenja proizvoda ili okoline u kojoj radi, može se simulirati kako bi taj sistem radio u realnim uslovima i na taj način predvideti mnogi potencijalni problemi i nedostaci. Takođe, simulacije mogu pomoći i u fazi razvoja različitih električnih komponenti sistema i na taj način se mogu blagovremeno u fazi razvoja identifikovati problematični delovi sistema i ispraviti, što može predstavljati veliku uštedu finansijskih sredstava i vremena.

Kako je softver koji vrši simuliranje izuzetno kompleksan i treba da kompajlira modele kojima se definišu sistemi, njegovo ponašanje zavisi od konfiguracije i operativnog sistema mašine na kojoj je instaliran i na kojoj se koristi. Drugačija implementacija delova softvera da bi se prilagodio različitim mašinama može biti veliki i skup proces i gotovo je nemoguće podržati pravilan rad na apsolutno svakoj mašini. Stoga, rešenje za korišćenje softvera na različitim platformama bi bilo da se zapakuje u neki kontejner koji bi mogao da se jednako ili približno jednako uspešno koristi na različitim mašinama. Kontejnerizacija različitih softvera sa okruženjem za njihovo pravilno izvršavanje radi lakšeg korišćenja na različitim mašinama bez potrebe da se mašina dodatno konfiguriše postaje standardna praksa poslednjih nekoliko godina i sve više kompanija se odlučuje da svoj softver distribuira i na ovaj način.

Ovaj rad se bavi problematikom testiranja sistema u realnom vremenu i jednim načinom postavke sistema za testiranje ovakvih sistema pomoću specijalizovanog softvera. Takođe, opisuje način kontejnerizovanja specijalizovanog softvera za testiranje i njegovo okruženja, pri čemu se dobija rešenje koje je moguće koristiti na računarima sa različitim operativnim sistemima koji imaju podršku za korišćenje kontejnerizovanih aplikacija. Zatim, opisuje prednosti sistema, probleme koji mogu da se javljaju i njihova rešenja i moguća proširenja sistema. Konačno rešenje je validirano u sistemu koji omogućava automatizovanje testiranje različitih električnih kola i električnih sistema, pri čemu se kao rezultat ovog procesa dobija izveštaj i omogućen je pregled istorijata testiranja, kao i pretraživanje rezultata prethodno izvršenih testova.

2. KONCEPTI TESTIRANJA SISTEMA U REALNOM VREMENU

2.1. Hardver u petlji

Hardver u petlji (engl. *Hardware-in-the-loop,* skraćeno *HIL*) je način ili tehnika za testiranje sistema u realnom vremenu tako što se korišćenjem posebnih hardverskih komponenti simulira okruženje realnog sistema i na taj način testira njegovo ponašanje u određenim situacijama. Vršenjem simulacija u različitim situacijama i scenarijima korišćenja moguće izvršiti veoma veliki broj testova za

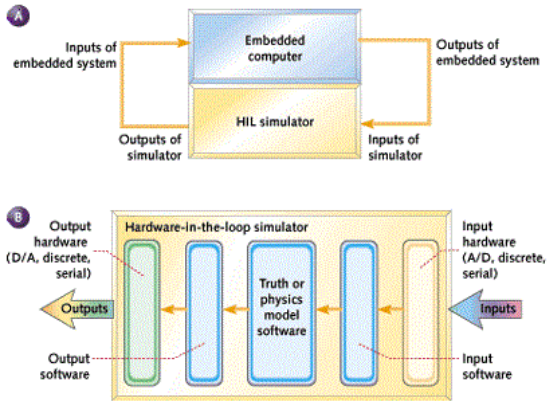
---

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.

kratko vreme bez većih troškova što znači da nije potrebno vršiti fizičko testiranje sistema u realnim uslovima.

Autor rada [1] opisuje hardver u petlji simulator kao uređaj koji je u stanju učiniti da sistem misli da se nalazi u okruženju u realnom svetu slanjem i primanjem signala kao u realnom svetu te na taj način proverava ponašanje sistema u stvarnom okruženju. Autori rada [2] gledaju na hardver u petlji kao simulator koji predstavlja sistem kojim se delovi testiranog podsistema zamene virtuelnim reprezentacijama i na taj način se testira ponašanje ostatka sistema. U radu [3] autor opisuje koncept hardvera u petlji kao metoda koja koristi koncept simuliranja modela procesa i hardvera, pri čemu model simulacije obezbeđuje potrebne signale koji se transformišu i dostavljaju kontrolerima, koji zatim generišu drugi tip signala i na taj način se čini da se hardverske komponente ponašaju što bliže pravim uređajima u realnom sistemu. Sistem za testiranje sistema u realnom vremenu po principu hardvera u petlji koji je korišćen za potrebe ovog rada sastoji se od računara na kom se nalazi softver koji interpretira modele i podržava izvršavanje testova i uređajem za testiranje sistema u realnom vremenu sa kojim je računar povezan i sa kojim može da komunicira.

Slika 1 reprezentuje tipičan primer sistema za testiranje po principu hardver u petlji, odnosno prikazuje komponente simulatora. A segment slike prikazuje računar povezan sa simulatorom hardvera u petlji koji razmenjuju informacije. B segment slike prikazuje unutrašnju strukturu simulatora i tok signala kroz njega. Na osnovu slike i interpretacija ovog tipa sistema može se uvideti simulator dobija kao ulaz različite vrednosti signala od računara i vraća računaru signale kao odgovor, odnosno daje informaciju o odzivu i ponašanju sistema u toku simulacija.



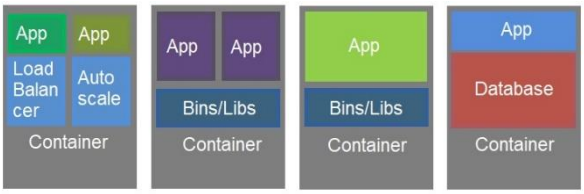
Slika 1. *Primer sistema za testiranje po principu hardver u petlji [1]*

2.2. Kontejnerizacija aplikacije

Kontejnerizacija aplikacije je proces virtuelizacije okruženja i aplikacije u kom se ona izvršava. Na ovaj način dobija se na lakšem pakovanju, instalaciji, distribuciji i korišćenju aplikacije. Krajnji korisnik stoga ne mora da podešava svoj računar i prilagođava okruženje aplikaciji koju želi da koristi, već pomoću softvera koji radi sa kontejnerima koristi aplikaciju zapakovanu u kontejner sa čitavim okruženjem neophodnim za njeno izvršavanje. Kako navodi autor rada [4], kontejnerizacijom se dobija na interoperabilnosti sa ciljem da se distribuira softver. Takođe, ističe jednostavnost kontejnera, portabilnost izvršavanja, mogućnost povezivanja kontejnera i mogućnost razvoja, testiranja i plasiranja na veliki broj servera.

Kako je na jednoj mašini moguće pokrenuti više različitih kontejnera ili više instanci identičnih kontejnera pri čemu se oni nezavisno izvršavaju, preporučljivo je više aplikacija pokrenuti u više različitih kontejnera. U slučaju da ove aplikacije treba da interaguju i razmenjuju informacije ili poruke, potrebno je obezbediti način komunikacije između njih. Slika 2 demonstrira primer jednog sistema zasnovanog na kontejnerima. U različitim kontejnerima se nalaze aplikacije sa različitom namenom koje obavljaju svoje poslove, a po potrebi mogu da međusobno komuniciraju i svoje funkcionalnosti koriste za ostvarenje funkcionalnosti drugih aplikacija u drugim kontejnerima.

U kontejner je moguće spakovati softver za testiranje sistema u realnom vremenu po principu hardver u petlji ako je moguće obezbediti sve potrebne biblioteke i alate od kojih zavisi njegovo izvršavanje. Takođe, potrebno je omogućiti aplikaciji unutar kontejnera da pristupa priključnim mestima računara domaćina da bi mogla da interaguje sa hardverom koji učestvuje u simulacijama. Na ovaj način bi se dobilo simuliranje sistema kao da je softver na tradicionalan način instaliran na računar, a samo korišćenje bi bilo daleko lakše i ne bi trebalo da se izvrši značajnije podešavanje okruženja osim da se obezbedi adekvatna tehnologija za rukovanje kontejnerom.



Slika 2. *Arhitektura sistema zasnovanog na kontejnerima [4]*

3. MODEL SISTEMA

3.1. Komponente sistema

Sistem za testiranje softvera u okruženju simulacije u realnom vremenu se sastoji od platforme za testiranje koja se nalazi na računaru i simulatora koji radi po principu hardvera u petlji koji je povezan sa računarom. Platforma za testiranje omogućava testiranje sistema u realnom vremenu. Zapakovana je u Docker kontejner koji ima pristup simulatoru. Na ovaj način nema potrebe da se značajno podešava okruženje računara, već je dovoljno samo obezbediti tehnologiju za upravljanje kontejnerom te nema potrebe za manuelnom instalacijom platforme.

3.2. Platforma za testiranje softvera

Platforma za testiranje softvera je specijalizovan softver sa velikim brojem alata što čini ovaj softver zahtevnim sa stanovišta performansi računara. U saradnji sa eksternom hardverskom jedinicom, odnosno simulatorom, omogućava simulaciju sistema u realnom vremenu. U skladu sa principom testiranja hardver u petlji, slanjem različitih signala hardverskom modulu od strane aplikacije vrši se simuliranje sistema tako što simulator reaguje na ulazne signale onako kako bi reagovao pravi sistem (softver) u realnom svetu i vraća signalne izlaze kao rezultat koji se evaluiraju i ako su u skladu sa očekivanim vrednostima

definisanim u testu, test će biti uspešan. Na kraju izvršavanja testova, kao rezultat se dobije izveštaj generisan od strane platforme koji demonstrira kako je tekla simulacija, koji su bili koraci u okviru nje i na koji način se simulirani sistem ponašao.

Platforma je kontejnerizovana korišćenjem Docker tehnologije i ima mogućnost korišćenja potpunih resursa računara na kom se pokreće. Takođe, potrebno je da ima pristup priključnim mestima računara da bi mogla aplikacija da komunicira sa simulatorom. Kako se testiranje često automatizuje, u kontejner je za tu potrebu dodata i aplikacija koja automatizuje testiranje u okviru master-agent arhitekture. Ideja master-agent arhitekture (u literaturi ranije poznato i kao *master-slave*) jeste da master definiše zadatak koji agent izvršava.

4. IMPLEMENTACIJA SISTEMA

4.1. Okruženje za testiranje u realnom vremenu

Typhoon HIL kontrolni centar (engl. *Typhoon HIL Control Center*) je aplikacija koja predstavlja skup alata za dizajniranje modela sistema u realnom vremenu, definisanje simulacija rada sistema, testiranje dizajniranog sistema, automatizovano testiranje, pregled izveštaja testiranja itd. Osim softverskih alata koje pruža kontrolni centar, korišćen je i „Typhoon HIL602+“ uređaj koji predstavlja simulator rada električnih sistema u realnom vremenu koji omogućava razvoj, optimizaciju, testiranja i proveru kvaliteta elektronskih sistema.

TyphoonTest je biblioteka koja omogućava izvršavanje testova pisanih u Python programskom jeziku. Može da izvršava testove pisane za testiranje sistema u realnom vremenu dizajniranih pomoću alata kontrolnog centra i generiše datoteke sa rezultatima izvršenih testova.

4.2. Kontejnerizacija aplikacije pomoću Docker tehnologije

Docker je najpopularnija tehnologija za kontejnerizaciju aplikacija u ovom trenutku. Kako razvojni tim ove tehnologije ističe [5], kontejneri u Docker-u su jedinice softvera u koje se zapakuju kod i njegove zavisnosti i koji može da se koristi u različitim okruženjima, a slike kontejnera su mali, nezavisni i izvršivi paketi koji sadrže sve što je potrebno za pravilan rad aplikacije. Na osnovu slike se instanciraju kontejneri i moguće je instancirati više kontejnera na osnovu jedne slike na jednom ili više različitih mašina. Rešenje koje je opisano ovim radom je bazirano na Linux CentOS 8 operativnom sistemu (korišćen kao osnovna slika) i instaliranim većim brojem standardnih biblioteka za Linux operativni sistem.

4.3. Platforma za testiranje softvera

U rešenju je Typhoon HIL kontrolni centar instaliran unutar Docker kontejnera. Kontejener sadrži potrebne sistemske biblioteke i Python koji je kompajliran iz izvornih datoteka, kao i potrebne biblioteke za testiranje. Na osnovu kontejnera je napravljena slika.

Da bi se pokretanjem kontejnera zasnovanim na ovoj slici moglo vršiti testira-nje, potrebno je koristiti Docker *volume* kojim se konte-jneru da pristup datoteci licence i navesti da kontejner može da koristi priključna mesta mašine domaćina da bi komunikacija sa simulatorom bila moguća.

Kako se testiranje učestalo izvršava i testovi se često dorađuju, korišćenje dobijenog rešenja tako što se svaki put testovi manuelno kopiraju unutar kontejnera i ručno pokreće testiranje, a nakon čega se rezultati manuelno izvlače iz kontejnera da bi se mogli analizirati, nije dobro rešenje sa stanovišta zadovoljstva korisnika. Stoga, rešenje je potrebno automatizovati da bi se lakše koristilo i korisnici bili zadovoljniji. Korišćenjem Jenkins aplikacije koja se uz kontrolni centar zapakuje unutar kontejnera, može se postići automatizacija. Rešenje je unapređeno u dva smera tako da su dobijene dve slike koje pružaju različite funkcionalnosti. Prva slika sadrži Jenkins agentsku aplikaciju koja omogućava ponašanje kontejnera kao agenta kojem se daju naredbe od strane mastera. Druga slika osim agentske aplikacije sadrži i celu Jenkins aplikaciju koja omogućava definisanje zadataka koji se izvršavaju na toj mašini. Prilikom pokretanja kontejnera na osnovu ove dve slike potrebno je navesti podatke potrebne za povezivanje kontejnera sa ulogom agenta sa master aplikacijom u vidu promenljivih okruženja.

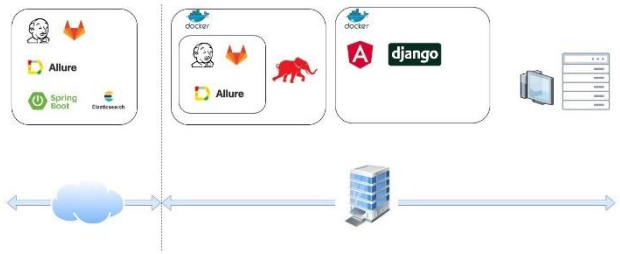
5. EVALUACIJA REZULTATA

Sistem za testiranje softvera u okruženju simulacije u realnom vremenu je pokazao zadovoljavajuće rezultate. Trajanje testiranja se ne razlikuje značajno u odnosu na trajanje pokretanja istih testova na mašini domaćina iz razloga što Docker kontejner ima mogućnost korišćenja svih dostupnih resursa mašine domaćina. Takođe, proširenjem opisanog sistema i automatizovanjem testiranja se dobija velika ušteda vremena za korisnike sistema čime se dobija na njihovom zadovoljstvu rada korišćenog sistema. Distribucija i upotreba dobijenog rešenja u vidu Docker slike je daleko jednostavnije od tradicionalnog pristupa i može da radi na mašinama sa različitim operativnim sistemima koji podržavaju Docker tehnologiju. Kako se koristi Pytest radni okvir za testiranje, moguće je integrisati dodatne priključke napisane u Python programskom jeziku koji mogu raditi uz TyphoonTest i obavljati programski definisane zadatke u okviru testiranja.

Ono što predstavlja manu dobijenog rešenja jeste količina memorije koju zauzima slika usled ogromnog prostora na disku koji zahteva Typhoon HIL kontrolni centar, zbog čega i objavljivanje slika na repozitorijum i preuzimanje slike dugo traje. Kako se za potrebe testiranja ne koristi veći broj funkcionalnosti koje pruža kontrolni centar, particionisanjem softvera na veći broj modula tako da se mogu koristiti isključivo moduli sa potrebnim funkcionalnostima bi se dobilo na smanjenju trajanja izgradnje slike kao i na smanjenju konačne veličine slike.

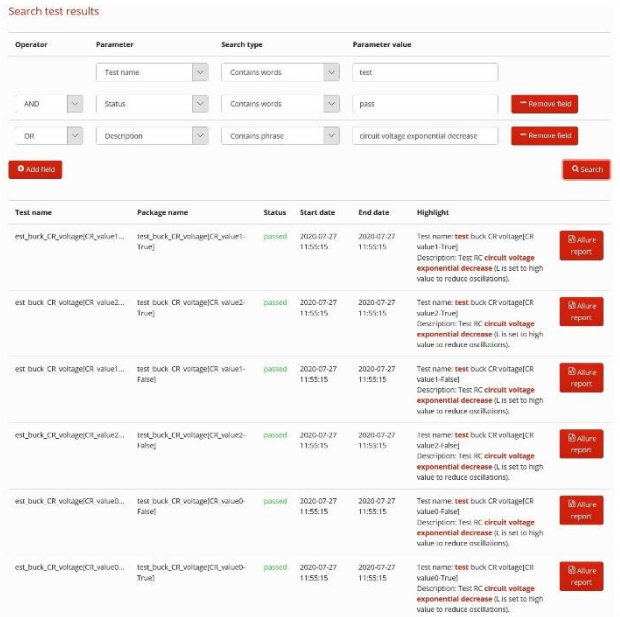
Druga mana rešenja proizilazi iz nedostataka Jenkins aplikacije, odnosno problema sa slanjem i čuvanjem velikih izveštaja i nemogućnošću pretraživanja dobijenih izveštaja. Nakon testiranja, generiše se izveštaj koji se šalje Jenkins master aplikaciji koji u slučaju izvršavanja velikog broja testova može zauzeti značajnu količinu memorije na disku, a slanje izveštaja preko mreže između agenta i mastera može dugo da traje zbog njegove veličine. Primljeni izveštaj Jenkins aplikacija čuva i omogućava pregledanje što nije problem za manji broj izveštaja. U slučaju čestih testiranja velikog obima, Jenkins aplikacija ne odgovara u potpunosti potrebama korisnika.

Sa ciljem rešavanja drugog problema koji prouzrokuju nedostaci Jenkins aplikacije, sistem za automatizovano testiranje je proširen aplikacijom čija je uloga da čuva testove i omogućava njihovo pregledanje i pretraživanje, a koja se nalazi u oblaku. U drugom kontejneru koji se pokreće na istoj mašini kao i kontejner koji vrši testiranje, nalazi se Django aplikacija koja nadgleda testiranje. Rezultate testiranja ova aplikacija šalje Spring Boot aplikaciji koja se nalazi u oblaku i koja čuva rezultate testiranja i omogućava njihovo pretvaranje u izveštaj čime se rešava jedna od mana Jenkins aplikacije. Pored Django aplikacije, pomenuti kontejner sadrži i Angular aplikaciju kojom je definisan korisnički interfejs. Spring Boot aplikacija koja se nalazi u oblaku komunicira sa Elasticsearch platformom koja se takođe nalazi u oblaku i koja omogućava čuvanje izvršenih testova koji se predstavljaju u vidu zasebnih dokumenata i pretraživanje po više različitih parametara. Model proširenog sistema prikazuje Slika 3.



Slika 3. *Sistem za automatizovano testiranje i evaluaciju rezultata testiranja*

Slika 4 prikazuje formu za definisanje upita za pretragu izvršenih testova i tabelu sa rezultatima pretrage. Za svaki parametar upita nakon prvog parametra moguće je definisati logički operator i/ili u odnosu na prethodni parametar ili skup parametara. Tabela rezultata prikazuje maksimalno tri stotine najbolje rangiranih rezultata te u slučaju da korisnik traži izvršeni test po parametru koji se često pojavljuje u polju koje je izabrano kao parametar pretrage ili je za parametar pretrage unesena reč visoke frekvencije, potrebno je da se preciznije definiše upit kako bi se pronašao rezultat koji je korisnik priželjkivao.



Slika 4. *Stranica za pretragu izvršenih testova*

6. ZAKLJUČAK

U ovom radu dat je opis sistema za testiranje softvera u okruženju simulacije u realnom vremenu. Opisane su komponente sistema, principi u skladu sa kojima one rade i osobine rešenja sa njegovim proširenjima.

Sistem za testiranje softvera u okruženju simulacije u realnom vremenu omogućava testiranje električnih sistema i dozvoljava integrisanje sa drugim alatima radi promene standardnog ponašanja sistema kao i davanje višestruke uloge sistemu. Specijalizovani softver koji omogućava testiranje je kontejnerizovan korišćenjem Docker tehnologije tako da su olakšani korišćenje i distribucija softvera. Testiranje se može automatizovati tako što se vrši integracija sa Jenkins aplikacijom koja se takođe nalazi u kontejneru i koja zadaje komande za pokretanje testiranja koje izvršava specijalizovani softver, a sama automatizacija radi prema master-agent arhitekturi.

Moguće je proširiti sistem sa drugim aplikacijama koje omogućavaju evaluaciju i pretraživanje rezultata testiranja. Dalje istraživanje bi bilo usmereno u pravcu integrisanja sistema za testiranje sa nekim od popularnih oblak kompjuting veb servisa koji nudi veliki izbor alata što bi smanjilo potrebu za korišćenjem dosta različitih alata i tehnologija za potrebe automatizacije testiranja i evaluacije dobijenih rezultata izvršenih testova.

7. LITERATURA

[1] Gomez, Martin. "Hardware-in-the-loop simulation." *Embedded Systems Programming* 14.13 (2001): 38-49.

[2] Fathy, Hosam K., et al. "Review of hardware-in-the-loop simulation and its prospects in the automotive area." *Modeling and simulation for military applications*. Vol. 6228. International Society for Optics and Photonics, 2006.

[3] Grega, Wojciech. "Hardware-in-the-loop simulation and its application in control education." *FIE'99 Frontiers in Education. 29th Annual Frontiers in Education Conference. Designing the Future of Science and Engineering Education. Conference Proceedings (IEEE Cat. No. 99CH37011*. Vol. 2. IEEE, 1999.

[4] Pahl, Claus. "Containerization and the paas cloud." *IEEE Cloud Computing* 2.3 (2015): 24-31.

[5] https://www.docker.com/resources/what-container (pristupljeno u septembru 2020.)

Kratka biografija:

Vanja Mijatov rođen je u Novom Sadu 1996. god. 2015. god. se upisao na osnovne akademske studije na Fakultetu tehničkih nauka u Novom Sadu na smer Softversko inženjerstvo i informacione tehnologije. Diplomirao je na osnovnim akademskim studijama 2019. godine i iste godine je upisao master akademske studije na Fakultetu tehničkih nauka u Novom Sadu, smer Softversko inženjerstvo i informacione tehnologije, na usmerenju Elektronsko poslovanje.
kontakt: vanja.mijatov@uns.ac.rs

Зборник радова Факултета техничких наука, Нови Сад

УДК: 004.9
ДОИ: https://doi.org/10.24867/10BE24Mihajlovic

КРЕИРАЊЕ СТРУКТУРЕ ЗА ЕФИКАСНУ МАНИПУЛАЦИЈУ ДИГИТАЛНИМ РЕСУРСИМА

CREATION OF A STRUCTURE FOR EFFICIENT MANIPULATION OF DIGITAL RESOURCES

Ана Михајловић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Манипулација дигиталним ресурсима представља веома битан аспект успешног пословања различитих пословних ентитета. Због тога, тема овог рада јесте анализа основних концепата и компоненти манипулације дигиталним ресурсима. Такође, рад укључује и предлог универзалне структуре за подршку рада са различитим дигиталним ресурсима, заједно са смерницама за њено креирање за конкретне пословне ентитете.*

Кључне речи: *дигитални ресурси, системи за управљање ресурсима, структура за манипулацију ресурсима, доменски сервиси*

Abstract – *Manipulation of digital resources is a very important aspect of successful business of various business entities. Therefore, the topic of this paper is the analysis of basic concepts and components of digital resource manipulation. Additionally, a proposal for a universal structure to support work with different digital resources is included, along with guidelines for its creation for specific business entities.*

Keywords: *digital resources, resource management systems, structure for resource manipulation, domain services*

1. УВОД

Свет у којем данас живимо налази се у процесу транзиције од света којим управљају физички објекти, ка свету који се потпуно ослања на њихове дигиталне репрезентације. Свеприсутна дигитализација у великој мери утиче на човека и његову егзистенцију у постојећем свету. Како би опстао у свету у којем доминирају дигиталне информације, човек мора да развије сопствени начин прихватања, обраде и употребе ових информација.

Упоредо са променама које дигитализација уводи у човеково схватање света, мења се и концепт модерне економије. У тржишним условима дигиталног доба, успех сваког предузећа мери се његовом способношћу да трансформише своје дигиталне ресурсе у готове производе који се пласирају корисницима широм света. Са друге стране, вредност сваког ресурса сразмерна је степену његове искористивости, која даље зависи од лакоће проналаска и обраде ресурса за актуелне потребе предузећа. Дакле, може се закључити да је опстанак предузећа на тржишту индиректно зависан од начина уређења и структуре у којој се дигитални ресурси похрањују.

Тема овог рада је управо дефинисање одговарајуће структуре за ефикасну манипулацију дигиталним ресурсима која представља темељ успеха на тржишту. У наставку рада приказане су неопходне теоријске основе везане за рад са дигиталним ресурсима, дат је приказ водећих система за управљање дигиталним ресурсима на тржишту, као и смернице за креирање структуре прилагођене потенцијалним захтевима великог броја пословних ентитета.

2. ДИГИТАЛНИ РЕСУРСИ

У уводу рада укратко је образложена потреба модерних предузећа за поседовањем структуре за управљање дигиталним ресурсима. Међутим, како би се концепт уређивања дигиталних ресурса у потпуности разумео, потребно је објаснити и основне појмове везане за ову област информационих технологија. На самом почетку, потребно је недвосмислено дефинисати појам дигиталног ресурса.

Дигитални ресурси своје корене налазе у материјалним ресурсима, који су стари колико и први људи. Развојем технике и технологије јавила се могућност, али и потреба за креирањем дигиталних репрезентација физички опипљивих ресурса. На овај начин, олакшана је не само њихова употреба и дистрибуција, већ је и продужен животни век постојећих ресурса. Међутим, не може се свака дигитална датотека сматрати дигиталним ресурсом. Појам дигитални ресурс може се дефинисати као „сваки текст или медиј који је у бинарном формату и укључује право на коришћење; дигиталне датотеке које не укључују ово право не сматрају се дигиталном имовином“ [1].

Дакле, главна одлика дигиталног ресурса која га разликује од дигиталне датотеке јесте то што сваки ресурс поред бинарног садржаја, мора да укључује и право на коришћење, али и да поседује одређену пословну вредност за пословни ентитет ком припада [2]. Дигиталним ресурсима се подједнако могу сматрати текстуалне датотеке, *PowerPoint* презента- ције, фотографије, видео снимци, 3Д објекти, итд., под условом да задовољавају претходно наведене критеријуме који их разликују од обичних датотека.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Бранко Милосављевић, ред. проф.

2.1. Системи за управљање ресурсима

Како би се могла остварити пословна вредност дигиталних ресурса, они морају бити ускладиштени и уређени на начин који омогућава интуитиван и брз приступ жељеном ресурсу. Последично, као одговор на поменути захтев развијени су различити системи за управљање ресурсима. Овај тип система дефинише се као „систем управљања за управљање имовином чија је функција успостављање политике управљања имовином и циљева управљања имовином" [3], где су политика и циљеви прописани од стране конкретних пословних ентитета који употребљавају дати систем. Као и сами ресурси, и ови системи су се мењали заједно са развојем технологије и интернета. Први системи за управљање били су намењени организацији материјалних ресурса, а како се мењало поимање ресурса, мењала се и структура ових система. На овај начин, од кабинета за складиштење папирних докумената, који представљају једне од првих система за управљање, развојем складишних технологија и интернета, настали су данашњи софтверски системи за управљање дигиталним ресурсима.

Предности које употреба ових система са собом повлачи су огромне, а најбоље их илуструје следећих десет критеријума које сваки систем за управљање ресурсима мора да задовољи [4]:

1. **Унос података**
   Могућност уношења различитих типова ресурса и њихових метаподатака у систем
2. **Безбедност података**
   Спречавање злоупотребе и преране објаве ресурса и њихових метаподатака
3. **Складиштење података**
   Складиштење ресурса и њихових метаподатака без обзира на њихов тип
4. **Трансформација дигиталног садржаја**
   Претварање дигиталних ресурса из једног облика и формата у други
5. **Обогаћивање ресурса метаподацима**
   Измена и додавање метаподатака у циљу обогаћивања ресурса
6. **Повезивање ресурса**
   Формирање и праћење веза између различитих ресурса, али и између оригинала и различитих верзија истог ресурса
7. **(Аутоматизована) обрада ресурса**
   Обрада ресурса путем аутмоатизованих токова
8. **Проналажење ресурса**
   Претрага и проналазак ресурса на основу задатих критеријума унутар система
9. **Приказивање ресурса**
   Прегледање ресурса унутар система, без преузимања на локалну машину
10. **Објављивање ресурса**
    Дељење и дистрибуција ресурса ка спољашњем свету

У структури која омогућава ефикасну манипулацију дигиталним ресурсима, централно место заузима управо систем за управљање ресурсима, и због тога је неопходно обратити посебну пажњу на одабир оног система који се најбоље уклапа у потребе и могућности ентитета за који се структура дефинише. Као и сваки софтверски систем, системи за управљање дигиталним ресурсима могу се поделити на основу њиховог порекла и начина имплементације у следеће три категорије:

- Комерцијални системи
- Системи отвореног кода
- Самостално развијени системи

Свака категорија имплицира одређене предности, али и мане, па је због тога потребно добро разумети очекивања које пословни ентитет има спрам увођења новог система у своју структуру.

Област развоја система за управљање је изузетно развијена, те се на тржишту може наћи велики број комерцијалних система који на задовољавајући начин могу да испуне већину уопштених захтева када је реч о манипулацији дигиталним ресурсима. Ипак, упркос томе што ова група система иза себе има компанију која гарантује за њихов квалитет и пружа услуге увођења и одржавања, ова група софтвера често долази са веома високим ценама, а свако прилагођавање система специфичним потребама ентитета повлачи нове трошкове. У наставку рада приложена је табела актуелних лидера на тржишту система за дигитално управљање. Међутим, како овај тип софтвера директно зависи од степена развијености доступних дигиталних технологија, ови системи се често унапређују па је препоручљиво да се пре коначног одабира жељеног система провери актуелно стање на тржишту.

Табела 1. *Водећи комерцијални системи за управљање*

Aprimo	https://www.aprimo.com/
Adobe Experience Manager Assets	https://www.adobe.com/experience-cloud/topics/digital-asset-management.html
Bynder	https://www.bynder.com/en/
MediaBeacon	https://www.mediabeacon.com/en
OpenText	https://www.opentext.com/
Widen	https://www.widen.com/

Осим комерцијалним системом, централно место у структури за управљање дигиталним ресурсима могуће је попунити и системом отвореног кода, а систем који се намеће као један од бољих представника ове групе на тржишту јесте *Nuxeo* платформа. Ова група система углавном долази по нижим (или чак и потпуно бесплатним) ценама и нуди већи степен прилагодљивости ентитету који га купује. Ипак, употреба ове групе софтвера захтева поседовање одговарајућег техничког знања и изискује напор за самостално одржавање и увођење система у рад.

Последња опција када је реч о одабиру система за управљање јесте самостални развој система, а као основа за његову имплементацију препоручује се употреба *Microsoft Azure* или *Amazon* веб сервиса. Иако је самостални развој једина гаранција за потпуну прилагођеност система ентитету који ће га користити,

неопходно је имати на уму да су ови системи изузетно комплексни, те осим времена, за имплементацију захтевају знање из неколико различитих области информационих технологија.

3. ФОРМУЛАЦИЈА КОРИСНИЧКИХ ЗАХТЕВА

Како би се могао исправно одабрати одговарајући систем за управљање ресурсима, осим познавања стања на тржишту, потребно је разумети и јасно дефинисати захтеве и очекивања које ентитет има спрам будућег система за управљање. То даље имплицира да основ за дефиницију прикладне структуре за манипулацију ресурсима представљају управо идентификовани кориснички захтеви.

Пре почетка процеса прикупљања захтева, неопходно је извршити детаљну анализу успостављених пословних процеса унутар предузећа. На овај начин, могуће је издвојити скуп корисника који се највише ослањају на постојеће дигиталне ресурсе, и који последично чине фокусне групе корисника за дефинисање нове структуре. Са представником сваке издвојене групе потребно је спровести интервјуе и радионице како би се добиле информације о начину рада са ресурсима и сарадње унутар групе, типовима и величинама ресурса са којима се ради, евентуалним проблемима у актуелном модусу рада, као и начину комуникације групе са спољашњим сарадницима.

Осим разговора са запосленима, на основу анализе процеса и постојеће структуре за манипулацију ресурсима (уколико она постоји) потребно је извести закључке о актелним проблемима које је у будућности неопходно решити. Такође, у фази прикупљања захтева потребно је сакупити и податке о количинама и величинама свих типова ресурса који ће бити смештени у будућу структуру. Системи за управљање ресурсима, описани у претходном поглављу, често се разликују у степену прилагођености одређеним типовима ресурса, те је у циљу одабира прикладног система од кључног значаја дефинисање типова ресурса релевантних за пословање ентитета и операција које над њима треба да се извршавају.

4. ДЕФИНИСАЊЕ СТРУКТУРЕ

Након формулације корисничких захтева и упознавања са могућностима система за управљање ресурсима, могуће је започети дефинисање нове структуре за ефикасан рад са релевантним дигиталним ресурсима.

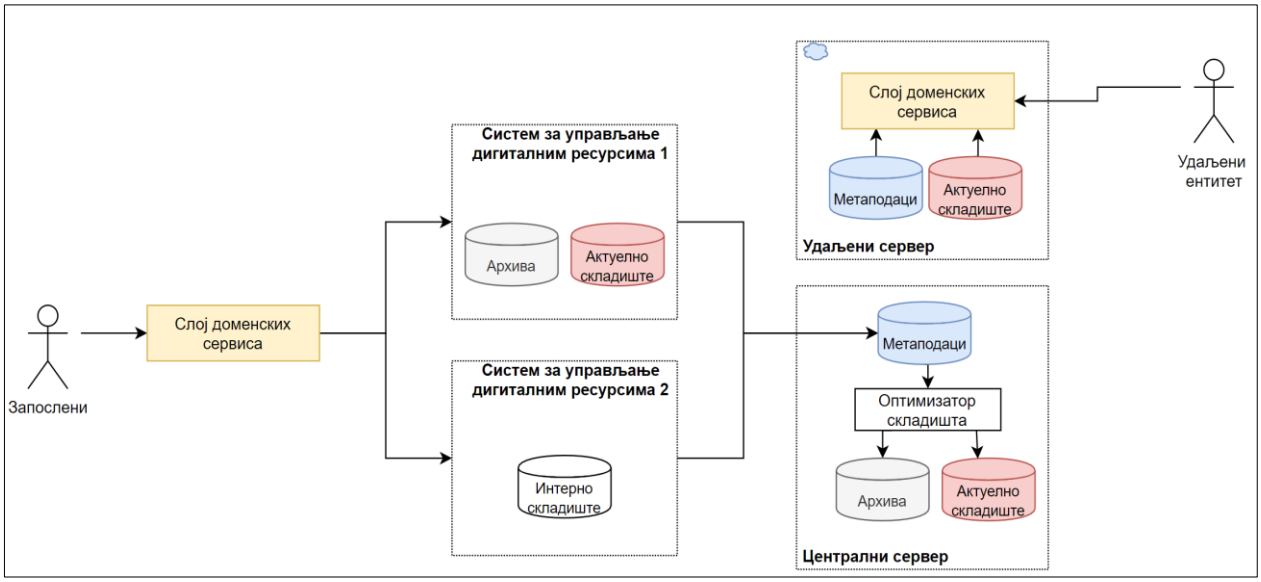
4.1. Приказ структуре

Као што је већ неколико пута напоменуто, пре почетка дефинисања структуре, потребно је пронаћи или развити одговарајући систем за управљање ресурсима. Систем се бира на основу степена његове прилагођености раду са типовима дигиталних ресурса који су означени као релевантни за пословање конкретног ентитета. Препоручљиво је увођење једног система који подржава рад са свим релевантним типовима, али уколико није могуће наћи систем који задовољава све критеријуме, онда је потребно одабрати више система, који заједно могу да одговоре на постављене захтеве.

Након одабира система, потребно је обратити пажњу на физички начин складиштења ресурса. Мали број данашњих тржишних система подржава концепт топлог и хладног складиштења података, а како се ови системи често наплаћују спрам количине ускладиштених ресурса и учесталости приступа истим, добро дефинисана структура физичких складишта у великој мери утиче на укупне трошкове пословног ентитета. Концепт топлог и хладног складиштења није ништа друго до раздвајање ресурса у два одвојена складишта са различитим складишним ценама на основу учесталости приступа ресурсима. Уколико систем за управљање нема уграђен концепт пребацивања застарелих ресурса у складиште са нижим складишним ценама, препорука је да се такав систем интерно изгради и интегрише у нову структуру. На слици 1 грубо је приказан претходни концепт, а у структуру је укључен и оптимизатор складишта који представља компоненту задужену за одржавање складишта и премештање ресурса из топлог у хладно складиште (хладно складиште названо је архива на слици 1) на основу предефинисаних критеријума.

Следећи аспект управљања ресурсима који је потребно узети у обзир приликом дефинисања структуре јесте начин чувања метаподатака ускладиштених ресурса. Метаподаци садрже различите информације о постојећим ресурсима, укључујући и податке о локацији на којој се ресурс налази. Издвајање метаподатака у одвојено складиште утиче на брзину добављања ресурса, и оно мора у сваком тренутку да пружа тачну слику система и локација. Као складиште за метаподатке може се искористити *SQL* база података, *Azure*, *Amazon* веб сервиси или *Elasticsearch* индекс, у зависности од установљених потреба и захтева пословног ентитета.

Приликом дефинисања нове структуре потребно је размотрити и начин добављања ресурса од стране удаљених клијената или пословних сарадника. Уколико се уочи потреба за преузимањем ресурса од стране удаљених ентитета, препоручљиво је увођење удаљеног сервера у нову структуру. Пожељно је да удаљени сервер буде смештен на облаку, како би се обезбедило најкраће могуће време добављања ресурса, без обзира на географску удаљеност субјекта од централног сервера задуженог за управљање дигиталним ресурсима. На удаљеном серверу може се пронаћи топло складиште података намењено чувању искључиво дигиталних ресурса релевантних за дато географско подручје сервера. Осим топлог складишта, потребно је подесити и базу метаподатака на удаљеном серверу која је идентична централној бази метаподатака и која служи за брз проналазак жељеног ресурса на централном серверу уколико га није могуће пронаћи у локалном топлом складишту. Како би се илустровала претходно описана структура у рад је укључена следећа слика:



Слика 1. *Дијаграм предложене структуре*

На крају, потребно је споменути и слој доменских сервиса који служи као примарна тачка приступа новој структури. Улога доменских сервиса је веома битна, јер је помоћу њих могуће прилагодити нову структуру специфичностима свих група корисника унутар пословног ентитета, уводи се униформност при раду са дигиталним ресурсима и продужава животни век дефинисане структуре. Међутим, да би рад са дигиталним ресурсима био истински униформан, сервиси би требало да буду дизајнирани тако да омогућавају потпуну независност од типа ресурса са којим се ради, као и од конкретног позадинског система у који се ресурс смешта. На овај начин олакшава се будуће уношење промена у начин рада сервиса, додавање или уклањање подржаних типова ресурса, као и измена позадинских система укључених у структуру. Као идеалан основ за имплементацију доменских сервиса издваја се бриц софтверски образац који омогућава „раздвајање апстракције од имплементације тако да се могу независно мењати" [5]. У контексту овог рада, апстракцију представљају класе задужене за рад са различитим ресурсима, док имплементацију чине конекторске класе за позадинске системе.

5. ЗАКЉУЧАК

Тржишна вредност модерног предузећа мери се његовом способношћу да ефикасно експлоатише поседоване дигиталне ресурсе. Међутим, да би то било могуће, унутар предузећа мора да постоји стабилна и прикладна структура која пружа подршку раду са ресурсима. Паралелно са анализом пословних процеса предузећа у циљу прикупљања захтева, неопходно је истражити актуелну ситуацију на тржишту софтверских система за управљање дигиталним ресурсима који представљају срж поменуте структуре. Након тога, потребно је одабрати системе који испуњавају прикупљене захтеве и око њих изградити одговарајућу архитектуру. Приликом дефинисања структуре потребно је посебну пажњу посветити физичком начину складиштења дигиталних ресурса, као и механизму за архивирање застарелих датотека. Уколико пословни ентитет има потребу за сарадњу са удаљеним субјектима, потребно је осмислити структуру која би им омогућила брзо добављање жељених ресурса са централног сервера. На крају, како би се осигурала дуга и униформна употреба креиране структуре пожељно је дефинисати слој доменских сервиса око ње, који би служили као посредник између дефинисане структуре и спољашњег света.

6. ЛИТЕРАТУРА

[1] „Digital Asset", *Techopedia*, 23. јануар 2017. [Online]. Доступно: https://www.techopedia.com/definition/23367/digital-asset. [Приступљено: јануар 2020].

[2] S. Diamond и E. Kolovitz, *Digital Asset Management for Dummies, Bynder Special Edition*, John Wiley & Sons, Ltd., The Atrium, Southern Gate, 2017. [E-book] Доступно: https://www.bynder.com/en/e-books /dam-for-dummies/

[3] *Asset management — Overview, principles, and terminology*, ISO Standard No. 55000:2014, 2014.

[4] IQ Equity, „10 Core DAM Characteristics", *IQ Equity* , 2014. [Online]. Доступно: https://www.iqequity.co.uk/products/10-core-dam-characteristics/. [Приступље но: фебруар 2020].

[5] Дејановић И., „Софтверски обрасци и компоненте" . [Online] Доступно: http://www.igordejanovic.net/courses/sok/03-strukturalni.[Приступљено: март 2020].

Кратка биографија:

Ана Михајловић рођена је у Новом Саду 1994. године. По завршетку гимназије „Светозар Марковић" уписује основне академске студије рачунарства и аутоматике на Факултету техничких наука у Новом Саду. Диплому стиче 2017. године, након чега своје образовање наставља на мастер академским студијама на истом одсеку Факултета техничких наука

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE25Kalabic

УПОРЕДНА АНАЛИЗА РАЗВОЈА SERVERLESS И CONTAINERIZED АПЛИКАЦИЈА НА AMAZON ПЛАТФОРМИ

COMPARATIVE ANALYSIS OF SERVERLESS AND CONTAINERIZED APPLICATION DEVELOPMENT ON AMAZON PLATFORM

Јелена Калабић, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У раду је пружена упоредна анализа развоја апликација са различитим архитектурама (serverless и containerized-microservice) на Amazon платформи. Такође, дат је преглед релевантних концепата и опис архитектура две апликације.*

Кључне речи: *Serverless, containerized, микросервис, AWS сервиси, Amazon*

Abstract – *This paper provides a comparative analysis of serverless and containerized-microservice application development on Amazon platform. An overview of relevant concepts and a description of two application architectures are given.*

Keywords: *Serverless, containerized, microservice, AWS services, Amazon*

1. УВОД

Архитектура софтвера одувек је била изузетно важна за развој софтвера и представља један од фактора који у значајној мери утичу на успех софтверског производа на тржишту. Током времена разни типови архитектура су долазили до изражаја и уживали велику популарност (попут монолитне, микросервис и *serverless* архитектуре).

У последњој деценији посебну популарност доживели су концепти попут контејнера (*container*) и *cloud computing*-а и чини се да је без њих тешко замислити савремени развој софтвера. Најпознатији и највећи *cloud* провајдери на тржишту тренутно су: *Amazon Web Services* (AWS), *Microsoft Azure* и *Google Cloud*. Могућности су велике и посебан је изазов одлучити се које технологије и који приступ су најпогоднији за развој одређеног производа.

Да ли развијати апликацију независну од *cloud* провајдера која може да се извршава практично било где? Или у потпуности користити сервисе које *cloud* провајдери нуде и фокусирати се само на развој пословне логике?

У наставку рада дат је преглед релевантних концепата, приказане су архитектуре две апликације и дато је поређење *containerized-microservice* и *serverless* приступа.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Бранко Милосављевић, ред. проф.

2. ПРЕГЛЕД РЕЛЕВАНТНИХ КОНЦЕПАТА

Претходних десетак година обележила је појава *cloud computing*-а који је значајно променио развој и одржавање софтверских производа. Шта је заправо ***cloud computing***? „*Cloud computing* је испорука захтеваних ИТ ресурса путем интернета. Модел плаћања заснива се на томе да се ресурси плаћају само онда када су заиста у употреби. Уместо да купујете, поседујете и одржавате физичке центре података (*data center*) и сервере, можете по потреби приступати технолошким услугама, попут рачунарске снаге (*computing power*), складиштења података (*storage*) и база података (*database*), које пружају *cloud* провајдери попут AWS-а“ [1]. Један од највећих играча на тржишту *cloud computing*-а јесте **AWS**. „AWS представља платформу веб сервиса који нуде решења за рачунарску снагу, складиштење и умрежавање на различитим нивоима апстракције. На пример, складиштење података може бити на нивоу блока (*block-level storage)* или високо дистрибуирано складиште објеката (*distributed object storage*)“ [2].

Платни модел заснива се на принципу плаћања по употреби (*pay-per-use*). Са појавом *cloud*-а поједини типови софтверских архитектура дошли су до изражаја, а у наставку ће бити описана два типа која су значајна за тему овог рада - serverless архитектура и контејнерска микросервис (*containerized-microservice*) архитектура. ***Serverless* архитектуру** тешко је дефинисати, али оно око чега се сви слажу јесте да *serverless* не значи да не постоје сервери на којима се извршава код већ да програмери/корисници не морају да брину о њима (не морају да покрећу и одржавају виртуелне или физичке сервере). *Serverless* пружа прилику програмерима да се фокусирају на писање кода уместо на одржавање сервера и база података.

У *serverless* свету *cloud* провајдери попут AWS-а динамички управљају доделом и дистрибуцијом ресурса. **Микросервиси** (*microservice*) представљају архитектонски приступ развоју софтвера заснован на изради апликације као колекције малих, независних сервиса.

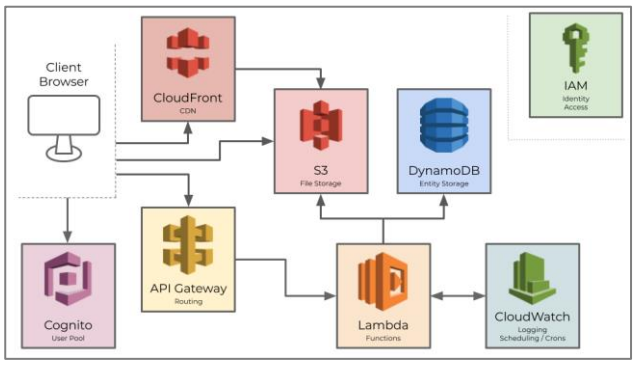
Са појавом и развојем *cloud*-а, тачније са еволуцијом контејнерских (*container*) архитектура појавио се и концепт изворних *cloud* микросервиса (*cloud native microservices*). „**Контејнер** представља стандардну јединицу софтвера која пакује код и све потребне зависности (*dependencies*) што омогућава брзо и

поуздано покретање апликације у различитим рачунарским окружењима“ [3]. Микросервиси се често испоручују у једном или више контејнера.

С обзиром да су контејнери изолована окружења, могу се користити за *deploy*-овање микросервиса брзо и безбедно без обзира на програмски језик који се користи за развој микросервиса.

3. ОПИС АРХИТЕКТУРА АПЛИКАЦИЈА

На слици 1. приказана је *serverless* архитектура апликације за складиштење и управљање корисничким фотографијама.



Слика 1. *Serverless архитектура апликације*

На слици 1. види се **IAM** компонента неопходна да би један сервис приступао другом и да би се дефинисала одређена права приступа те је она имплицитно повезана са свим компонентама система и те везе су изостављене зарад једноставности саме слике.

Клијентска апликација (***Client***) представља SPA (*Single Page Application*) апликацију развијену у *JavaScript* програмском језику уз употребу *Vue.js framework*-а. Клијентска апликација ускладиштена (*hosted*) је у **S3** складишту.

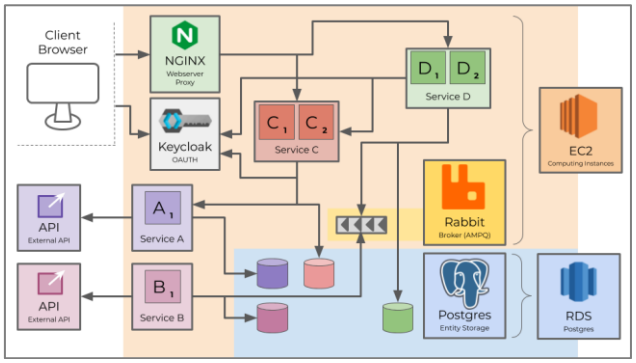
Статички садржај клијентске апликације испоручује се корисницима путем ***CloudFront***-а. *CloudFront*-ова главна улога је брже испоручивање садржаја крајњим корисницима. S3 се осим за складиштење клијентске апликације користи и за складиштење корисничких фотографија и њихових умањених верзија.

За регистровање, аутентификацију и ауторизацију корисника задужена је ***Cognito*** компонента. Сви захтеви ка серверском делу апликације рутирају се кроз API *Gateway* компоненту што резултује активирањем одређене ламбда функције (***Lambda***). Ламбда функције имплементиране су у *framework*-у Node.js-а – *Express*-у.

DynamoDB представља базу података у којој се складиште подаци о фотографијама и албумима.

Сви записи (*logs*) о активности ламбда функције и евентуалним проблемима који се могу јавити завршавају у компоненти ***CloudWatch***. Слика 2. приказује *containerized-microservice* архитектуру апликације која олакшава свакодневни рад запосленима у компанији која се бави дигиталним оглашавањем/рекламирањем (*digital advertising*).

На слици 2. могуће је уочити микросервисе, њихове међусобне везе као и везе ка екстерним платформама. Клијентска апликација (***Client***)



Слика 2. *Containerized-microservice архитектура апликације*

развијена је у *JavaScript* програмском језику уз употребу *Vue.js framework*-а. Као што је приказано на слици, клијентска апликација има две директне везе – са ***Keycloak*** сервисом и **NGINX** веб сервером. *Keycloak* сервис задужен је за аутентификацију корисника приликом пријављивања у систем. **NGINX** веб сервер задужен је за испоручивање клијентске апликације и за прослеђивање (*proxying*) свих захтева ка сервисима. NGINX не приступа свим сервисима из безбедносних разлога.

Сви сервиси који су обухваћени розе бојом у продукционом окружењу извршавају се унутар контејнера, при чему сервиси могу имати једну (нпр. сервис A) или више покренутих инстанци (нпр. сервис D). Сви контејнери, унутар којих се извршавају микросервиси, покренути су унутар AWS **EC2** инстанци. Као систем за управљање базама података користи се *Amazon* **RDS**. Сваки микросервис има сопствену базу података, али уколико се сервис извршава у више контејнера они сви деле исту базу података. Већина микросервиса имплементирана је у *Java* програмском језику уз употребу *Spring Boot framework*-а док су поједини сервиси имплементирани *JavaScript* програском језику уз употребу *Express framework*-а. За контејнере се користи *Docker*, брокер за размену порука је *RabbitMQ* а *PostgreSQL* користи се као база података.

4. ПОРЕЂЕЊЕ АРХИТЕКТУРА

У претходном поглављу описане су две апликације, са две различите архитектуре и оне су предмет поређења у овом поглављу.

4.1. Сложеност развоја

За поређење сложеност развоја изабрани су следећи параметри поређења: време и напор потребни да би се савладала/научила употреба технологија и сервиса неопходних за развој апликације, слобода при избору технологија, контрола над сервером и извршним оружењем, конфигурација сервиса, време потребно за *deployment*, комплексност тестирања и *debug*-овања, могућност креирања локалног (*offline*) извршног окружења.

За *serverless* приступ приликом развоја апликације морамо бити спремни да уложимо време и труд за изучавање *cloud* сервиса које желимо да користимо да бисмо били у стању да их исправно конфигуришемо и омогућимо да међусобно комуницирају као и да научимо да користимо *cloud* алате и корисничке интерфејсе. Додатнуману представља чињеница да стечено знање углавном није применљиво приликом употребе сервиса другог *cloud* провајдера.

Уколико се одлучимо за развој апликације са контејнерском микросервис архитектуром, највероватније ћемо се одлучити за технологије и сервисе које већ довољно добро познајемо што значи да је потреба за усвајањем нових знања мања него код *serverless* приступа. Када је реч о слободи приликом избора технологије, ту је дефинитивно предност на страни контејнерског микросервис приступа јер можемо да изаберемо било коју технологију/-програмски језик. *Serverless* приступ са собом носи одређена ограничења приликом избора технологије јер *cloud* провајдери подржавају ограничен скуп технологија.

Серверска страна *cloud* сервиса тј. комплетна инфраструктура на којој се заснива апликација са *serverless* архитектуром и окружење у коме се извршава су под контролом *cloud* провајдера и оно што сам корисник може да конфигурише је прилично ограничено.

С друге стране, употребом контејнера ми смо ти који су у потпуности задужени за избор оперативног система, библиотека, технологија, за бригу о потребним ресурсима, за дефинисање конфигурационих фајлова (попут *Dockerfile*-а) и слично. Међутим, када је реч о комплексности конфигурације сервиса предност је на страни *serverless* приступа.

Када је реч о времену потребном за *deployment* апликације, иако се може рећи да су оба приступа мање-више слична, блага предност ипак се може дати *serverless* приступу (јер контејнерима иницијално треба више времена за конфигурисање системских подешавања, библиотка и слично).

Апликације са *serverless* архитектуром тешко је тестирати јер је серверско окружење тешко реплицирати у локалном окружењу. С друге стране, контејнери се извршавају исто без обзира у ком окружењу (продукционом, тест) су покренути. И када је у питању *debug*-овање апликације предност је на страни контејнерске микросервис архитектуре.

Оно што може бити од значаја за програмере при изршавању њихових задатака приликом развоја апликације јесте могућност покретања свих сервиса неопходних за рад целе апликације локално (идеално без потребе да имају интернет конекцију - *offline*).

Чињеница да *serverless* приступ у потпуности почива на употреби *cloud* сервиса чини практично немогућим да се апликација у потпуности покрене локално.

С друге стране, употребом контејнера могуће је локално покренути све неопходне сервисе.

4.2. Сложеност инсталације и одржавања

Пре самог поређења апликације са *serverless* архитектуром и апликације са контејнерском микросервис архитектуром која се извршава на „голом“ (*bare*) серверу са аспекта сложености инсталације и одржавања, побројан је део активности које су обухваћене процесима инсталације и одржавања даби се стекао утисак о њиховој комплексности.

Активности које је обавезно спровести су следеће: активности у вези са постојећим захевима када је реч о саобраћају и доступности, подешавање мрежне инфраструктуре, подешавања рачунарских ресурса, прављење резервних копија података, дефинисање метрика и упозорења на нивоу система и платформе, писање неопходне документације и слично.

У склопу одржавања неопходно је редовно проверавати да ли је потребно ажурирати рачунарске ресурсе, оперативни систем, системске сервисе и слично, тестирати нове верзије, мењати конфигурације уколико је потребно, надгледати параметре попут оптерећења процесора, заузетости меморије, саобраћаја, тестирати могућност обнове података на основу резервних копија и слично.

Једна од основних предности *serverless* приступа јесте чињеница да је *cloud* провајдер задужен за спровођење свих (или бар већине) ових активности. С друге стране, у случају контејнерског микросервис приступа сва одговорност је на људима који се баве развојем и одржавањем апликације.

4.3. Трошкови продукције

Cloud провајдери пружају калкулаторе који корисницима омогућавају да израчунају потенцијалне трошкове употребе *cloud* сервиса. Са друге стране, када одлучимо да изнајмљујемо сервере имамо информацију о цени сервера коју одређују карактеристике сервера попут меморије, CPU-а и SSD диска. Из тог разлога у наставку се приказују конкретни трошкови само за рачунарске ресурсе (*compute*). Наплаћивање AWS сервиса функционише по принципу да се плаћа онолико колико се заправо троши. У случају **ламбда** функција то значи да се плаћа у складу са бројем захтева и временом извршавања функције као и потребном меморијом. На месечном нивоу бесплатно је првих милион захтева након чега се плаћа 0,2 $ за милион захтева. Када је реч о цени извршавања функција параметри који утичу на цену су време и меморија потребни да би се функција извршила. На месечном нивоу бесплатно је 400.000 GB-s а након тога се плаћа 0,00001667 $ по GB-s. С друге стране, када је реч о ценама за изнајмљивање сервера на *DigitalOcean*-у оне зависе од карактеристика саме машине (меморија, CPU, SSD диск) и цене из стандардне понуде крећу се од 5 до 960 $ месечно. **У првом случају** ћемо претпоставити да имамо апликацију која се често користи али не захтева пуно меморије и времена да би се извршавала. Рецимо да имамо следеће карактеристике: број захтева – 40.000.000, време потребно за извршавање – 0,3 s а потребна меморија 128 MB.

Резултати прорачуна су показали да би трошкови за AWS *Lambda* сервис били 26,137 $ а за DigitalOcean сервере са карактеристикама 1 GB меморије, 1 CPU, 25 GB SSD – минимални трошкови били би 50 $. Трошкови за ламбду су скоро па дупло мањи те се може закључити да је у овом случају ламбда бољи избор. **У другом случају** ћемо претпоставити да имамо апликацију која се користи за неке захтевне обраде или анализе велике количине података што значи да је временски и меморијски захтевна а да је интензитет саобраћаја мањи.

Рецимо да имамо следеће карактеристике: број захтева – 1.000.000, време потребно за извршавање – 2 min тј. 120 s а потребна меморија 3008 MB. Резултати прорачуна су показали да би трошкови за AWS *Lambda* сервис били 5869,507 $ а за DigitalOcean сервере са карактеристикама 3 GB меморије, 1 CPU, 60 GB SSD – минимални трошкови били би 1410 $. Из ових резултата може се закључити да су трошкови за *DigitalOcean* сервере мањи. На примеру *serverless* апликације приказане у овом раду може се уочити да се поред ламбда функција користе и други AWS сервиси (S3, *DynamoDB*, *Cognito*, API *Gateway*, *CloudFront*). Уколико бисмо желели да идентичну апликацију развијемо и извршавамо на серверима *DigitalOcean*-а морали бисмо пронаћи open-source (јавно доступне) алтернативе (попут *MinIO*, *MongoDB*, *Keycloak*, *nginx*, *CloudFlare*), инсталирати их и конфигурисати у сладу са нашим потребама. То би последично само повећало број сервера неопходних за извршавање апликације а самим тим би и трошкови порасли.

4.4. Скалабилност

Када је реч о скалабилности ламбда функције може се рећи да је то једна од њених основних предности и карактеристика јер AWS аутоматски скалира ламбда функцију. С обзиром да је код функције *stateless* AWS ламбда може да покрене онолико копија функције колико је потребно. AWS ламбда ће динамички алоцирати капацитет у складу са долазним саобраћајем – када има пуно захтева које треба обрадити биће покренуто више инстанци, у супротном када нема пуно захтева стопираће претходно покренуте инстанце. Број инстанци које се могу извршавати у паралели зависи од региона и креће се од 500 до 3.000. Оно што карактерише све AWS сервисе па самим тим и ламбду јесте да су високо доступни (*highly available*).

С друге стране, када је реч о скалабилности сервера ту је прича много другачија. Ми смо ти који су дужни да обезбеде довољан број сервера да би се успешно обрадили сви долазни захтеви. То подразумева свакодневно надгледање (*monitoring*) употребе ресурса сервера и да ли је тренутна инфраструктура у стању да се избори са долазним саобраћајем. Када се уочи потреба за новим серверима, потребно их је изнајмити, затим обавити сву потребну конфигурацију и инсталацију неопходну за извршавање апликације. Такође, за доступност сервиса смо ми задужени што значи да све потенцијалне проблеме који доводе до прекида нормалног извршавања аликације морамо бити у стању што пре да решимо јер време када апликација није доступна носи за последицу новчане губитке.

5. ЗАКЉУЧАК

Основна идеја рада била је да се прикажу предности и мане два различита приступа развоја софтверског производа тј. предности и мане развоја апликација са *serverless* и контејнерском микросервис архитектуром. Конкретан одговор на питање који приступ је бољи није могуће дати јер избор приступа у великој мери зависи од домена, случајева употребе (*use cases*), ограничавајућих спољашњих фактора попут расположивог буџета, људских ресурса, временских рокова и слично. Међутим, чини се да је *serverless* приступ погодан онда када је код једноставан тј. када пословна логика није комплексна са аспекта ресурса и времена извршавања, када се апликација употребљава периодично или једноставно није могуће предвидети саобраћај а желимо скалабилно решење.

С друге стране, када имамо ресурсно захтевне и дуготрајне обраде или када имамо комплексну апликацију са уједначеним саобраћајем (у смислу да је сервер већи део дана заузет обрађивањем захтева) онда је контејнерски микросервис приступ боља опција од *serverless* приступа. Када је реч о идејама за наставак истраживања на ову тему, било би занимљиво приказати и истражити потенцијалне алтернативе AWS сервиса код других *cloud* провајдера и направити поређење трошкова. Такође, могло би се направити поређење трошкова изнајмљивања сервера за различите провајдере као што је то урађено на примеру *DigitalOcean*-а.

6. ЛИТЕРАТУРА

[1] *What is could computing?*, https://aws.amazon.com/what-is-cloud-computing/ (приступљено у септембру 2020.)

[2] Michael Wittig, Andreas Wittig. *Amazon Web Services in Action, Second Edition*. Manning Publications Co, New York, 2018. ISBN 9781617295119

[3] *What is a Container?*, https://www.docker.com/resources/what-container (приступљено у септембру 2020.)

Кратка биографија:

Јелена Калабић рођена је 22.10.1994 године у Ужицу, Република Србија. Године 2013. завршава општи смер гимназије „Светозар Марковић“ у Новом Саду и уписује се на Факултет техничких наука, одсек Електротехника и рачунарство, смер Рачунарство и аутоматика. Године 2017. уписује мастер академске студије на истом смеру. Добитница је „Доситеја“ стипендије за постигнут успех на основним академским студијама. Године 2018. добија стипендију немачке привреде „др Зоран Ђинђић“ и шест месеци проводи на стручној пракси у Хамбургу у компанији *Esome advertising technologies*.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE26Krivokapic

UPOREDNA ANALIZA ALATA ZA UPRAVLJANJE *API*-JIMA

COMPARATIVE ANALYSIS OF API MANAGEMENT TOOLS

Marija Krivokapić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *Ovaj rad se bavi uporednom analizom sledeća tri alata: Akana, Apigee i 3Scale. Osnovnih 6 koraka koji su sastavni deo životnog ciklusa jednog API-ja je odrađeno u svakom od navedena 3 alata, ne bi li se utvrdilo koji je najlakši za upotrebu i koji pruža najviše mogućnosti.*

Ključne reči: *API, alati, API Menadžment, Akana, Apigee, 3Scale*

Abstract – *Main goal of this paper is to perform comaprative analysis of following 3 tools: Akana, Apigee and 3Scale. Basic 6 steps, that are part of API lifecycle, have been performed in each of 3 mentioned tools, in order to realize which of them is easiest in reallife use, and as well which of them is offering the biggest number of possibilities.*

Keywords: *API, tools, API Management, Akana, Apigee, 3Scale*

1. UVOD

U današnje vreme, eri računara, napredak tehnologije je sve veći. Budući da je život nemoguće zamisliti bez elektronskih uređaja, interneta, društvenih mreža, pojavljuje se velika ekspanzija novih tehnologija koje olakšavaju kreiranja svega što je potrebno korisnicima. Jedna od tih tehnologija je predstavljanje servisa preko *API*-ja radi lakšeg i sigurnijeg korišćenja veb i mobilnih aplikacija. Inicijalno, *API* je kreiran u svrsi definisanja i opisivanja bilo kog programskog interfejsa za biblioteku, ili modula koji predstavljaju deo nekog većeg softverskog sistema. Tokom poslednjih par godina, najčešće se koristi kao arhitekturalni stil u klijent-server komunikaciji, sa oslanjanjem na HTTP protokole [1].

Sa pojavom *API*-ja, nastaje i nova oblast rada pod nazivom *API Management*. Ta oblast obuhvata procese za distribuciju, kontrolu i analizu *API*-ja. Glavni cilj *API Management*-a je što veće olakšavanje potreba programera, prilikom kreiranja *API*-ja, kao i tokom nadgledanja aktivnosti prilikom upotrebe već postojećih *API*-ja. Jedna od najbitnijih prednosti *API*-ja je mogućnost brzih izmena samog servisa radi ispunjavanja korisničkih zahteva. Upotreba mikroservisne arhitekture umnogome pomaže ubrzavanju razvijanja softvera. HTTP orijentisani API-ji predstavnjaju sponu između mikroservisa u sinhronoj interakciji mikroservisne arhitekture. Još jedna korisna strana *API*-ja je ta što upotrebom različitih polisa obezbeđuju sigurnost [2].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.

2. KORIŠĆENI ALATI

2.1. Akana

Ovaj alat omogućava potpuni životni ciklus *API Management*-a sa kraja na kraj. Taj životni ciklus obuhvata dizajniranje, implementiranje, obezbeđivanje sigurnosti, nadgledanje (*monitoring*) i na kraju publikovanje *API*-ja. U okviru svega prethodno navedenog se podrazumeva razmena podataka preko *API*-ja, kao i povezivanje i integrisanje različitih aplikacija [3].

2.2. Apigee

Isto kao prethodno opisan alat, i Apigee omogućava potpuni životni ciklus *API Management*-a sa kraja na kraj. Pored svega pomenutog, Apigee nudi mogućnost monetizacije [4]. Takođe, još neke od pogodnosti koje ovaj alat nudi su automatsko razvijanje dokumentacije, podržavanje cloud sistema, upotreba principa mašinskog učenja zarad što boljih analiza [5].

2.3. 3Scale

3Scale je alat koji omogućava potpuni životni ciklus *API Management*-a sa kraja na kraj, i kao i Apigee pruža mogućnost monetizacije. Neki od benefita ovog alata su: kontrola saobraćaja, integracija sa OpenShift-om i RedHat Fuse-om, podržavanje hybrid cloud sistema... [6].

3. METOD ISTRAŽIVANJA

3.1. Kreiranje *API*-ja

Prilikom njegovog kreiranja najbitnije je definisati 3 stvari:

- Pristupni (*access*) URL – to je URL na koji će se slati zahtevi sa aplikacije (neki *frontend*)
- Metode – resursi koji će biti omogućeni na *API*-ju. U okviru njihovog definisanja specificira se koji HTTP glagol će se odnositi na taj resurs kao i sam naziv resursa.
- Backend URL – predstavlja pritupnu tačku krajnjeg sistema. Na ovaj URL se prosleđuju primljeni zahtevi.

3.2. Konfiguracija autorizacije

U idealnom slučaju, zarad što veće sigurnosti, trebalo bi imati 2 autorizacije, jednu postavljenu između klijenta i *API*-ja, a drugu između *API*-ja i *backend*-a. Ove konfiguracije se razlikuju od platforme od platforme, ali u najčešćem slučaju se to podešava uz pomoć ugrađenih polisa.

3.3. Dodavanje restrikcija

U cilju sprečavanja preopterećenja servera, DDoS napada, kao i zbog oslobođanja resursa u što kraćem vremenskom roku, uvode se različite restrikcije. Tri najvažnije restrikcije su: Concurrency, Quota i Timeout. Pored ovih restrikcija, zarad ograničavanja dolaznih zahteva, kreiraju se CORS polise.

3.4. Izmena zahteva

Postoje slučajevi kada *backend* očekuje zahtev drugačije formatiran u odnosu na zahtev koji je prvobitno došao do *API*-ja. Kod svakog zahteve se mogu modifikovati i zaglavlja i samo telo zahteva.

Ukoliko se radi o slučaju izmene tela zahteva, kao što je npr. pretvaranje iz XML-a u JSON, najčešće se koriste ugrađene polise.

3.5. Testiranje *API*-ja

Radi provere same funkacionalnosti, kao i verifikacije da li su ispunjeni svi klijentski zahtevi, nakon kreiranja *API*-ja sledi njegovo testiranje. Pored validnih, često se svesno šalju neispravni zahtevi ne bi li se proverilo da li je pokriven svaki slučaj koji kasnije može dovesti do velikih problema i incidenata. Još jedna od vrsta testiranja su stres ili performans testovi. Prilikom ovakvog testiranja šalje se velik broj zahteva, i cilj je simuliranje realnog sistema.

3.6. Publikovanje i monitoring

Od momenta publikovanja tj. postavljanja *API*-ja na produkciju, on postaje javno dostupan što znači da pored programera, i ljudi iz spoljnih sistema mogu slati zahteve na taj *API*. Nakon toga, kad *API* počne da se upotrebljava, programeri imaju mogućnost monitoringa. To podrazumeva pregled svih primljenih zahteva, bilo da su oni uspešni ili ne.

4. REZULTATI

4.1. Akana

Prilikom kreiranja *API*-ja programeri koriste dva portala:

- Community Manager (CM) – za kreiranje *API*-ja, konfigurisanje metoda, URL-ova, kreiranje aplikacija, testiranje i nadgledanje *API*-ja
- Policy Manager (PM) – takođe nudi mogućnost kreiranja i nadgledanja *API*-ja, ali je manje *user friendly*, te se ne koristi u te svrhe. Za razliku od CM-a, programeri samo ovde mogu kreirati polise za restrikcije. S druge strane, PM ne nudi mogućnost testiranja *API*-ja.

4.1.1. Kreiranje *API*-ja

U slučaju kreiranja REST *API*-ja, bira se prva opcija prilikom čega se unose naziv *API*-ja i *backend* URL. Ukoliko se pak radi o SOAP *API*-ju, bira se druga opcija, gde se uveze fajl koji se prethodno mora nalaziti lokalno na samoj mašini.

Sledeći korak je definisanje metoda. Kod SOAP *API*-ja je ovaj korak nepotreban budući da se sve te informacije već prethodno trebaju nalaziti u Swagger/WSDL fajlovima. Prilikom kreiranja metoda definiše se njen naziv i bira se vrsta metode tj. bira se HTTP glagol.

Zatim sledi definisanje opisa samog *API*-ja, koje uključuje unos detalja o *API*-ju, podataka o verzijama, dodavanje ikonice i tagova.

4.1.2. Konfiguracija autorizacije

Za potrebe autorizacije, kod Akane se kreiraju aplikacije. Svaka aplikacija ima svoj jedinstven identifikator i šifru, koji imaju ulogu *username*-a i *password*-a prilikom kreiranja validnih tokena. Prilikom njenog kreiranja programer definiše naziv, identifikator (App ID) i šifru (Shared Secret).

Zatim se treba vratiti na *API*, i sa toolbar-a koji se nalazi sa leve strane treba odabrati deo „Implementations“. Na sredini se nalazi deo sa polisama koji je takođe bitan deo podešavanja autorizacije. Tu se mogu naći *default*-ne ugrađene polise, ali takođe i polise kreirane od strane programera. Svaka od polisa se može primeniti na bilo koji *API*.

4.1.3. Dodavanje restrikcija

Restrikcije se definišu na nivou koji je između *API*-ja i aplikacije. Kreiranje restrikcija se vrši u PM-u. U zavisnosti od vrste polise se bira podfolder, „Operational Policies“ za CORS polise ili „QoS Policies“ za kreiranje polisa koje ograničavaju broj zahteva ili otvorenih konekcija u određenom vremenskom periodu i sl.

Nakon što su sve neophodne polise kreirane, aplikacija i API se povezuju. Njihovo povezivanje se vrši u CM-u. Klikom na „Access“ dugme na stranici *API*-ja, programerima se nudi lista postojećih aplikacija i oni biraju sa kojom aplikacijom iz liste žele da povežu svoj *API*. Na kraju ostaje da odabere koje sve restrikcije želi da važe za njegov *API*.

4.1.4. Izmena zahteva

Bilo kakva izmena zahteva u Akani podrazumeva izmenu procesa samog resursa. „Receive“ i „Reply“ se smatraju početnom i krajnjom aktivnošću i automatski se kreiraju prilikom kreiranja svake metode. Svaka željena izmena zahteva se izvršava definisanjem odgovarajućih skripti koje moraju biti smeštene između „Receive“ i „Reply“ aktivnosti.

Ukoliko je potrebno izvršiti transformaciju formata tela zahteva, koristi se aktivnost pod nazivom „FreeMarker“.

Za dodavanje i uklanjanje zaglavlja, kao i za modifikacije izgleda resursa koristi se aktivnost pod nazivom „Script“.

4.1.5. Testiranje *API*-ja

Potrebno je iz toolbar-a odabrati opciju „Test Client“. Zatim se programer preusmerava na stranicu za terstiranje gde može da odabere koju medotu želi da testira. Dugme „Setup“ služi za odabir aplikacije, tj. zahteve kog klijenta programer želi da simulira, ukoliko je više aplikacija povezano sa jednim *API*-jem. Klikom na dugme „Invoke“ inicira se zahtev, koji simulira klijentski zahtev.

4.1.6. Publikovanje i monitoring

Akana je konfigurisana tako, da nisu potrebni posebni koraci za publikovanje. Onog momenta kada se definišu zona i *context path*, *API* postaje javno dostupan. Do logova i pregleda prispelih zahteva se može doći odlaskom na deo pod nazivom „Analytics“. Pored vremena, može se filtrirati i po aplikaciji sa koje je došao zahtev. Uspešni zahtevi su označeni zelenom bojom, a pali crvenom. Prikazane su osnovne informacije kao što su status kod, vreme iniciranja zahteva, utrošeno vreme na odgovor... Klikom na bilo koji od tih zahteva, dobija se detaljniji pregled za taj konkretan zahtev.

4.2. Apigee

Ceo proizvod se sastoji iz sledeće 3 komponente:

- Apigee Edge – glavna komponenta, koja omogućava izvršavanje svih koraka potrebnih za nastanak jednog *API*-ja, počevši od kreiranja pa sve do njegovog publikovanja i monitoringa.

- Apigee Sense – komponenta kojoj je glavna uloga zaštita *API*-ja od neželjenih zahteva uključujući i napade od zlonamernih klijenata. Ova komponetna konstantno analizira saobraćaj zahteva, pri čemu identifikuje šablone koji mogu predstavljati nepoželjne zahteve.
- Apigee Monetization – podkomponenta koja programerima pruža mogućnost pristupa planovima stopa pretplata budućih klijenata, omogućava automatizaciju procesa isplaćivanja i naplate,...

4.2.1. Kreiranje *API*-ja

Prvobitno se kreira specifikacija koja je u stvari fajl tipa YAML i u sebi sadrži opise samih *API*-ja, metode tog *API*-ja zajedno sa HTTP glagolima, parametrima i kodovima odgovora.

Sledeći korak je kreiranje proksija. Odabirom opcije za kreiranje novog proksija programeru se nudi izbor vrste proksija. Najčešću upotrebu ima „Reverse Proxy“, tj. proksi kod kog se konfiguriše krajnji URL *backend* servisa.

Poslednji korak prilikom kreiranja proksija je izbor da li taj proksi treba da postoji na testnom ili produkcionom okruženju, ili pak na oba.

4.2.2. Konfiguracija autorizacije

Iz sigurnosnih razloga, Apigee uvodi proizvode koji predstavljaju neki konkretan *API*. Nakon uspešno kreiranog proizvoda, sledi kreiranje aplikacije. Prilikom kreiranja aplikacije osim osnovnih detalja kao što je naziv, treba upisati listu programera, ili bar jednog, kojima će biti dozvoljen pristup aplikaciji.

Po završetku kreiranja proizvoda i aplikacije, sledi dodavanje polise za autentifikaciju. Prvo se treba vratiti na detalje proksija i otići na *Develop* tab. Postoje predefinisane polise za različite vrste autentifikacije. Posle kreiranja polisa, one će postati vidljive u odeljku sa polisima, te ih onda treba prevući na dijagram koji simulira izgled toka.

4.2.3. Dodavanje restrikcija

Dodavanje restrikcija je poprilično slično dodavalju polisa za autentifikaciju, s tim što u ovom slučaju treba posmatrati početni deo liste vrsta polisa označen kao „Traffic Management“. Restrikcija koja se najčešće koristi kao zaštita od napada je ograničavanje broja zahteva poslatih u određenom vremenskom intervalu. Ona se dodaje kreiranjem polise tipa „Spike Arrest“. Ova polisa je tipa XML i ograničenje, tj. broj zahteva u intervalu se specificira pod tagom „<Rate>“.

4.2.4. Izmena zahteva

Ukoliko je potrebno dodavanje nekih zaglavlja sa predefinisanim vrednostima ili preuzetim iz tela inicjalnog zahteva, preporučuje se upotreba polisa koje su u suštini JavaScript fajlovi.

U slučaju transformacija zahteva iz jednog tipa u drugi kao npr. iz JSON-a u XML i obrnuto, Apigee nudi ugrađene polise. Nakon kreiranja neke od ovih polisa neophodno je jedino prevući ih na deo toka u dijagramu.

4.2.5. Testiranje *API*-ja

Proces testiranja počinje klikom na dugme pod nazivom „Start Trace Session“. Jedna sesija traje 10 minuta. Za slanje zahteva preporučuje se upotreba nekog drugog alata kao što je npr. Postman. Programer se može kretati kroz svaki momenat zahteva, izvršenje svake skripte, te pred kraj, kada se dođe do dela za odgovor tačno se ispisuje odgovor koji je *backend* poslao.

4.2.6. Publikovanje i monitoring

Nakon svakog kreiranja nove verzije, programer ima opciju da odluči da li će ta verzija ići na testno ili produkcino okruženje, birajući iz padajućeg menija dela „Deployment“.

Što se tiče monitoringa, u „Analyze“ delu, programeru se pružaju različite mogućnosti pregleda izveštaja: može da bira sa okruženja želi da vidi podatke; može da bira vremenski period za koji ga interesuje saobraćaj, kao i da li želi izveštaje za sve proksije, ili ga pak interesuje samo jedan konkretan proksi.

4.3. 3Scale

Poslednji alat koji će biti obrađen, slično kao i prethodni, sastoji se iz više komponenti:

- Admin portal – glavni portal gde se kreiraju i definišu sve bitne stvari vezane za jedan *API*. Ono što je specifično je činjenica da svaki *API*, ima svoj poseban admin portal kom se pristupa preko jedinstvenog URL-a. Onog momenta kada se definiše „company name“, automatski se kreiraju linkovi za developer i admin portal sa istim nazivom.
- Developer portal – portal čiji je glavni cilj pružanje što boljeg iskustva i olakšavanje programerima koji će kasnije manipulisati sa već kreiranim *API*-jima. To se ostvaruje omogućavanjem sve neophodne dokumentacije zajedno sa primerima kodova, slučajevima upotrebe, cenovnim planovima itd.

4.3.1. Kreiranje *API*-ja

Za razliku od alata koji su prethodno obrađeni, u slučaju ovog alata, nema potrebe za ručnim kreiranjem *API*-ja budući da se on automatski kreira, sa predefinisanim vrednostima. Svaki od ovih podataka se kasnije treba izmeniti spram klijentskih zahteva.

Sledeći korak koji je dodavanje metoda. Iznad liste svih metoda, koja je inicijalno prazna se nudi opcija pod nazivom „+ New method“.

Poslednja stvar koja se treba izmeniti jeste *backend* URL.

4.3.2. Konfiguracija autorizacije

Prvi korak je kreiranje aplikacionog plana. Za jedan *API* može da se definiše više aplikacionih planova, gde uvek postoje dva predefinisana. U okviru plana se definiše lista korisnika i kreira se aplikacija.

Nakon što su sve potrebne komponente kreirane, treba definisati vrstu autorizacije. Naime, 3Scale, nudi 3 izbora: Api key, Uređeni par app_id i api_key, OAuth. Pored izbora vrste autentifikacije, programer ima mogućnost odabira lokacije kredencijala.

4.3.3. Dodavanje restrikcija

Restrikcije se definišu u okviru aplikacionog plana, s tim što ovaj alat nudi isključivo restrikciju koja se odnosi na ograničavanje broja zahteva u određenom vremenskom intervalu. Odabirom opcije „+ New usage limit“ kreira se nova restrikcija tj. limit, i tom prilikom se definišu maksimalan broj zahteva i vremenski period.

4.3.4. Izmena zahteva

Odabirom dela „Policies“ iz „Integration“ odeljka, programeru se prikazuje lanac svih dodataih polisa i redosled njihovog izvršavanja.

Što se tiče polisa koje se odnose na izmene tela zahteva, to je moguće dodavanjem ekstenzija za polise, koje je moguće izvršiti samo uz pomoć Red Hat Fuse-a i OpenShift klastera, za šta su neophodne mnoge dodatne instalacije.

4.3.5. Testiranje *API*-ja

Da bi započela testiranja *API*-ja, on se prvo mora prebaciti u fazu za testiranje.

Potom treba kreirati novu specifikaciju odlaskom na opciju „ActiveDocs“. Nakon što programer sva polja popuni popuni, klikom na dugme „Try it out!“, okida se zahtev.

4.3.6. Publikovanje i monitoring

Postupak publikovanja je identičan stavljanju *API*-ja na testno okruženje, osim sto treba odabrati opciju „Promoting v.1 to Production APIcast“.

Na samom kraju ostaje još samo monitoring. Odlaskom na odeljak „Analytics“, nude se različite mogućnosti nadgledanja *API*-ja na produkciji. Pored ukupnog saobraćaja, nude se mogućnosti pregleda proseka na dnevnom nivou, na nivou sata, alerti, kao i status kodovi zahteva.

5. ZAKLJUČAK

API Management, iako relativno nova tehnologija, umnogome pomaže i olakšava posao programerima omogućavajući dosta funkcionalnsti, unapred predefinisanih. Što se tiče ključnih osobina alata za upravljanje *API*-jima, svaki od ovih alata ih podržava. Lista ključnih osobina je sledeća:

- Access Control – svaki od alata podržava kontrolu pristupa na taj način što administratori definišu listu programera koji imaju pravo pristupa.
- Analytics – kao i prethodnu osobinu, svaki od 3 alata koja su obrađena u ovom radu, podržava analizu javno dostupnih servisa, pružajući informacije o saobraćaju, broju uspešnih i neuspešnih zahteva, status kodovima itd. Ipak, kod ove osobine, najbolje se pokazala Akana, budući da ona, osim informacija o uspešnosti zahteva, čuva podatke i o tome kako su izgledali zahtevi.
- API Design – osobina prisutna kod svih alata. U svakom od njih dizajniranje tj. kreiranje API-ja je izuzetno jednostavan korak.
- Testing Management – svaki alat pruža mogućnost testiranja API-ja pre nego što postanu javno dostupni. U slučaju testiranja, najbolje se pokazao Apigee, zato što kod svakog zahtev koji se pošalje u toku jedne sesije, moguće je ispratiti i pregledati svaki odrađen korak, počevši od izgleda svih zaglavlja i tela, do izvršenja svake polise koja je ubačena u tok..
- Threat & Traffic Protection – osobina koja se izvršava uz pomoć restrikcija, preko predefinisanih ili ručno kreiranih polisa. Slično prethodnoj osobini, i ovde se može reći da je Apigee najbolji alat, zato što pruža najveći broj ugrađenih polisa koje programeri mogu koristiti.
- Version Control – poslednja u nizu osobina koju pruža svaki do alata. Dodavanje nove verzije je poprilično slično realizovano u svakom od alata.

Na samom kraju, može se izvući zaključak da je Apigee alat koji je najbolje koristiti u cilju upravljanja *API*-jima. Ovo se odnosi na perspektivu programera, tj. Apigee je najjednostavniji i pruža najviše mogućnosti, ugrađenih komponenti, olakšavajući programerima ceo životni ciklus jednog *API*-ja.

6. LITERATURA

[1] https://medium.com/@robert.broeckelmann/what-are-apis-the-technology-perspective-ca7e33d383c1 (pristupljeno u aprilu 2020.)

[2] https://www.redhat.com/en/topics/api/what-is-api-management (pristupljeno u aprilu 2020.)

[3] https://www.akana.com/products/api-platform (pristupljeno u aprilu 2020.)

[4] https://docs.apigee.com/api-platform/get-started/what-apigee-edge (pristupljeno u aprilu 2020.)

[5] https://searchapparchitecture.techtarget.com/definition/Apigee (pristupljeno u aprilu 2020.)

[6] https://www.redhat.com/en/technologies/jboss-middleware/3scale (pristupljeno u maju 2020.)

Kratka biografija:

Marija Krivokapić rođena je 21.12.1995. godine u Novom Sadu. Završila je osnovnu školu „Petefi Šandor“ kao nosilac Vukove diplome. Gimnaziju „Jovan Jovanović Zmaj“ završava 2014. godine kao nosilac Vukove diplome, tokom čijeg pohađanja je bila i član školskog hora. Školske 2014/2015 upisuje Fakultet tehničkih nauka, smer Elektroenergetski softverski inženjering. Letnji semestar školske 2017/2018 bila je na razmeni u Univerzitetu u Groningenu, u Holandiji, gde je položila sve predmete. U septembru 2018. završava osnovne studije, nakon čega upisuje master studije takođe na Fakultetu Tehničkih nauka, smer Računarstvo i Automatika, podsmer – Elektronsko poslovanje.

Зборник радова Факултет техничких наука, Нови Сад

UDK: 004.9
DOI: https://doi.org/10.24867/10BE27Todorovic

НАМЕНСКИ ЈЕЗИК И ОКРУЖЕЊЕ ЗА МОДЕЛОВАЊЕ ПРОИЗВОДНИХ ПРОЦЕСА И ГЕНЕРИСАЊЕ ДОКУМЕНТАЦИЈЕ

A DOMAIN-SPECIFIC LANGUAGE AND A FRAMEWORK FOR PRODUCTION PROCESS MODELING AND GENERATING DOCUMENTATION

Милица Тодоровић, *Факултет техничких наука, Нови Сад*

Област : ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *У овом раду описан је текстуални наменски језик за моделовање производних процеса. Креирани наменски језик омогућава једноставно и брзо креирање формалних, рачунарима разумљивих модела производних процеса неопходних за одвијање процеса производње у паметним фабрикама. За описани наменски језик имплементирани су генератори техничке документације. Генератори омогућавају аутоматско креирање неопходне техничке документације, чиме се смањује време потребно за израду документације, али и повећава њена конзистентност и квалитет. За развој мета-модела коришћено је окружење Eclipse Modeling Framework, радни оквир Xtext за развој текстуалне синтаксе и језик Xtend за развој генератора.*

Кључне речи: *производни процеси, техничка документација, индустрија 4.0, наменски језици, развој софтвера вођен моделима*

Abstract – *In this paper, we present a textual domain-specific language for production process modeling. The created domain-specific language enables easy and efficient creation of formal, machine-readable production process models, which are necessary for production in smart factories. For the described language, technical documentation generators were implemented. Generators enable automatic creation of required technical documentation, which reduces the time necessary for the creation process and increases quality and consistency of the documentation. Eclipse Modeling Framework was used to create the meta-model. Xtext was used to develop the concrete language syntax, while Xtend was used to implement documentation generators.*

Keywords: *Production Processes, Technical Documentation, Industry 4.0, Domain-Specific Languages, Model-Driven Software Development*

1. УВОД

Кроз историју, три велике индустријске револуције проузроковале су значајне промене у начину на који се реализује производња. Тренутно се дешава четврта индустријска револуција, проузрокована повећаном потражњом за производима прилагођеним специфичним жељама и потребама купаца. Немачка, Сједињене Америчке Државе и друге државе са високо развијеном индустријом, покренуле су националне програме и истраживања на тему четврте индустријске револуције.

Четврта индустријска револуција заснива се на новим технологијама и концептима попут сајбер-физичких система (енгл. *Cyber-Physical System*), интернета свега (енгл. *Internet of Things*) и бежичних мрежа сензора (енгл. *Wireless Sensor Network*). Ове технологије омогућавају креирање паметних објеката, попут паметних машина, производа и материјала. Паметни објекти, заједно са људима, чине паметне фабрике. Паметне фабрике су свесне тренутног стања производног система и плана производње. На основу познавања тих података, паметна фабрика доноси одлуке о оптималном начину производње у реалном времену и врши координисање и синхронизацију људи и паметних објеката у оквиру производног система. Да би паметна фабрика могла изабрати оптималан начин производње, она мора познавати производне процесе. Због тога, производни процеси морају бити формално описани на начин који је разумљив паметним фабрикама.

Формални модели производних процеса могу се користити и за генерисање техничке документације. Правне регулативе обавезују произвођаче да обезбеде пратећу техничку документацију за производ и процес производње. Ово резултује потребом за постојањем велике количине техничке документације. Велика количина потребне техничке документације као резултат има велике трошкове, ниску ефикасност и дуго трајање процеса креирања документације, али и низак квалитет документације [1]. Аутоматско генерисање дела или све потребне документације на основу формалног описа производног процеса решило би описане проблеме.

Анализом постојећих језика за моделовање производних процеса установљено је да ниједан од њих не омогућава моделовање свих детаља производног процеса неопходних за аутоматско генерисање техничке документације. Због тога, одлучено је да се креира нови наменски језик за описивање производних процеса назван *Production Domain-Specific Language* (*pDSL*).

Осим *Увода* и *Закључка*, овај рад садржи три поглавља. Поглавље *Преглед постојећег стања у области* садржи опис постојећих начина креирања формалних модела производних процеса, а затим даје теоријске основе моделом вођеног развоја и наменских језика.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Владимир Димитриески, доцент.

Поглавље *Наменски језик за моделовање производних процеса* описује креирани наменски језик. Поглавље *Генерисање документације* описује имплементиране генераторе техничке документације.

2. ПРЕГЛЕД ПОСТОЈЕЋЕГ СТАЊА У ОБЛАСТИ

У оквиру овог поглавља испитани су постојећи начини моделовања производних процеса. Затим су дате теоријске основе моделом вођеног развоја софтвера. На крају је испитана могућност њихове примене за формално моделовање производних процеса и аутоматско генерисање техничке документације.

2.1. Преглед постојећих начина за моделовање производних процеса

Производни процеси у фабрици обично су описани са више различитих техничких докумената. Различите скице и документи попут *Bill of Materials* (*BOM*) моделују изглед, карактеристике и састав производа. На основу њих и карактеристика производног система дефинише се производни процес. На пример, компаније често креирају карте токова процеса (енгл. *Flow Process Chart*) или *Bill of Operations* (*BOO*) да би описале производне процесе. Наведени документи описују кораке производног процеса, али не описују ток материјала. Дакле, подаци потребни паметној фабрици подељени су у више различитих докумената. Такви модели не морају нужно бити разумљиви рачунарима и извршиви, а у појединим случајевима нису ни дигитални.

Осим приказа модела производног процеса кроз поменуту документацију, модели производних процеса могу се креирати помоћу језика за концептуално моделовање процеса. На пример, *Bussiness Process Modeling Notation* (*BPMN*), *Unified Modeling Language* (*UML*) и Петријеве мреже могу се користити за моделовање производних процеса. Поменути формализми нису креирани са циљем да се њима моделују производни процеси, већ имају општу намену. Због тога, недостају им концепти за изражавање дела особина производних процеса.

Постоји више проширења поменутих формализама развијених са циљем превазилажења поменутих недостатака. На пример, у раду [2] предложено је проширење *BPMN*-а концептима потребним за опис ресурса и материјала потребних за извршавање производних корака. Предложено проширење подразумева навођење конкретних ресурса, па је модел креиран његовом употребом зависан од производног система. Моделовање сарадње више учесника да би се обавио заједнички циљ није могуће.

Сличне недостатке имају и остали анализирани језици. Због тога, предложено је креирање новог наменског језика. Нови наменски језик треба да омогући ефикасно и лако креирање формалних модела производних процеса, који садрже све детаље потребне да би се техничка документација аутоматски генерисала.

2.2. Теоријске основе моделима вођеног развоја

Моделима вођене (енгл. *Model-Driven*) парадигме подразумевају употребу модела у свим корацима развоја комплексног система. Моделима вођене парадигме обухватају и моделима вођен развој софтвера (енгл. *Model-Driven Software Development*, *MDSD*). Неки од основних циљева моделом вођеног развоја софтвера (*MDSD*) су бржи поступак развоја софтвера, повећање квалитета софтвера, повећање степена поновне употребљивости, олакшавање управљања сложеним системима, повећање укључености доменских експерата у развој софтвера и побољшање комуникације између наручиоца и тима који развија софтвер [3].

У овом раду примењен је моделима вођен развој софтвера на процес пројектовања производних процеса, при чему је главни циљ примене овог приступа било омогућавање аутоматског генерисања техничке документације за производни процес. *MDSD* захтева постојање формалних модела. Поред генерисања документације, формални модели производних процеса могу се употребити за симулацију и оптимизацију процеса, визуелизацију и праћење извршавања процеса, као и одлучивање и контролу у стварном времену. Формални модели такође омогућавају лакшу валидацију и проверу логичке исправности модела, чиме се повећава квалитет финалног модела процеса.

Изабрани начин формирања модела јесте употребом наменског језика. С обзиром на то да је наменски језик прилагођен домену за који се креира, процес моделовања би требало да је ефикаснији и лакши него што би био да се користи језик опште намене.

На крају, трансформације модела у текст аутоматизују процес генерисања документације. Аутоматизација процеса креирања документације резултује бржим креирањем документације и смањењем трошкова.

3. НАМЕНСКИ ЈЕЗИК ЗА МОДЕЛОВАЊЕ ПРОИЗВОДНИХ ПРОЦЕСА

Приликом развоја наменског језика потребно је изабрати један или више типова конкретних синтакси које језик треба да поседује. Два најчешће коришћена типа конкретних синтакси су: графичка и текстуална конкретна синтакса [3]. Текстуалне конкретне синтаксе одликују се пре свега добром програмском подршком за моделовање. Алати за уређивање текста су распрострањени, изразито зрели и велики број корисника је већ навикнут на њих. Поред тога, алати за уређивање текста нуде додатне функционалности попут означавања синтаксе бојама, надопуњавања, анотирања грешака и приказа разлика између два документа или две верзије истог документа. Због тога, изабрано је да за наменски језик *pDSL* буде развијена текстуална синтакса. Међутим, најбоље решење за наменски језик *pDSL* је да буду развијене обе синтаксе. Поред свих предности поседовања текстуалне синтаксе, језик *pDSL* тако би имао и предности поседовања графичке синтаксе. На пример, графичка синтакса би омогућила графичку симулацију извршавања процеса. Креирање графичке конкретне синтаксе је један од могућих праваца даљег развоја наменског језика *pDSL*. Узевши ово у обзир, изабран је приступ креирања конкретне синтаксе на основу мета-модела.

3.1. Опис технологија коришћених за развој наменског језика *pDSL*

За креирање мета-модела коришћен је *Eclipse Modeling Framework* (*EMF*). *EMF* је радни оквир за моделовање и генерисање кôда намењен за креирање алата заснованих на структурираном моделу података. *EMF* као језик за мета-моделовање користи језик *Ecore*.

Концептима наменског језика за мета-моделовање *Ecore* није било могуће исказати сва потребна ограничења. Ова ограничења исказана су помоћу наменског језика *Object Constraint Language* (*OCL*), односно помоћу његове имплементације под називом *Eclipse OCL*.

За имплементацију конкретне текстуалне синтаксе коришћен је *Xtext*. *Xtext* је радни оквир за развој програмских језика и наменских језика. Обезбеђује наменске језике и одговарајући API за описивање различитих аспеката језика. На основу описа аспеката језика омогућује генерисање свих алата потребних за употребу језика.

3.2. Апстрактна синтакса

На слици 1. приказани су основни концепти апстрактне синтаксе наменског језика *pDSL*.

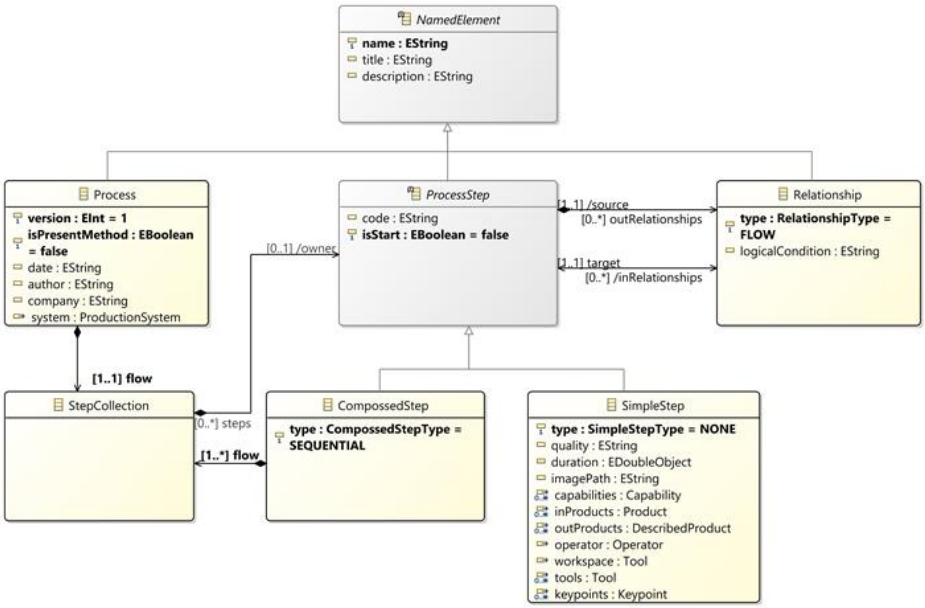
Коренски концепт апстрактне синтаксе је процес (*Process*). За процес је, пре свега, потребно описати кораке који се реализују у циљу извршења процеса (*ProcessStep*). У наменском језику разликују се сложени кораци, односно потпроцеси (*CompossedStep*) и грануларни кораци (*SimpleStep*). Улога потпроцеса је пре свега омогућавање описивања различитог тока извршења корака, на пример, извршавање у паралели. Друга улога потпроцеса је омогућавање груписања корака, ради боље прегледности. Насупрот њима, грануларни кораци представљају недељиву активност, са јасно дефинисаним циљем, улазима и излазима. Поред тога, за корак се могу дефинисати конкретни ресурси, односно средства за рад (*Tool*) и извршилац активности (*Operator*) потребни за обављање корака, или способности (*Capability*) које ресурси треба да имају да би обављали корак. Навођење способности уместо конкретних ресурса омогућава креирање модела независног од производног система. Ресурси могу бити накнадно изабрани на одговарајући начин, у складу са задатим способностима, жељеним квалитетом и производним системом.

3.3. Конкретна синтакса

Радни оквир *Xtext* на основу мета-модела генерише почетну верзију конкретне синтаксе. Да би синтакса била употребљива, генерисану почетну граматику потребно је прилагодити домену. Поред граматике потребно је прилагодити и остале аспекте језика. У случају наменског језика *pDSL* постојала је потреба за делимичном изменом начина разрешавања опсега и увођењем додатних валидација. На основу граматике и описа осталих аспеката синтаксе, *Xtext* генерише алат за уређивање кôда. По потреби, алат за уређивање кôда се такође може прилагодити језику. У случају наменског језика *pDSL* било је потребно додати шаблоне (енгл. *templates*), али и делимично изменити начин на који алат за уређивање кôда генерише сугестије приликом уноса модела (енгл. *content assist*). Ове измене начињене су како би се убрзало и олакшало креирање модела употребом алата за уређивање кôда. На листингу 1. дат је пример изгледа конкретне синтаксе, који описује један корак производног процеса.

```
INSPECTION ProveraKvaliteta(
description: "Uzima se nasumični uzorak i nosi u laboratoriju na analizu."
workstation: LaboratorijaZaProcenuKvaliteta
tools: [OtvaracKonzervi, UredjajZaMerenjeKiselosti,
        UredjajZaMerenjeKolicineSecera]
operator: Laborant
duration: 120
image: "C:\\temp\\quality.jpg"
links with other steps: [
        IdiNaPakovanje : GO TO Pakovanje,
        IdiOdlaganjeOtpada : GO TO Unistavanje
        if "Proizvod nije u skladu sa željenim kvalitetom."
]
)
```

Листинг 1. *Пример описа процесног корака*



Слика 1. *Основни концепти наменског језика pDSL*

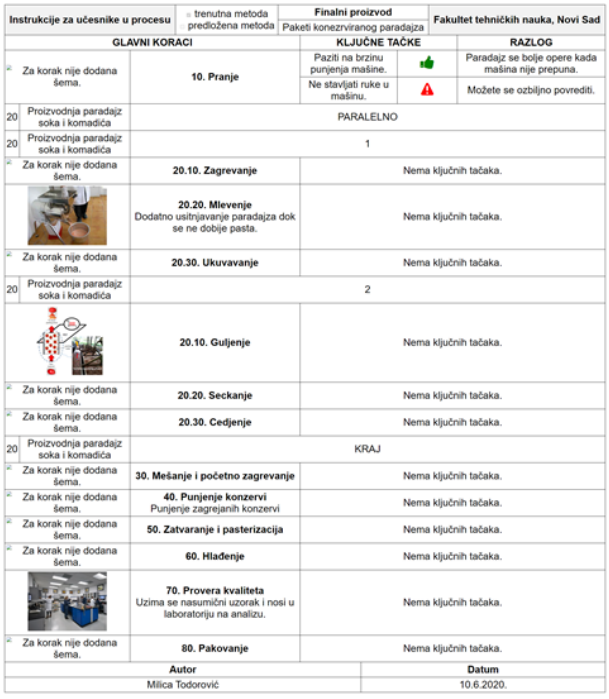
4. ГЕНЕРИСАЊЕ ДОКУМЕНТАЦИЈЕ

Да би се омогућило аутоматско генерисање техничке документације развијено је више генератора од којих сваки за циљ има генерисање одређеног типа документације намењеног за планирање, реализацију, праћење и оптимизацију производних процеса. Имплементирани су следећи генератори: генератор карти токова процеса, генератор операционих листова, *BOM* (енгл. *Bill of Materials*) генератор, *BOMO* (енгл. *Bill of Materials and Operations*) генератор, *JBS* (енгл. *Job Breakdown Sheet*) генератор, генератор прегледа средстава за рад и генератор прегледа учесника процеса. За њихову реализацију коришћена је комбинација програмског кôда писаног у програмском језику *Java* и кôда писаног језиком *Xtend*.

Сви генератори свој излаз формирају на основу података садржаних у моделу производног процеса. Такође, сви генерисани документи имају исто заглавље и подножје, користе исти фонт и формат, као и исти начин приказа појединих елемената попут назива корака и производа. Све ово доприноси конзистентности докумената.

На слици 2. приказан је пример генерисаног документа *JBS*. *JBS* представља документ за обуку радника који приказује све кораке и за сваки од њих наводи кључне тренутке и објашњење како они утичу на извршавање корака. Кључне тачке излистане у *JBS* обухватају савете како процес обавити ефикасно и на безбедан начин.

Да би се за производни процес генерисао *JBS*, али и карта тока процеса и *BOMO*, корацима је потребно доделити ознаке које указују на редослед обављања корака. Додела ознака врши се аутоматски, на основу позиције корака унутар тока извршавања процеса. Након успешног генерисања ознака, могуће је генерисати документе.



Слика 2. *Пример генерисаног документа JBS*

За генерисање документа *JBS* осим концепата за моделовање тока процеса, значај имају концепт кључног тренутка корака (*Keypoint*), али и атрибути самог корака попут назива, скице и описа.

5. ЗАКЉУЧАК

Наменски језик *pDSL*, описан у овом раду, омогућава креирање формалних, рачунарима разумљивих модела производних процеса. Језик поседује концепте помоћу којих је могуће описати сложене токове производних корака, материјала и ресурса. Поред тога, тестиране су могућности имплементације генератора од којих сваки за циљ има генерисање изабраног типа документације намењене за планирање, реализацију, праћење и оптимизацију производних процеса.

Осим тога што је погодан за функционисање паметне фабрике, формални модел производног процеса представља подлогу за реализацију система за симулацију и оптимизацију процеса, визуелизацију и праћење извршавања процеса, као и аутоматско генерисање техничке документације. Аутоматско генерисање документације побољшало би ефикасност и повећавало би брзину процеса креирања документације. Генерисање документације допринело би конзистентности техничке документације, као и њеној усклађености са прописаним стандардним, чиме би се осигурао квалитет документације.

Наменски језик *pDSL* може бити додатно развијен и унапређен. На пример, може се проширити графичком конкретном синтаксом. Графичка синтакса би, пре свега, омогућила реализацију система за визуелизацију и симулацију извршавања процеса. Поред тога, графичка синтакса би допринела лакшем моделовању и откривању грешака. Осим тога, могуће је и креирање додатних генератора за моделе описане наменским језиком *pDSL*. Додатни генератори који се могу имплементирати укључују генераторе документације у вези са системом квалитета, али и генераторе других видова техничке документације.

6. ЛИТЕРАТУРА

[1] A. Barthelmey, D. Störkle, B. Kuhlenkötter, and J. Deuse, “Cyber Physical Systems for Life Cycle Continuous Technical Documentation of Manufacturing Facilities,” *Procedia CIRP*, vol. 17, pp. 207–211, 2014, doi: 10.1016/j.procir.2014.01.050.

[2] S. Zor, D. Schumm, and F. Leymann, “A Proposal of BPMN Extensions for the Manufacturing Domain,” *Proceedings of 44th CIRP international conference on manufacturing systems, Madison, Wisconsin, USA*, pp. 1–7, 2011.

[3] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven software engineering in practice*, Second edition. San Rafael, Calif.: Morgan & Claypool Publishers, 2017.

Кратка биографија:

Милица Тодоровић рођена је 1997. године у Милићима у Босни и Херцеговини. Факултет техничких наука уписала је 2015. Дипломски рад из области Електротехника и рачунарство одбранила је 2019. године.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE28Ivanisevic

NAMJENSKI JEZIK I OKRUŽENJE ZA MODELOVANJE GREŠAKA U PROIZVODNIM PROCESIMA I GENERISANJE DOKUMENTACIJE FMEA

DOMAIN SPECIFIC LANGUAGE AND TOOL FOR MODELING ERRORS IN PRODUCTION PROCESSES AND GENERATION OF FMEA DOCUMENTATION

Đorđije Ivanišević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – PRIMJENJENE RAČUNARSKE NAUKE I INFORMATIKA

Kratak sadržaj – *U industriji se trenutno događa četvrta revolucija čiji cilj je da upotrebom savremenih tehnologija odgovori na zahtjeve tržišta za brzom adaptacijom proizvoda, a posljedično i procesa proizvodnje. U situaciji kada se proizvodni procesi brzo mijenjaju, mogu se pojaviti greške koje je teško predvidjeti što povećava cijenu troškova proizvodnje. U ovom radu predstavljeni su namjenski jezik i okruženje za modelovanje grešaka u proizvodnim procesima, koji bi mogli uticati na povećanje stepena automatizacije i smanjenje cijene troškova proizvodnje. Na osnovu modela proizvodnih procesa koji su rezultat upotrebe predloženog namjenskog jezika moguće je automatski generisati dokumentaciju FMEA i time skratiti vrijeme potrebno za ručno pisanje dokumentacije i smanjiti broj grešaka koji se u takvom procesu javlja.*

Ključne reči: *Namjenski jezici, namjensko modelovanje, razvoj vođen modelima, proizvodni procesi*

Abstract – *The fourth revolution is currently taking place in the industry, with the goal to respond to the market demands for the rapid product and process adaptation. When production processes are changing rapidly, errors offten occur and are difficult to predict and process. Thus, the manufacturing costs are increased. In this paper, a domain-specific language and a tool for the production process and error modeling are presented. Their purpose is to increase the degree of automation and to reduce the manufacturing costs. Based on the production process models, it is possible to automatically generate FMEA documentation and thus shorten the time required to write the documentation manually which could reduce the number of errors that occur in such process.*

Keywords: *Domain-specific languagues, domain-specific modeling, model-driven development, production processes*

1. UVOD

U industriji se trenutno događa četvrta revolucija koja je nastala iz potrebe za brzom adaptacijom proizvodnje kako bi bilo moguće proizvoditi visoko prilagođene proizvode za korisnike. U tradicionalnom pristupu proizvodnji,

svaka izmjena na ciljnom proizvodu zahtjevala bi rekonfiguraciju proizvodnog pogona što je vremenski zahtjevno i utiče na povećanje troškova proizvodnje. Upotrebom savremenih tehnologija koje omogućavaju integraciju i sinhronizaciju fizičkih i virtualnih aspekata proizvodnih pogona teži se skraćenju vrijemena potrebnog za rekonfiguraciju. Fabrike koje omogućavaju takav vid integracije nazivaju se pametne fabrike. Pametne fabrike uvijek su „svjesne“ trenutnog stanja, donose odluke u realnom vremenu analizom velike količine podataka i tako omogućavaju svim učesnicima proizvodnog procesa da obavljaju svoje zadatke u što kraćem vrijemenskom periodu.

U situaciji kada se proizvodni procesi često mijenjaju, teško je isprojektovati proizvodni proces tako da bude tolerantan na greške. U ovom radu predložen je namjenski jezik i okruženje koji bi omogućili modelovanje grešaka u proizvodnim procesima na formalan i precizan način. U tradicionalnom pristupu proizvodnji, odlučivanje o izvršavanju sekvence operacija koje otklanjaju uočenu grešku zasniva se na procedurama koje nisu dovoljno automatizovane. Operater u trenutku pojave greške mora „ručno“ da otkloni grešku kako bi se proizvodnja nastavila. Prednost pristupa sa predloženim namjenskim jezikom ogleda se u tome što bi izvršavanje koraka koji obrađuju grešku bilo automatizovano na osnovu modela procesa proizvodnje.

Upotrebom predloženog namjenskog jezika, podaci o kontrolnim aktivnostima i potencijalnim greškama u proizvodnji bili bi integrisani u model proizvodnog procesa. Takve podatke bilo bi moguće iskoristiti za automatsko generisanje dokumentacije, što bi proces kreiranja dokumentacije značajno ubrzalo ali i smanjilo broj grešaka koje su posledica ručnog kreiranja dokumentacije. Ciljna dokumentacija za generisanje predložena u ovom radu preporučena je metodologijom FMEA (engl. *Failure Modes and Effect Anlysis*). Metodologija FMEA koristi istorijat uočenih grešaka za analizu, ublažavanje efekata i prevenciju pojave grešaka u budućnosti. Glavni cilj metodologije FMEA jeste stvaranje uslova za konfiguraciju proizvodnog pogona u kome se posljedice greške uočavaju i eliminišu prije nego što proizvod dođe do krajnjeg korisnika [1].

Korišćenjem namjenskog jezika za modelovanje grešaka u proizvodnim procesima bilo bi moguće povećati automatizaciju direktnim izvršavanjem specificiranih koraka od strane mašina i smanjiti troškove proizvodnje. Takođe, postojanje formalnog jezika omogućilo bi lakše i fleksibilnije modelovanje izmjena proizvodnog procesa i grešaka koje bi se mogle javiti kao posljedica tih izmjena.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vladimir Dimitrieski, docent.

2. PREGLED POSTOJEĆEG STANJA U OBLASTI

2.1. Postojeći načini za modelovanje proizvodnih procesa

Potreba za modelovanjem i opisivanjem proizvodnih procesa razvila se i prije pojave računara. Neke fabrike i danas koriste razne vidove nedigitalne tehničke dokumentacije koja opisuje tokove materijala, redoslijed izvršavanja operacija, učesnike procesa, kontrolu kvaliteta i slično. Međutim, da bi bilo moguće vršiti validaciju modela proizvodnih procesa i transformacije u druge modele ili tekst, neophodno je da model bude kreiran upotrebom formalnog jezika sa jasno definisanom sintaksom. Neki od jezika koji se koriste za modelovanje proizvodnih procesa su: Petrijeve mreže, BPMN (engl. *Business Process Model and Notation*) i UML (engl. *Unified Modeling Language*).

Modelovanje kompleksnih proizvodnih procesa upotrebom UML-a i Petrijevih mreža rezultuje modelima koji nisu dovoljno čitljivi i pregledni. U poređenju sa UML-om i Petrijevim mrežama, BPMN sadrži koncepte koji su više prilagođeni iskazivanju strukture procesa. Međutim, koncepti BPMN-a orijentisani su ka modelovanju poslovnih procesa i nisu dovoljno prilagođeni domenu proizvodnje. Kreiranjem proširenja za prethodno nabrojane jezike mogli bi se riješiti prethodno nabrojani nedostaci. Međutim, proširenja često povećavaju kompleksnost modela i projektanti proizvodnih procesa nisu navikli na njihovu sintaksu.

Razvoj namjenskog jezika za modelovanje proizvodnih procesa koji je, sa jedne strane razumljiv krajnjem korisniku, a sa druge strane dovoljno specijalizovan da može opisati sve koncepte iz domena, trebalo bi da riješi prethodno nabrojane probleme.

2.2. Namjenski jezici

Jezici za modelovanje služe da za iskazivanje informacija ili znanja o posmatranom sistemu, pri čemu imaju tačno definisan skup sintaksnih pravila. Prilikom modelovanja projektanti moraju poštovati sintaksna pravila formalno definisanih jezika za modelovanje. Postoje dvije osnovne kategorije u koje se mogu svrstati jezici za modelovanje: namjenski jezici i jezici opšte namjene. Jezici opšte namjene mogu se koristiti za opis sistema nezavisno od konkretnog domena. Namjenski jezici predstavljaju jezike specijalno kreirane za određeni kontekst kako bi olakšali opis problematike usko vezane za posmatrani domen. Kako modelovanje proizvodnih procesa zahtejva rad sa konceptima usko vezanim za domen proizvodnje, u ovom radu predložen je pristup modelovanja procesa upotrebom namjenskog jezika.

Prilikom razvoja namjenskog jezika neophodno je definisati njegovu apstratknu i konretnu sintaksu. Apstraktnu sintaksu definišu meta-modeli. Meta-modeli sadrže definiciju jezika za modelovanje i omogućavaju kreiranje klasa modela reprezentovanih tim jezikom [2]. Vizualna notacija koncepata uvedenih u meta-modelu definisana je konkretnom sintaksom.

U zavisnosti od načina iscrtavanja elemenata modela postoje dvije glavne kategorije konkretnih sintaksi: tekstualna i grafička. Upotrebom tekstualnih konkretnih sintaksi informacije o modelu se kodiraju sekvencama karaktera, dok grafička sintaksa podrazumijeva kodiranje informacija o modelu u vidu prostorno uređenih grafičkih i tekstualnih simbola.

Za implementaciju namjenskog jezika za modelovanje grešaka u proizvodnim procesima izabrana je grafička konkretna sintaksa. Glavni razlog za to je da bude omogućena lakša vizualizacija grešaka za korisnike koji nemaju programerskog iskustva tj. nisu navikli na tekstualne programske jezike.

2.3. Modelom vođeni razvoj

Modelom vođene (engl. *Model-Driven*) metodologije se mogu definisati kao metodologije koje koriste prednosti modela u inženjerskim aktivnostima. Prednosti korišćenja modela su: mogućnost istraživanja različitih rješenja u ranoj fazi razvoja, kreiranje više različitih pogleda na sistem, lakše otkrivanje grešaka i brži razvoj proizvoda. Osim toga, korišćenjem modela se povećava nivo apstrakcije za posmatrani sistem i tako se omogućava bolje razumijevanje kompleksnih sistema.

U modelom vođene paradigme spada i modelom vođeni razvoj softvera, odnosno MDSD (engl. *Model Driven Software Development*). Primjenom metodologije MDSD modeli se koriste u svim fazama životnog ciklusa jednog softverskog proizvoda. Dakle, primjenjuju se u svrhe opisivanja korisničkih zahtjeva, kreiranja statičkih analiza, kreiranja apstraktne predstave sistema, generisanja dokumentacije i kreiranja automatskih testova.

Definisanjem mapiranja između različitih modela moguće je vršiti transformacije modela koje takođe predstavljaju bitan aspekt paradigme MDSD. Transformacije se definišu na nivou meta-modela, a primjenjuju na nivou modela. Transformacije se takođe mogu posmatrati kao modeli, odnosno kao instance meta-modela koji ih opisuje. Prilikom razvoja predloženog namjenskog jezika korišćene su transformacije modela procesa u dokument FMEA.

3. IMPLEMENTACIJA NAMJENSKOG JEZIKA ZA MODELOVANJE GREŠAKA U PROIZVODNIM PROCESIMA

3.1. Apstraktna sintaksa

Meta-model namjenskog jezika za upravljanje greškama definiše njegovu apstraktnu sintaksu. Za razvoj meta-modela korišćen je EMF (engl. *Eclipse Modeling Framework*). EMF je radni okvir koji se koristi za kreiranje meta-model upotrebom jezika *Ecore* i za generisanje alata zasnovanog na meta-modelu. Kako sva ograničenja nije moguće definisati upotrebom jezika *Ecore*, za specifikaciju složenijih ograničenja korišćen je OCL (engl. *Object Constraint Languague*). Na slici 1. prikazan je meta-model jezika za modelovanje grešaka u proizvodnim procesima.

Model je korijenski koncept koji u sebi sadrži sve ostale koncepte. Unutar *Model*-a se definišu proizvodni procesi koji su modelovani sa *Manufacture* konceptom. Globalni obrađivač grešaka (*GlobalErrorHandler*) predstavlja zaseban proces koji služi za korigovanje grešaka. Svrha ovog koncepta je da pojednostavi modelovanje proizvodnog procesa u situaciji kada postoji više različitih grešaka koje se obrađuju na isti način. Dakle, i

Manufacture i *GlobalErrorHandler* su procesi i zbog toga nasljeđuju apstraktna klasu *Process*. Oba koncepta mogu da sadrže sljedeće komponente: procesne elemente (*ProcessElement*), veze (*Relationship*) i kontrolne aktivnosti (*Inspection*). *ProcessElement* je predstavljen apstraktnom klasom i njegova svrha jeste generalizacija dva elementa proizvodnog procesa – procesnih koraka i kapija.

Procesni korak (*ProcessStep*) služi za modelovanje tačno jednog zadatka u proizvodnom procesu, a kapija (*Gate*) za modelovanje kontrole toka izvršavanja. Kapije mogu biti tipa: PARALLELISM – omogućavaju kreiranje paralelnih tokova i DECISION – omogućavaju kreiranje odluke u kontroli toka. *Relationship* se koristi za modelovanje veza između procesnih elemanta, a *Inspection* za modelovanje kontrolnih aktivnosti koje otkrivaju greške.

3.2. Konkretna sintaksa

Za razvoj grafičke konkretne sintakse korišćen je *Sirius*. *Sirius* je okruženje koje služi za kreiranje grafičke sintakse jezika i omogućava automatsko generisanje alata za modelovanje.

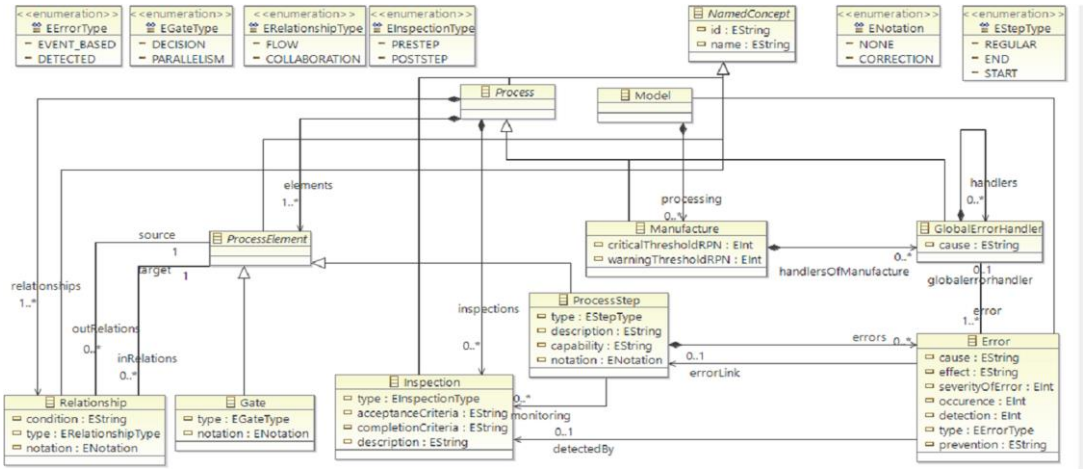
Za prikaz podataka iz modela proizvodnog procesa korišćene su tri različite dijagramske reprezentacije: *ModelRepresentation*, *ManufactureRepresentation* i *GlobalHandlerRepresentation*. *ModelRepresentation* omogućuje prikaz modela sa modelovanim proizvodnim procesima prikazanim u vidu grafičkih simbola. *ManufactureRepresentation* prikazuje unutrašnju strukturu svakog od modela proizvodnih procesa, dok *GlobalHandlerRepresentation* prikazuje modele procesa koji opisuju obradu grešaka nastalih u proizvodnim procesima. Sve tri reprezentacije imaju definisana mapiranja grafičkih simbola za elemente modela (tabela 1.), kao i alate za manipulaciju nad tim elementima. *ManufactureRepresentation* je jedina reprezentacija koja ima dva dodatna sloja: *Manufacture* i *QulaityAssurance*. Dijagrami kompleksnih proizvodnih procesa obuhvataju previše informacija što ih čini nepreglednim, te je svrha organizovanja komponenti kroz slojeve da omogući svakom korisniku da vidi samo one dijelove modela proizvodnog procesa koji su njemu neophodni. *Manufacture* sloj sadrži komponente koje su inženjeru proizvodnih procesa potrebne za modelovanje strukture proizvodnog procesa, dok *QualityAssurance* ima komponente potrebne inženjeru kontrole kvaliteta da modeluje potencijalne greške i kontrolne aktivnosti.

Na slici 2 prikazan je primjer modela proizvodnog procesa za proizvodnju majica. U primjeru su modelovane tri greške uzrokovane događajem i jedna detektovana greška. Detektovana greška se otkriva tokom izvršavanja kontrolne aktivnosti, dok se greška uzrokovana događajem desila u neočekivanim uslovima.

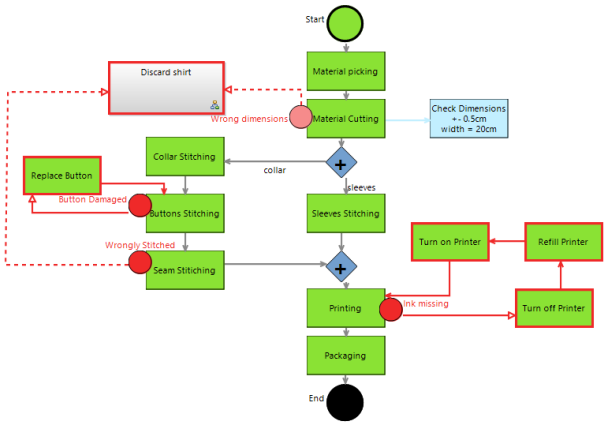
Greške sa nazivima *Wrong dimensions* i *Wrongly stiched* obrađuju se posredstvom globalnog obrađivača greške *Discard shirt*. Greške sa nazivima *Ink missing* i *Button Damged* obrađuju se lokalno – korišćenjem sekvence korektivnih procesnih koraka.

Tabela 1. *Grafički simboli svih elemenata modela*

Element	**Simbol**
Regularni procesni korak	
Korektivni procesni korak	
Regularna kapija odlučivanja	
Regularna kapija paralelizma	
Korektivna kapija odlučivanja	
Korektivna kapija paralelizma	
Detektovana greška	
Greška uzrokovana događajem	
Kontrolna aktivnost	
Globalni obrađivač greške	
Regularna veza između procesnih elemenata	
Veza između procesnkog koraka i kontrolne aktivnosti	
Veza između greške i korektivnog procesnog koraka	
Veza između greške i globalnog obrađivača greške	
Veza između dva korektivna procesna koraka	



Slika 1. *Meta-model namjenskog jezika za modelovanje grešaka u proizvodnim procesima*



Slika 2. *Primjer modela procesa za proizvodnju majica*

4. GENERISANJE DOKUMENTACIJE FMEA

Formalno kreiran model proizvodnog procesa upotrebom predloženog namjenskog jezika, moguće je iskoristiti za automatsko generisanje digitalne dokumentacije preporučene metodologijom FMEA.

Metodologija FMEA se koristi prilikom analize rizika u oblasti proizvodnih procesa. Rizik greške se posmatra iz tri aspekta: ozbiljnost greške u pogledu njenog uticaja na korisnika proizvoda, broj proizvoda koji imaju neki vid oštećenja na uzorku od jednog miliona proizvoda i vjerovatnoća uočavanja greške. Svaka greška sadrži atribute koji opisuju nabrojane aspekte. Atributima se dodjeljuje cjelobrojna vrijednost iz intervala od 1 do 10. Ukupni rizik greške ocjenjuje se na osnovu broja RPN (engl. *Risk Priority Number*) koji se računa množenjem pomenute tri vrijednosti [3]. Na nivou proizvodnog procesa projektant definiše dva praga vrijednosti za RPN koji svaku grešku klasifikuju u neki od tri opsega za procjenu rizika greške: nizak, srednji ili visok. Vrijednost RPN-a za pomenute opsege u ciljnoj dokumentaciji se redom prikazuje sa zelenom, žutom i narandžastom pozadinom. Na taj način projektantima je skrenuta pažnja na greške sa većim stepenom rizika kako bi izmenili proizvodni proces u cilju prevencije te greške ili ublažavanja njenog efekta. Izgenerisana dokumentacija za primjer modela sa slike 2. prikazana je na slici 3.

Process step	Potential failure mode	Potential effect(s) of error	Severity	Potential cause(s) of error	Occurance	Current process controls prevention	Current process controls detection	Detection	Current error handler	RPN
Buttons Stitching	Button Damaged	Time consuming	4	Invalid input material	3	Following work instruction	Error occured without control mechanisms	5	LOCAL: Starts with step: Replace Button	60
Seam Stitiching	Wrongly Stitched	Material Discarded	7	Machine excecuted wrong operation	1	Following work instrunctions	Error occured without control mechanisms	5	GLOBAL: Discard shirt	35
Printing	Ink missing	Time consuming	5	Bad monitoring	2	Following work instrunctions	Error occured without control mechanisms	2	LOCAL: Starts with step: Turn off Printer	20
Material Cutting	Wrong dimensions	Time loss, material loss	7	Operator mistake	8	Following work instructions	Check Dimensions : Visual control	9	GLOBAL: Discard shirt	504

Slika 3. *Dokumentacija FMEA za proizvodnju majica*

Generator koda za HTML tabelu sa sadržajem FMEA dokumenta parsira strukturu modela sve dok se ne dođe do *Error*-a .Vrijednosti svih atributa koncepta *Error* se potom čitaju i koriste za kreiranje sadržaja dokumenta FMEA.

5. ZAKLJUČAK

Namjenski jezik predložen u ovom radu omogućava kreiranje preciznih i formalnih modela grešaka u proizvodnji i procesa koji te greške otklanjaju. Izborom grafičke konkretne sintakse prilikom specifikacije jezika omogućeno je stvaranje okruženja koje je intuitivno krajnjim korisniku. Okruženje predloženo u ovom radu implementirano je tako da ga koriste dva tipa korisnika: inženjeri proizvodnih procesa i inženjeri kontrole kvaliteta. Inženjeri proizvodnih procesa koriste okruženje da kreiraju modele koji opisuju strukturu proizvodnog procesa sa tačno definisanim koracima proizvodnje. Tako definisan model sa jasno utvrđenim granicama procesa, inženjeri kontrole kvaliteta proširuju konceptima koji formalno opisuju kontrolne aktivnosti, pojavu i obradu grešaka. Implementiran je generator koji na osnovu modela proizvodnog procesa generiše dokumentaciju FMEA. Svrha dokumenta FMEA jeste da se u budućnosti izbjegnu pojave uočenih i predviđenih grešaka. To se postiže analizom prethodnih grešaka i uočavanjem kritičnih koraka proizvodnje koja predstavljaju područja visokog rizika u smislu posljedica koja ostavljaju na cijeli sistem.

Neka od mogućih proširenja rada su konkretizacija jezika na određenu granu industrije, lokalizacija okruženja, dodavanje tekstualne konkretne sintakse kao i implementacija generatora za druge tipove dokumentacija. Konkretizacijom jezika za određenu granu industrije modeli bi bili prilagođeniji logici proizvodnje iz tog sektora. Interfejs okruženja za modelovanje prilagođen je korisnicima sa engleskog govornog područja. Lokalizacija okruženja na druge jezike bi omogućila i korisnicima koji ne razumiju engleski jezik da efektivno kreiraju modele. Dodavanjem tekstualne konkretne sintakse, inženjerima sa programerskim iskustvom bi proces modelovanja bio ubrzan.

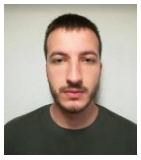
6. LITERATURA

[1] C. P. Ahire, A S. Relklar, "Correlating Failure Modes Effect Anlysis (FMEA) & Overall Equipment Effectiveness (OEE)", *Procedia Engineering* 38, ISSN 3482-3486, 2012, doi: 10.1016/j.proeng.2012.06.402

[2] M. Brambilla, J. Cabo, M. Wimmer, "Model-Driven Software Engineering in Practise"(Second edition), Morgan & Claypool Publishers, 2017

[3] M. Svingerova, M. Melichar, "Evaluation of Process Risks in Industry 4.0 Environment", *Proceedings of the 28th DAAAM International Symposium*, pp. 1021-1029, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-11-2, ISSN 1726-9679, 2017 doi:10.2507/28th.daaam.proceedings.142

Kratka biografija:



Đorđije Ivanišević rođen je u Nevesinju 1996. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Primjenjene računarske nauke i informatika odbranio je 2019.god.
kontakt: djordjij3_ivanis@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.39
DOI: https://doi.org/10.24867/10BE29Mrdjen

PRAĆENJE POKRETNIH OBJEKATA U VIDEU KORIŠĆENJEM PASIVNOG FILTRA I DETEKTORA VISOKIH PERFORMANSI

TRACKING OF MOVING OBJECTS IN VIDEO USING PASSIVE FILTER AND HIGH PERFORMANCE DETECTOR

Jelena Mrđen, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIČKO I RAČUNARSKO INŽENJERSTVO

Kratak sadržaj – *U ovom radu je razmatran problem praćenja pokretnih objekata u videu i implementirani su elementi sistema za praćenje koji su zasnovani na principu pasivnog filtriranja detekcija u svakom kadru video snimka. Korišćenjem DETRAC protokola kroz numeričke eksperimente određeni su optimalni parametri konfiguracije IOU algoritma sa YOLOv3 detektorom.*

Ključne reči: *Praćenje objekata; IOU algoritam; YOLOv3 detekor; DETRAC evaluacija.*

Abstract – *The paper describes the problem of tracking moving objects in video and implements system components based on passive filtering of detections in each frame. Through numerical experiments based on DETRAC evaluation protocol the best parameters for the configuration of IOU algorithm and YOLOv3 object detector were determined.*

Keywords: *Object tracking; IOU algorithm; YOLOv3 object detector; DETRAC evaluation protocol.*

1. UVOD

Praćenje pokretnih objekata u videu predstavlja jedan od klasičnih problema kojim se bavi kompjuterska vizija. Rešenje ovog problema podrazumeva detektovanje objekta od interesa u svakom kadru video sekvence, prepoznavanje njegovih karakteristika, kao i dalje praćenje datog objekta kroz celokupnu sekvencu. Shodno tome se problem praćenja objekata može podeliti na tri zasebna zadatka: detekciju objekata, koja podrazumeva uočavanje objekta od interesa i određivanje njegovog položaja u sceni, prepoznavanje objekta, koje daje uvid u konkretan tip i karakteristike objekta koji je detektovan, a zatim i praćenje objekata čiji je krajnji cilj određivanje putanje kretanja objekta kroz celokupnu video sekvencu.

Cilj ovog rada je razmatranje problema praćenja pokretnih objekata u videu i implementacija elemenata sistema koji omogućavaju praćenje, a koji su zasnovani na pasivnom filtriranju detekcija na nivou pojedinačnih kadrova. U ovom radu je predstavljen sistem za praćenje velikog broja objekata u videu. Ovaj sistem upotrebljava IOU (*engl. Intersection over Union)* algoritam za asocijaciju detekcija koji je zasnovan na pasivnom filtru dok je u cilju generisanja detekcija, koje mu daju osnovnu funkcionalnost, upotrebljen YOLOv3 (*engl. You Only Look Once, version 3)* detektor. Cilj ovog rada jeste, pored implementacije elemenata sistema koji omogućavaju praćenje pokretnih objekata, određivanje optimalnih parametara ovog sistema kroz evaluaciju njegovih performansi kao i analiza uticaja YOLOv3 detektora na rad algoritma. Evaluacija performansi sistema je izvršena pomoću DETRAC (*engl. DeTection And TraCking)* protokola za evaluaciju nad 60 zahtevnih video sekvenci koje prikazuju realne scene iz saobraćaja.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Brkljač, docent.

2. YOLOv3 DETEKTOR

Cilj algoritama za detekciju objekata upotrebom 2D podataka tj. slika u boji je lociranje i klasifikacija objekata od interesa na slikama ili video snimcima koji se mogu posmatrati kao sekvenca pojedinačnih statičnih slika. Kao rezultat algoritama za detekciju se dobija okvir koji obuhvata objekat od interesa, poznatiji pod nazivom *engl. Bounding box*, klasa objekta koji je detektovan kao i verovatnoća da objekat zapravo postoji u datom okviru, koji se takođe naziva i skor sigurnosti.

YOLOv3 je neuronska mreža za detekciju objekata koja omogućava procesovanje detekcija u realnom vremenu. Ova mreža predviđa četiri koordinate za svaki okvir koji definiše objekat. Ukoliko se predlog okvira objekta idealno poklapa sa stvarnim tj. anotiranim okvirom objekta (*engl. Ground Truth – GT)*, skor sigurnosti treba da bude jednak jedinici.

Ukoliko je ovo preklapanje manje od vrednosti 0.5 ta predikcija se zanemaruje. Takođe, mreža definiše klasu objekta koji je zahvaćen okvirom upotrebom klasifikacije u više klasa pomoću logističkog klasifikatora. Predikcija okvira objekata se vrši na tri različite skale i ulazi ove mreže mogu biti rezolucije 320x320, 416x416 i 608x608 piksela

Što je rezolucija ulazne slike veća, detekcije su tačnije tj. imaju veći skor sigurnosti i lokalizacija objekta u okviru je bolja, ali se vreme generisanja detekcija povećava u skladu sa povećanjem rezolucije ulaznih slika.

Za izvlačenje obeležja upotrebljava se hibridna *Darknet-53* mreža koja se sastoji od 53 sloja i trenirana je na *ImageNet* bazi podataka. Iako ova mreža u poređenju sa drugim kompleksnijim mrežama za detekciju objekata daje nižu mAP (*engl. mean Average Precision*) vrednost brzina izvršavanja ove mreže je znatno veća u odnosu na druge metode. Detaljan opis arhitekture i principa rada YOLOv3 mreže se može pronaći u [1].

3. IOU ALGORITAM ZA PRAĆENJE OBJEKATA U VIDEU

Ovaj rad je zasnovan na IOU algoritmu za praćenje objekata koji je predstavljen u radu [2]. Za svaki kadar video sekvence se prvobitno generišu detekcije a zatim se primenjuje algoritam za asocijaciju detekcija. Upotreba detektora velike preciznosti i video snimaka koji imaju veliki broj kadrova u sekundi značajno olakšava proces praćenja objekata. IOU algoritam polazi od pretpostavke da detektor za svaki objekat koji se prati daje odgovarajuće detekcije kao i da se javlja veoma mali broj promašenih detekcija tj. lažnih negativa.

Takođe se pretpostavlja da detekcije istog objekta u susednim kadrovima imaju veliki IOU indeks, što je uobičajeno za video sekvence koje imaju veliki broj kadrova u sekundi. Ukoliko su sve pretpostavke ispunjene, praćenje objekata u se može izvršiti bez izvlačenja obeležja objekata koji se prate. IOU algoritam ostvaruje praćenje objekata asociranjem detekcija na osnovu IOU indeksa trenutne detekcije i detekcije iz prethodnog kadra, ukoliko je taj indeks veći od praga σ_{IOU}. Sve detekcije koje nisu dodeljene trenutnoj putanji započinju novu putanju, dok se sve putanje koje nemaju novu asociranu detekciju smatraju završenim. Performanse IOU algoritma su unapređene filtriranjem svih putanja koje su kraće od vrednosti t_{min} kao i onih putanja koje nemaju nijednu detekciju čiji je skor sigurnosti veći od praga σ_h. Kratke putanje se uklanjaju jer dovode do lažnih pozitiva.

Zahtev da svaka putanja ima barem jednu detekciju visokog skora sigurnosti obezbeđuje da putanja pripada pravom objektu od interesa dok istovremeno koristi detekcije malog skora sigurnosti u cilju generisanja kompletnih putanja objekata.

Ulaz u algoritam čini skup detekcija objekata koji se pojavljuju u kadrovima video sekvence, što znači da je pre primene ovog algoritma neophodno prikupiti detekcije svih objekata čije putanje želimo pratiti.

IOU algoritam je predstavljen slikom 1 na kojoj D^f označava detekcije u kadru f, d_j označava *j*-tu detekciju u tom kadru, T_a označava aktivne putanje, T_f završene putanje dok F označava ukupan broj kadrova u video sekvenci.

Prilikom incijalizacije se vrši filtriranje detekcija čiji je skor sigurnosti manji od σ_l praga čime se, pre samog izvršavanja algoritma, vrši eliminacija detekcija malih skorova sigurnosti koje bi potencijalno dovele do lažnih pozitiva. Sveukupna kompleksnost ovog algoritma je veoma mala u odnosu na druge popularne sisteme za praćenje objekata.

Kako ovaj algoritam ne koristi nikakve vizuelne informacije, može se posmatrati kao jednostavan proces filtriranja na nivou detekcija. Ovo znači da je pri online upotrebi IOU algoritma za praćenje, u kombinaciji sa detektorom visokih performansi, računska zahtevnost ovog algoritma zanemarljiva.

Ukoliko se algoritam izvršava oflajn, sa prethodno procesovanim detekcijama, se može postići izvršavanje od 100 kadrova u sekundi (*engl. Frames per second – fps)*.

Takođe je bitno napomenuti da se, zahvaljujući brzini ovog algoritma, na ovu metodu mogu dodati i druge komponente za praćenje objekata, [1].

IOU ALGORITAM ZA PRAĆENJE OBJEKATA

ULAZ:
$D = \{D_0, D_1, ..., D_{F-1}\} = \{\{d_0, d_1, ..., d_{N-1}\}, \{d_0, d_1, ..., d_{N-1}\}, ...\}$
INICIJALIZACIJA:
$T_a = \emptyset, T_f = \emptyset$
$D = \{\{d_i | d_i \in D_j, d_i \geq \sigma_l\} | D_j \in D\}$
FOR f = 0 to F :
  FOR $t_i \in T_a$:
    IF $d_{best} = d_j$ ako je $max(IOU(d_j, t_i)), d_j \in D_f$
      $IOU(d_{best}, t_i) \geq \sigma_{IOU}$:
      dodaj d_{best} u t_i
        izbaci d_{best} iz D_f
      ELSE:
        IF najveći_skor $(t_i) \geq \sigma_h$ and $len(t_i) \geq t_{min}$:
          dodaj t_i u T_f
          izbaci t_i iz T_a
  FOR $d_j \in D_t$:
  započni novu putanju t sa d_j i dodaj je u T_a
FOR $t_j \in T_A$:
  IF najveći_skor $(t_i) \geq \sigma_h$ and $len(t_i) \geq t_{min}$:
    dodaj t_i u T_f
return T_f

Slika 1. *IOU algoritam za praćenje objekata u videu [1]*

4. DETRAC PROTOKOL ZA EVALUACIJU

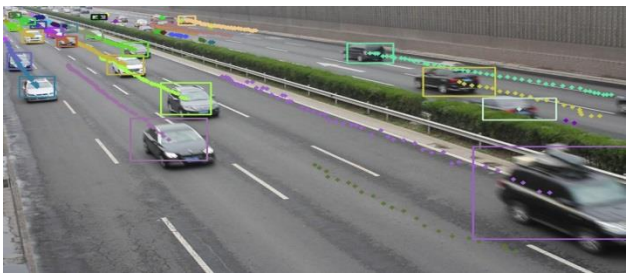
Evaluacija performansi sistema za praćenje objekata je neophodna radi procene uspešnosti njegovog rada. Takođe je veoma značajna kao metoda koja omogućava poređenje različitih sistema za praćenje i njihovu standardizaciju. Većina sistema za evaluaciju podrazumeva zasebnu evaluaciju performansi sistema za detekciju objekata i sistema za praćenje objekata. Sistemi za praćenje objekata se obično evaluiraju upotrebom iste metode za detekciju objekata, a samim tim i istih detekcija. Iako je ova strategija opšte prihvaćena u literaturi i daje koristan uvid u sisteme za praćenje velikog broja objekata u video sekvencama, obično je nedovoljna za potpunu analizu ovih sistema. Konkretno, veoma je bitno razumeti uticaj performansi detektora na celokupan sistem za praćenje objekata. Ovo je moguće ostvariti primenom sistema za evaluaciju koji zajedno analizira rad detektora objekata i sistema za praćenje objekata, [3]. Jedan od ovakvih sistema jeste DETRAC protokol za evaluaciju. DETRAC protokol za evaluaciju je zasnovan na DETRAC skupu podataka koji se sastoji od 100 video sekvenci. Sve sekvence su snimane sa frekvencijom od 25fps pri čemu rezolucija svakog pojedinačnog kadra iznosi 960x540 piksela. Postoji više od 140 hiljada kadrova, 8,250 ručno anotiranih motornih vozila i 1.21 miliona anotiranih okvira objekata u DETRAC skupu podataka. DETRAC skup podataka je podeljen na skup za obuku i skup za testiranje, koji se sastoje od 60 i 40 video sekvenci, respektivno.

U osnovne metrike za evaluaciju peformansi algoritama za asocijaciju detekcija spadaju, [3]: primarno praćeni objekti (*engl. Mostly Tracked – MT*), primarno promašeni objekti (*engl. Mostly Lost – ML*), promene identiteta (*engl. Identity Switches - IDS*), fragmenti trajektorija objekata (*engl. Fragmentations of Target trajectories – FM*), lažni pozitivi (*engl. False positives – FP*), lažni negativi (*engl. False negatives – FN*), tačnost (*engl.*

Multi-Object Tracking Accuracy - MOTA) i preciznost (*engl. Mult- Object Tracking Precision - MOTP*). FP metrika predstavlja broj izlaza algoritma za asocijaciju detekcija koji predstavljaju lažne pozitive dok FN predstavlja broj objekata koji su promašeni od strane bilo koje trajektorije u svakom kadru tj. broj lažnih negativa. IDS metrika opisuje koliko puta se odgovarajuća oznaka identiteta objekta koji se prati, tj. njegove trajektorije, promenila. FM definiše koliko puta je trajektorija objekta koji se prati prekinuta. IDS i FM metrike definišu tačnost praćenih trajektorija. ML i MT metrike opisuju procenat praćenih trajektorija koji je manji od 20% i veći od 80% ukupne anotirane trajektorije objekta, respektivno.
Na osnovu opisanih osnovnih metrika koje se upotrebljavaju u cilju evaluacije performansi algoritama za asocijaciju detekcija se za sve sekvence skupa podataka izračunavaju DETRAC MOT metrike. DETRAC MOT metrike uzimaju u obzir rad detektora i rad algoritma za asocijaciju detekcija i definisane su u zavisnosti od krive preciznosti-odziva detektora (*engl. precision-recal*), u zajedničkoj oznaci PR, i uvode PR-MOTA, PR-MOTP, PR-MT, PR-ML, PR-IDS, PR-FM, PR-FP i PR-FN metrike. Performanse algoritama za praćenje objekata se rangiraju spram PR-MOTA vrednosti tj. veća PR-MOTA vrednost nagoveštava bolje performanse algoritma. Ukoliko su PR-MOTA vrednosti dva algoritma jednake, dalje rangiranje se vrši spram PR-MOTP vrednosti.

5. REZULTATI

Krajnji cilj ovog istraživanja je određivanje optimalnih parametara konfiguracije IOU algoritma i YOLOv3 detektora uz upotrebu DETRAC protokola za evaluaciju, po uzoru na rad [1]. U [1] su upotrebljene četiri konfiguracije IOU algoritma i njegove performanse su testirane uz pomoć četiri različita detektora a to su R-CNN, CompACT, EB i ACF detektori. Svaka od ove četiri konfiguracije je testirana za različite opsege parametara σ_{IOU}, σ_h i t_{min}. Po uzoru na [1] je konfiguracija IOU algoritma i YOLOv3 detektora takođe testirana za različite vrednosti ovih parametara u cilju pronalaženja njihovih optimalnih vrednosti. Bitno je naglasiti da su svi detektori upotrebljeni u [1] prvobitno obučeni nad DETRAC skupom za obuku nad kojim su takođe određene i vrednosti njihovih optimalnih parametra. Upotrebljeni YOLOv3 detektor je obučen nad COCO skupom podataka i prepoznaje 80 klasa objekata. Kako je neophodno prvobitno prikupiti detekcije svih objekata koje želimo pratiti, a u DETRAC skupu podataka su od interesa samo klase koje definišu motorna vozila, prilikom generisanja detekcija za DETRAC skup za obuku upotrebom YOLOv3 detektora su uzete u obzir samo objekti klasa '*car', 'bus'* i *'truck'*. Svaki pojedinačan kadar video sekvence, koji ujedno predstavlja i ulaz YOLOv3 mreže, je smanjen na rezoluciju 608x608 piksela. Primer rada konfiguracije IOU algoritma i YOLOv3 detektora na sekvenci DETRAC skupa za obuku je dat na slici 2.
Nakon generisanja detekcija izvršena je evaluacija performansi konfiguracije IOU algoritma i YOLOv3 detektora po DETRAC protokolu. DETRAC protkol za evaluaciju integrisan je u *eng. MATLAB* okruženju i može se preuzeti kao celokupan sistem sa [4].



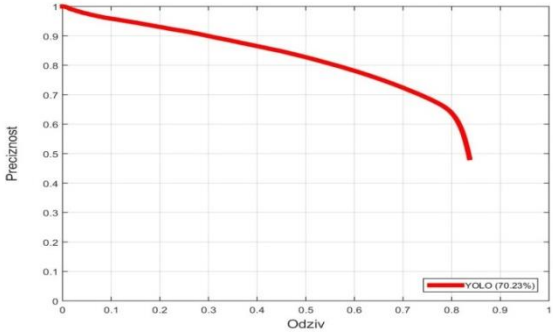
Slika 2. *Vizuelizacija rada IOU algoritma i YOLOv3 detektora*

U Tabeli 1 dati su opsezi parametara koji su upotrebljeni u cilju određivanja optimalnih vrednosti parametara konfiguracije IOU algoritma i YOLOv3 detektora.

Tabela 1. *Opsezi parametara za svaki detektor*

Detektor	σ_{IOU}	σ_h	t_{min}
YOLOv3	0,4 – 0,7	0,4 – 0,9	1 - 4

Kako bi se izvršila evaluacija sistema je neophodno prvenstveno evaluirati peformanse detektora a zatim izvršiti evaluaciju performansi algoritma za asocijaciju detekcija. Kao rezultat evaluacije performansi detektora se dobija PR kriva i prosečna preciznost detektora za dati skup podataka, dok se kao krajnji rezultat evaluacije performansi algoritma za asocijaciju detekcija dobijaju DETRAC MOT metrike opisane u poglavlju 4. Na Slici 3 je prikazana PR kriva YOLOv3 detektora nad DETRAC skupom za obuku. Njegova prosečna preciznost nad ovim skupom iznosi 70.23%.



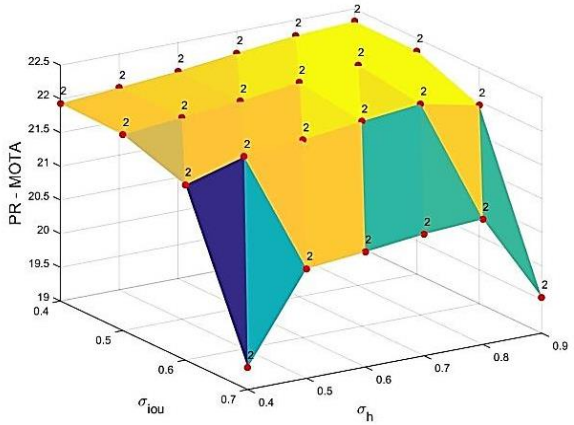
Slika 3. *PR kriva YOLOv3 detektora nad DETRAC skupom za obuku*

Tabela 2. Rezultati evaluacije konfiguracije IOU algoritma i YOLOv3 detektora

Detektor	YOLOv3
σ_{IOU}	0,4
σ_h	0,9
t_{min}	2
PR-MOTA	22,32%
PR-MOTP	29,06%
PR-MT	15,42%
PR-ML	22,85%
PR-IDS	3108,81
PR-FM	3224,52
PR-FP	9798,44
PR-FN	160200,22
Brzina u FPS	13,828

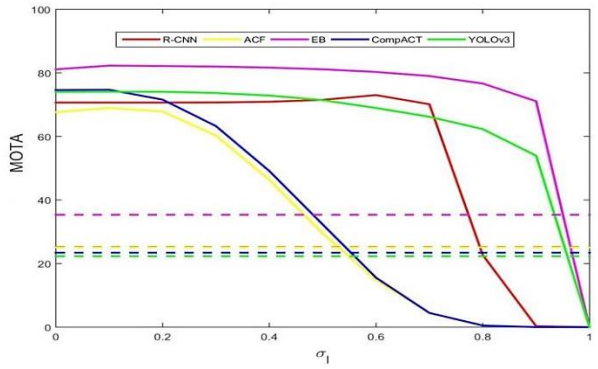
U Tabeli 2 date su vrednosti optimalnih parametara IOU algoritma za YOLOv3 detektor kao i vrednosti DETRAC MOT metrika koje su ostvarene njihovom upotrebom. Konfiguracija IOU algoritma sa YOLOv3 detekcijama daje najveću, a samim tim i najbolju, PR-MOTA vrednost od 22.32% za kombinaciju parametara $\sigma_{IOU} = 0.4$, $\sigma_h = 0.9$ i $t_{min} = 2$. Na Slici 4 je prikazan grafik koji omogućava poređenje PR-MOTA vrednosti dobijenih za različite kombinacije ovih parametara.

Za svaku kombinaciju pragova σ_{IOU} i σ_h određena je vrednost t_{min} koja daje najbolje PR-MOTA vrednosti. Tačke označene crvenom bojom predstavljaju rezultat evaluacije performansi algoritma dok broj iznad njih označava optimalnu vrednost parametra t_{min}. Sa ove slike se može uočiti da se najbolje PR-MOTA vrednosti dobijaju ukoliko je parametar σ_{IOU} postavljen na vrednost od 0.4. Sa daljim rastom parametra σ_{IOU} dolazi do pada PR-MOTA vrednosti dok sa rastom parametra σ_h dolazi do suptilnog rasta PR-MOTA vrednosti.



Slika 4. *PR-MOTA vrednosti za različite kombinacije parametara σ_{IOU} i σ_h , predložene konfiguracije IOU algoritma i YOLOv3 detektora*

Na slici 5 prikazane su MOTA vrednosti koje se dobiju upotrebom optimalnih parametara konfiguracije IOU algoritma i YOLOv3 detektora za različite vrednosti praga σ_l. Radi lakšeg poređenja rezultata su na ovoj slici prikazane i MOTA vrednosti za različite detektore upotrebljene u [1]. Isprekidane linije predstavljaju odgovarajuće PR-MOTA vrednosti.



Slika 5. *MOTA vrednosti za različite vrednosti praga sigma σ_l razmatrane konfiguracije*

Sa Slike 4 može se uočiti da se koleno krive, koja predstavlja promenu MOTA vrednosti u skladu sa promenom parametra σ_l za YOLOv3 detektor, javlja kada je parametar σ_l izmedju vrednosti 0.8 i 0.9. Ovo naglašava da sa porastom σ_l preko ovih vrednosti dolazi do pada performansi sistema. U poređenju sa krivama ostalih detektora iz [1] se može uočiti da je PR-MOTA vrednost za konfiguraciju IOU algoritma i YOLOv3 detektora manja u odnosu na konfiguracije iz rada [1]. Manje PR-MOTA vrednosti mogu se pripisati činjenici da YOLOv3 detektor nije prethodno obučen nad DETRAC skupom podataka, nad kojim su takođe određene i vrednosti njegovih optimalnih parametara, dok su svi detektori upotrebljeni u [1] prvobitno obučeni nad ovim podacima.

6. ZAKLJUČAK

Ovaj rad kombinuje jednostavan algoritam za praćenje objekata u videu i detektor objekata visokih performansi koji omogućava da se praćenje objekata vrši u realnom vremenu. Iako predstavljeni rezultati ukazuju na mogućnost primene predloženog sistema za praćenje, takođe postoji mogućnost značajnog unapređenja upotrebljene konfiguracije IOU algoritma i YOLOv3 detektora. Naime, kako upotrebljeni YOLOv3 detektor nije prethodno obučavan nad DETRAC skupom podataka, dalje unapređenje predloženog sistema za praćenje bi se moglo ostvariti ukoliko bi se detektor prvobitno obučio nad pomenutim skupom.

Ovo unapređenje bi potencijalno moglo povećati tačnost celokupnog sistema a samim tim dovesti do boljih rezultata. Takođe, u cilju testiranja rada ove konfiguracije i potencijalnog poboljšanja prikazanih rezultata bi trebalo razmotriti optimalnu rezoluciju slika koje se dovode na ulaz YOLOv3 detektora prilikom generisanja detekcija za IOU algoritam.

4. LITERATURA

[1] J. Redmon and A. Fahradi, *"YOLOv3: An Incremental Improvement"*, 2018.

[2] E. Bochinski, V. Eiselein and T. Sikora, *"High-Speed Tracking-by-Detection Without Using Image Information"*, 2017.

[3] L. Wena et al., *"*UA-DETRAC: A *New Benchmark and Protocol for Multi-Object Detection and Tracking"*, 2020.

[4] *"The UA-DETRAC Benchmark Suite*", Detrac-db.rit.albany.edu, 2020. [Online]. Dostupno na: http://detrac-db.rit.albany.edu/download.[Pristupljeno: 08- Sep- 2020].

Kratka biografija:

Jelena Mrđen rođena je u Novom Sadu 1996. god. Master rad na Fakultetu tehničkih nauka iz oblasti Energetika, elektronika i telekomunikacije – Obrada signala odbranila je 2020.god. Osnovne akademske studije završila je 2019. godine na studijskom programu Biomedicinsko inženjerstvo.
kontakt: jelenamrdjen3@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE30Aleksic

IMPLEMENTACIJA MIKROSERVISNE ARHITEKTURE POMOĆU SPRING, JENKINS I OPENSHIFT PLATFORME

IMPLEMENTATION OF MICROSERVICE ARCHITECTURE USING SPRING, JENKINS AND OPENSHIFT PLATFORM

Isidora Aleksić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Tema rada jeste analiziranje osnovnih koncepata mikroservisne arhitekture web aplikacija. Posebno je proučen Jenkins alat za kontinuiranu integraciju, kao i OpenShift platforma za implementaciju i eksploataciju softvera. Komponente prikazane arhitekture su višeslojno organizovane a navedeni su i principi organizacije komponenti.*

Ključne reči: *Mikroservisi, API, deploy*

Abstract – *This paper analyzes the basic concepts of microservice web application architecture. The Jenkins tool for continuous integration, as well as the OpenShift platform for software deployment, are presented in more detail. Components of this architecture are organized into multiple layers, while component organization principles are also defined.*

Keywords: *Microservices, deployment, software architecture*

1. UVOD

Mikroservisi se sastoje od malih, samostalnih servisa koji rade zajedno [1]. Cilj ovih pojedinačnih servisa je da razlože kompleksnu aplikaciju na manje jednostavnije jedinice koje će biti grupisane na osnovu njihovih zajedničkih funkcionalnosti (*Single Responsibility Principle*). Komunikacija između ovih servisa se obezbeđuje uz pomoć API-a (*Application Programming Interface*).

Aplikacije koje su izgrađene korišćenjem mikroservisne arhitekture imaju određene karakteristike. Aplikacija je podeljena na više komponenti, od kojih svaka vrši određenu funkciju. Svaka komponenta je napravljena tako da predstavlja nezavisnu uslugu koja se može menjati, ažurirati i brisati bez ometanja ostatka aplikacije. U projektima, svaki mikroservis ima jasnog vlasnika. Svaki tim ima jasan, nepreklapajući niz odgovornosti. U praksi neke usluge mogu doći sa drugih strana putem API-ja i možda neće biti u potpunosti vidljive vlasniku aplikacije. Osnovno pravilo prilikom izmene servisa jeste deploy tih izmena neće uticati ni na jedan drugi servis [1]. Potrebno je na odgovarajući način sistem razdvojiti u samostalne servise kako bismo obezbedili aplikaciju koja će biti jednostavna za dalji razvoj. Mnoge organizacije su ustanovile da korišćenjem mikroservisne arhitekture omogućava brže isporučivanje softvera.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.

Mnoge organizacije su ustanovile da korišćenjem mikroservisne arhitekture omogućava brže isporučivanje softvera i brže prihvatanje novijih tehnologija.

Mikroservisi nam daju znatno više slobode reagovanja i donošenja različitih odluka.

Ovo nam omogućava brži odgovor na neizbežne promene. Kroz primere i implementaciju analizirane su funkcionalnosti takve arhitekture.

2. KORIŠĆENE TEHNOLOGIJE I TERMINOLOGIJE

2.1 Spring

Spring [2] je Java platforma koja pruža širok opseg mogućnosti za podršku prilikom razvoja veb aplikacija. *Spring* rukuje infrastrukturom, tako da programer može usmeriti svoj fokus na razvoj aplikacije. *Spring* nudi kontejner koje se odnose na *Spring appliation context* uz pomoć kojeg se kreiraju i rukuju aplikacione komponente koje čine sistem [3].

Spring radi sa *JavaBean* klasama gde su sva polja privatna (*private*), serijalizabilna, imaju podrazumevani konstruktor kao i get/set metode pomoću kojih se pristupa podacima. *Spring* je osmišljen da bude dopuna Java standarda za poslovne aplikacije.

Spring predstavlja bolju alternativu za kompleksni J2EE standarda, naročito deo koji se tiče rada sa JavaBeans.

2.2 Swagger

Swagger [4] je jednostavan alat za upotrebu API programera (timove i pojedince). Omogućava razvoj tokom celog životnog ciklusa API-ja, od dizajna i dokumentacije, do testiranja i primene. Swagger se sastoji od mešavine otvorenog koda, besplatnih i komercijalno dostupnih alata koji omogućavaju da se naprave API-jevi.

Swagger omogućava da opišemo strukturu API-ja tako da ih mašine mogu čitati. Sposobnost API-ja da opišu sopstvenu strukturu je osnova u Swaggeru. Čitajući strukturu API-ja može se automatski napraviti interaktivna API dokumentacija.

Takođe može automatski generisati klijentske biblioteke za API na mnogim jezicima i tražiti druge mogućnosti poput automatizovanog testiranja. Swagger to radi tako što traži od API-ja da vrati YAML ili JSON koji sadrži detaljan opis celokupnog API-ja.

Specifikacija traži da se uključi informacije poput operacije koje API podržava, koji su parametri API-ja i šta vraća i da li je potrebna neka autorizacija.
Swagger spec. možemo ručno da napišemo ili ga možemo automatski generisati iz anotacija u našem izvornom kodu.

2.3 Jenkins

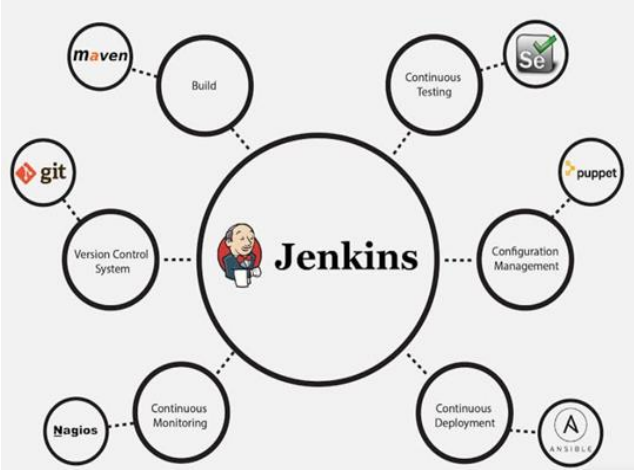
Jenkins [5] je server za automatizaciju otvorenog koda koji se može koristiti za automatizaciju svih vrsta zadataka koji se odnose na izgradnju, testiranje i isporuku ili upotrebu softvera.

Može se instalirati preko sistemskih paketa, Docker-a ili se može čak samostalno pokretati na bilo kojoj mašini sa instaliranim Java Runtime Environment (JRE).

Jenkins postiže kontinuiranu integraciju uz pomoć dodataka. Dodaci omogućavaju integraciju različitih DevOps faza. Ukoliko želimo da integrišemo određeni alat, moramo prvo da instaliramo dodatak za taj alat.

Za na primer: Git, Maven 2 projekat, Amazon EC2, HTML izdavač itd., slika 1.

Najosnovniji kontinuirani *delivery pipeline* imaće, u najmanju ruku, tri faze koje bi trebale definisati u Jenkinsfile-u: Build, Test i Deploy.



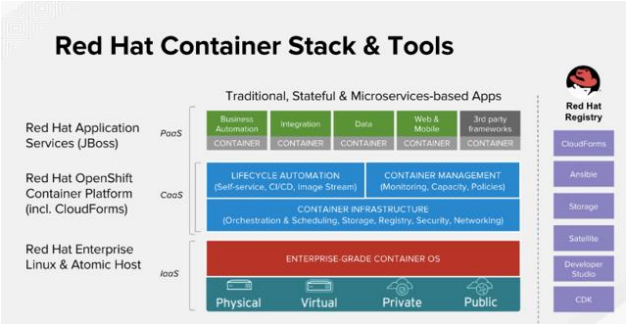
Slika 1. *Jenkins integriše različite DevOps faze*

2.4 Openshift container platform

Openshift container platform [6] treba da omogući programerima da se usredsrede na svoj kod koristeći postojeće alate i poznate tokove rada, uključujući CI/CD.

Konzistentna kontejnerska platforma veoma je važna za Ops timove koji žele da sa jednog mesta, sa potpunom vidljivošću od nivoa aplikacije do kontejnera, OS-a, virtualizacije i hardverskog sloja, upravljaju tradicionalnim mikroservisima, slika 2.

Ops timovi (operativni timovi) takođe brinu o bezbednosti, modularnosti, kontroli pristupa i polisama.



Slika 2. Prikaz Red Hat Container Stack i alata

3. MIKROSERVISNA ARHITEKTURA

Mikroservisi su tehnika razvoja softvera - varijanta servisno-orijentisanog arhitekturalnog (SOA) stila koji struktuira aplikaciju kao kolekciju labavo povezanih servisa.

Cilj razvoja servisa u mikroservisnoj arhitekturi je da servisi budu što manji, optimalni i fokusirani isključivo na svoj domen. Servisi su alternativno rešenje monolitnim sistemima koji u sebi sadrže jednu aplikaciju zaduženu za kompletnu interakciju sa korisnikom i bazom podataka. Sa vremenom, kako se radi na aplikaciji, povećava se količina programskog koda i time promene postaju sve teže i raste kompleksnost. Iako se pri razvoju koristi primeri za odvajanje delova aplikacije te za strukturiranje koda unutar programske komponente, te komponente postaju sve komplikovanije. Desi se i da funkcije i metode povezuju više domena odjednom, što otežava da se implementira nova funkcionalnost ili ukloni greška. Ukoliko se pri razvoju monolitne aplikacije ima na umu da programski kod čini koheziju, odnosno da su zajednički koncepti čvrsto povezani, tada je prelazak iz postojeće monolitne aplikacije u mikroservisnu arhitekturu nešto jednostavniji.

Ukoliko servis deli previše svoje funkcionalnosti, korisnici servisa mogu da postanu zavisni o internoj logici toga servisa. Servisi koriste aplikacijski programski interfejs (*Application programming interface-API*) za komuniciranje i sarađivanje sa drugim servisima putem mreže. Svaki servis pruža svoj interfejs drugim servisima pa odabir tehnologije interfejsa nije trivijalan zadatak.

Pošto postoji nezavisnost između servisa, potpuno je svejedno koje tehnologije koristi servis, sve dok ti servisi imaju „zajednički jezik“, API. To omogućava odabir odgovarajuće tehnologije za svaki zadatak, skraćivanje vremena razvoja i poboljšane performanse aplikacije.

Servisi mogu biti implementirani pomoću različitih programskih jezika, baza podataka i softverskih okruženja. To omogućava svakom servisu da bude ponovo izgrađen, ponovo isporučen i upravljan nezavisno i bez promene njihovih potrošača (*consumer*).

Kod tradicionalnih monolitnih aplikacija stvari se komplikuju ako tehnologija koja se koristi u celoj aplikaciji nije najbolji alat za neki zadatak ili ta funkcionalnost uopšte nije podržana. U tom slučaju potrebno je napraviti svoju implementaciju potrebnog

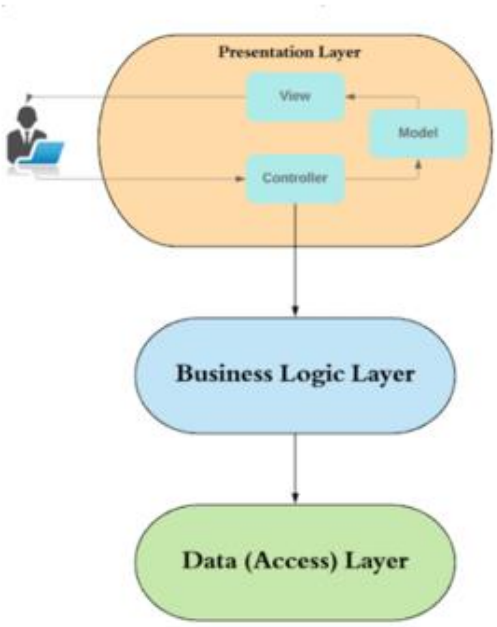
algoritma u korišćenoj tehnologiji ili spajati dve tehnologije, što otežava daljnji razvoj [1].

3.1 Višeslojna arhitektura

Kod razvoja web aplikacija često se sama aplikacija strukturira u više delova ili slojeva. Primer takve arhitekture je višeslojna ili N-slojna arhitektura, gde N označava broj slojeva. Kada se priča o višeslojnoj arhitekturi, najčešće se misli na troslojnu arhitekturu koja aplikacijske komponente razdvaja na prezentacijski, aplikacijski i sloj podataka, slika 3.

Prezentacijski sloj je u direktnom kontaktu sa korisnikom, prikazuje korisniku interfejs i informacije i održava interakciju sa korisnikom.

Za komunikaciju sa korisnikom, prezentacijski sloj koristi preglednik korisnika. Sloj koji se nalazi u sredini naziva se aplikacijski sloj koji je zadužen za upravljanje aktivnostima koje određena web aplikacija treba da obavlja na osnovu interakcije korisnika. Taj sloj je takođe zadužen za komunikaciju prezentacijskog sloja i sloj podataka i njime upravlja tokom aplikacije. Treći sloj troslojne arhitekture je sloj podataka. Sloj podataka sadrži podatke i logiku pristupa i čuvanje podataka. [7].



Slika 3. *Prikaz višeslojne arhitekture*

3.2 Servisno modeliranje

Posmatrajući usluge u mikroservisnoj arhitekturi, može se reći da su njihove zajedničke karakteristike slaba povezanost (loose coupling) i velika kohezija

Slabo povezivanje (*loose coupling*) u mikroservisnoj arhitekturi koristi se kako bismo imali mogućnost menjanja jednog servisa bez uticaja ili izmena drugog servisa te je to jedna od najvažnijih prednosti servisa. Slabo povezani servisi ne znaju puno jedni o drugima, čak i kada neposredno sarađuju.

Slabo povezani servisi nemaju puno informacija jedni o drugima, čak i kada neposredno sarađuju. Ako se pri razvoju ne uzme u obzir da se sva komunikacija odvija kroz mrežu, tada može doći do problema sa performansama, pa je iz tog razloga važno pripaziti na obim razmene poruka između servisa. Povećana složenost komunikacije između dva servisa dovodi i do čvršće povezanosti između njih, što je isto suprotno od onoga što se želi postići.

Cilj visoke kohezije (High cohesion) je grupisanje logike programa u povezane celine. Svaka celina sadrži logiku i ponašanje koje je usko vezano za njen kontekst, dok je sve ostalo što nije direktno vezano za taj kontekst, u drugoj celini.

Pronalaženjem granica celina obezbeđuje da je ponašanje povezano sa jednim kontekstom u jednoj celini, a drugačije ponašanje u drugoj celini, drugim rečima, granice celina kreiraju se prema njihovim kontekstima. Te se celine pretvaraju u servise pa se kod zahteva za promenom ponašanja pojedinog servisa ono promeni samo na jednom mestu, usled čega je servis znatno brže spreman za produkcijsko okruženje (production environment).

Ukoliko se funkcionalnost proteže kroz više servisa, nekoliko servisa će se morati izmeniti da bi se promenila funkcionalnost, dok će se oni možda u isto vreme proizvoditi. Menjanjem većeg broja servisa odjednom, drastično se povećava rizik od grešaka te je sam postupak izmene puno teži, s obzirom da se izvršava na nekoliko servisa. [7].

4. ZAKLJUČAK

Mikroservisna arhitektura popularan je izbor za implementaciju web aplikacija. Detaljno su analizirani i opisani osnovni koncepti mikroservisne arhitekture i tehnologije, koje su korišćenje u toku izrade aplikacije. Mikroservisna arhitektura omogućava fleksibilnost koja je potrebna za prilagođavanje rastućem broju korisnika Internet i Web aplikacija kao i broju tehnologija i alata dostupnih za razvoj istih. Cilj je pojednostavljenje svakog servisa i implementacionih rešenja koja neće zavisiti od tehnologije koja se trenutno koristi.

Neophodno je upoznati se sa domenom monolitne aplikacije kako bi se mogao pojednostaviti veliki broj tesno povezanih funkcionalnosti u samostalne servise. Tek tada se možemo odlučiti za mikroservisnu arhitekturu kao moguće rešenje. Postoji dosta pogodnih razloga za implementaciju web aplikacije uz pomoć mikroservisa. Meutim, razvojni proces takvih aplikacija je komplikovaniji i traži više angažovanja, koordinacije, znanja i rada na projektu. Javlja se i problematika rada sa velikim brojem alata i tehnologija u isto vreme.

Ukoliko je jedan od osnovnih zahteva web aplikacije skalabilnost, tj. podrška za veliki broj korisnika ili podrška za veliki broj tehnologija koje aplikacija koristi, tada je mikroservis pogodno iskoristiti kao softversko rešenje.
Održavanje i razvoj web aplikacije zasnovane na mikroservisima ne mora biti teže od razvoja aplikacije u jednostavnijem skupu tehnologija.

5. LITERATURA

[1] Sam Newman, Building Microservices, O'Reilly Media, 2015.

[2] Spring Framework Documentation. Dostupno na: http://docs.spring.io/springframework/docs/current/spring-framework-reference/html/index.html

[3] Craig Walls, Spring in Action, Manning Publication Co., 2018.

[4] Swagger Documentation. Dostupno na: https://swagger.io/docs/specification/about/

[5] Jenkins Documentation. Dostupno na: https://www.jenkins.io/doc/

[6] Openshift Documentation. Dostupno na: https://docs.openshift.com/container-platform/3.3/architecture

[7] Chapter 4 Layered Architecture for Web Applications, 2018

Kratka biografija:

Isidora Aleksić rođena je u Novom Sadu 1995. god. Master rad na Fakultetu tehničkih nauka iz oblasti Računarstva i automatika – Elektronsko poslovanje odbranila je 2020.god.
kontakt: isidoraxaleksic@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.31
DOI: https://doi.org/10.24867/10BE31Spica

UTICAJ DUGAČKIH VODOVA NA PRENOSNE MREŽE

INFLUENCE OF LONG OVERHEAD LINES ON TRANSMISSION NETWORKS

Sanja Špica, Dejan Jerkan, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je razvijen MATLAB Simulink model uprošćenog elektroenergetskog sistema (EES-a), kojim se modeluje ponašanje dugačkih vodova u visokonaponskoj prenosnoj mreži. Date su teorijske osnove mehanizma uspostavljanja napona i struja duž dugačkih vodova pomoću tzv. jednačina telegrafičara. Analiziran je uticaj dužine voda na fazne stavove napona i struja na njegovom početku, odnosno kraju, kao i bilans aktivnih, odnosno reaktivnih snaga duž voda.*

Ključne reči: *Prenosna mreža, MATLAB Simulink, Nadzemni vodovi, Bilans snaga, Fazni pomeraj*

Abstract *The paper presents MATLAB Simulink model of a simplified power system, which models the behavior of long overhead lines in a high-voltage transmission network. The theoretical basis of the mechanism of establishing voltages and currents along overhead lines are given by means of the so-called telegrapher's equations. The influence of overhead line length on the phase shift of voltages and currents at the beginning and at the end of line, as well as the balance of active or reactive power along the line are analyzed.*

Keywords: *Transmission network, MATLAB Simulink, Overhead lines, Balance of electric power, Phase shift*

1. UVOD

Prenosna mreža elektroenergetskog sistema (EES-a) ima za primarni cilj energetski efikasan prenos električne energije od primarnih izvora, koji se konvertuju u električnu energiju u konvencionalnim elektranama, do velikih distributivnih potrošačkih područja.

Efikasan prenos je jedino moguće ostvariti ukoliko se on vrši na visokom, a često i vrlo visokom naponskom nivou. Sa druge strane, ekonomski je najisplativije konvertovati primarne izvore energije u električnu energiju na mestima gde se primarni izvori geografski i nalaze, što je neretko veoma udaljeno od područja koja zahtevaju isporučivanje velikih količina električne energije.

Samim tim sledi da se prenosna električna mreža često sastoji od elemenata koji moraju da prenose električnu energiju na velike razdaljine, što se postiže izgradnjom nadzemnih vodova odgovarajućih dužina. Stacionarni režimi EES-a podrazumevaju rad u ustaljenim, simetričnim i prostoperiodičnim radnim režimima.

NAPOMENA:
Ovaj rad proistekao je iz master rada, čiji mentor je bio dr Dejan Jerkan, docent.

Ustaljeni režimi podrazumevaju rad na mrežnoj učestanosti od 50 Hz, odosno 60 Hz u nekim zemljama. Sa tehničkog stanovišta, električne veličine na pomenutim mrežnim učestanostima spadaju u sporopromenljive veličine, za koje se često može smatrati da u određenom delu sistema gotovo trenutno postižu iste vrednosti u istim trenucima. Ovo, međutim, ne mora biti slučaj u prenosnim mrežama, u kojima se dužine nadzemnih vodova često mere i u stotinama kilometara. Naime, usled velike dužine vodova kojima se električna energija prenosi, dolazi do situacija u kojima nije zanemariljiva pojava kašnjenja, koja nalikuje efektima rasprostiranja elektromagnetnih talasa.

Na taj način se imaju radni režimi u kojima dolazi do faznog pomeraja između veličina na početku, odnosno kraju dugačkih nadzemnih vodova, a kao posledica kašnjenja u njihovom rasprostiranju. Ovaj efekat kašnjenja može imati negativne posledice po funkcionisanje EES-a, ponajviše sa aspekta pravilnog delovanja određenih tipova zaštite. U prenosnoj mreži se za zaštitu nadzemnih vodova najčešće koristi podužna diferencijalna zaštita, koja radi na principu poređenja vrednosti struja i napona sa različitih strana voda. Ukoliko je vod u stacionarnom stanju, zbog svoje dužine može doći do situacije da se fazni stavovi struja i napona osetnije razlikuju na njegovom početku, odnosno kraju, što može dovesti do pogrešnog i nepotrebnog delovanja zaštite.

Zbog toga je neophodno na pravilan način i unapred tačno poznavati efekat kašnjenja koji vod svojom dužinom unosi u stacionarne radne režime EES-a, kako bi se fazni pomeraj koji tom prilikom nastaje mogao unapred uvažiti.

2. MODELOVANJE DUGIH NADZEMNIH VODOVA

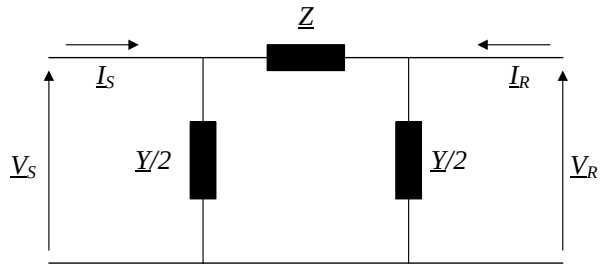
Nadzemni vodovi većih dužina se moraju modelovati tako da uvažavaju efekat kašnjenja koji se manifestuje u pojavi faznog pomeraja između odgovarajućih parova prostoperiodičnih veličina sa početka, odnosno kraja voda.

U tu svrhu se koriste takozvane jednačine telegrafičara, kojima se definišu veze između parova napona, odnosno struja sa kraja voda i proizvoljne tačke na mestu x na samom vodu:

$$\begin{aligned} \underline{V}(x) &= \underline{V}_R \cdot ch\left(\underline{\gamma} x\right) + \underline{Z}_C \cdot \underline{I}_R \cdot sh\left(\underline{\gamma} x\right) \\ \underline{I}(x) &= \frac{\underline{V}_R}{\underline{Z}_C} \cdot sh\left(\underline{\gamma} x\right) + \underline{I}_R \cdot ch\left(\underline{\gamma} x\right), \end{aligned} \tag{1}$$

gde je:

x $[km]$ tekuća prostorna koordinata merena od prijemnog kraja voda (indeks „R“); $\underline{V}_R[kV]$ i $\underline{I}_R[A]$ predstavljaju fazni napon i struju na prijemnom kraju voda; $\underline{Z}_C = \sqrt{\frac{\underline{z}}{\underline{y}}}[\Omega]$ predstavlja karakterističnu impedansu voda; $\underline{\gamma} = \sqrt{\underline{z}\underline{y}} = \alpha + j\beta$ predstavlja konstantu prostiranja voda, α predstavlja koeficijent slabljenja, a β predstavlja fazni koeficijent voda; $\underline{z} = r + jx\ [\Omega/km]$ predstavlja podužnu impedansu voda, r je podužna otpornost, a x podužna reaktansa voda; $\underline{y} = g + jb\ [S/km]$ predstavlja podužnu admitansu voda, g je podužna provodnost, dok je b podužna susceptansa voda. Na slici 2.1 je prikazana pofazna, simetrična π – ekvivalentna šema voda, u kojoj su naznačene redna impedansa i otočne admitanse, koje su dobijene od gore navedenih podužnih parametara množenjem sa dužinom voda.



Slika 2.1. *Pofazna π - ekvivalentna šema voda*

Da bi se efekat kašnjenja na vodovima mogao pravilno kvantifikovati, neophodno je definisati još neke veličine koje karakterišu električne veličine na mrežnoj učestanosti, koje se sada moraju tretirati kao elektromagnetni talasi.

Talasna dužina, koja predstavlja rastojanje na vodu nakon kojeg se menja fazni stav napona ili struje za celih 2π [rad] se može izračunati kao:

$$\lambda = \frac{2\pi}{\beta} = \frac{2\pi}{\sqrt{b \cdot x}} = \frac{1}{f_n\sqrt{l_1 c_1}}, \qquad (2)$$

gde je:

$\lambda[km]$ predstavlja talasnu dužinu; $f_n[Hz]$ predstavlja mrežnu frekvenciju; $l_1[H/km]$ i $c_1[F/km]$ predstavljaju podužnu induktivnost i podužnu kapacitivnost voda, respektivno. Brzina prostiranja elektromagnetnih talasa napona i struje duž voda (v), računajući da je reč o rasprostiranju u vakuumu iznosi:

$$v = \lambda \cdot f_n = \frac{1}{\sqrt{l_1 c_1}}, \qquad (3)$$

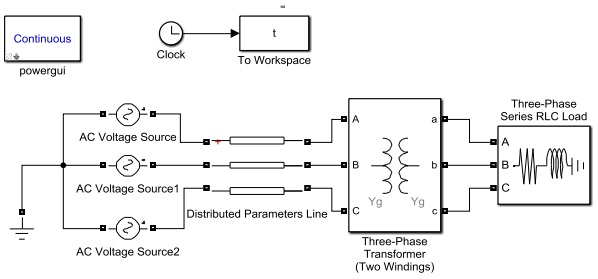
Kako je brzina prostiranja elektromagnetnih talasa napona i struja duž voda za vazdušne vodove bliska brzini svetlosti u vakuumu, $v_0 = 3 \cdot 10^8 \frac{m}{s}$, za tu vrstu vodova važi da je, pri nazivnoj učestanosti od $f_n = 50\ Hz$, talasna dužina prostiranja napona i struja duž voda (λ_v) približno 6000 km.

Napon i struja na početku u funkciji napona i struje na kraju voda mogu se dobiti iz jednačine (1), kada se u nju uvrsti stvarna dužina voda $L[km]$ umesto tekuće koordinate x:

$$\begin{aligned} \underline{V}_S &= \underline{V}_R \cdot ch\left(\underline{\gamma}L\right) + \underline{Z}_C \cdot \underline{I}_R \cdot sh\left(\underline{\gamma}L\right) \\ \underline{I}_S &= \frac{\underline{V}_R}{\underline{Z}_C} \cdot sh\left(\underline{\gamma}L\right) + \underline{I}_R \cdot ch\left(\underline{\gamma}L\right). \end{aligned} \qquad (4)$$

3. MATLAB SIMULINK MODEL UPROŠĆENOG EES-a

U svrhu istraživanja uticaja dužine vazdušnog voda na uspostavljanje ustaljenih radnih režima u prenosnim mrežama EES-a načinjen je uprošćeni model u MATLAB *Simulink*-u, koji je prikazan na slici 3.1.



Slika 3.1. *MATLAB Simulink model uprošćenog EES-a*

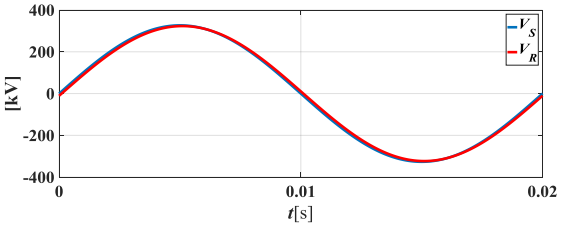
U tabeli 1 dati su osnovni parametri elemenata koji sačinjavaju modelovani EES. Vod je modelovan kao element sa raspoređenim parametrima, koji dopušta uvažavanje efekta kašnjenja, onako kako je to priloženo relacijama u prethodnom poglavlju.

Tabela 1.*Parametri elemenata uprošćenog modela EES-a*

Element	vod	transformator	potrošač
Naponski nivo	400*kV*	400/110*kV/kV*	110 *kV*
Parametri	r=0,0127 $\frac{\Omega}{km}$ l=0,934·10^{-3} $\frac{H}{km}$ c=12,74·10^{-9} $\frac{F}{km}$	u_x=8% u_r=0,2%	/
Snaga	/	*Sn*=250 *MVA*	P=200 *MW* Q=50 *Mvar*

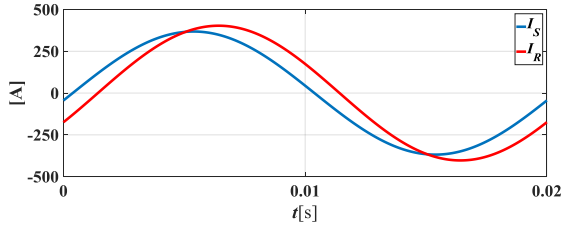
3.1. Karakteristični odzivi voda dužine 100 km

Na razvijenom modelu izvršene su simulacije, u kojima je načinjena varijacija dužine modelovanog voda sa raspo-ređenim parametrima. Izvedene su varijante sa dužinama voda od 100 km, 400 km i 800 km, gde se , kroz priložene karakteristične odzive jasno mogu uočiti osobenosti koje promenljiva dužina voda manifestuje u njima.



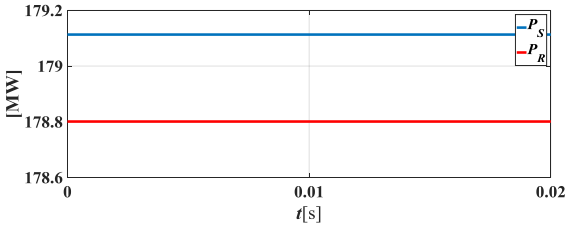
Slika 3.2. *Odzivi napona faze a na početku i kraju voda dužine 100 km*

Na slici 3.2 dat je uporedni prikaz talasnih oblika napona faze a na početku i kraju voda dužine 100 km. Ovaj vod spada u relativno kratke nadzemne vodove naponskog nivoa 400 kV, tako da je slabo izražen efekat kašnjenja na njemu, što se jasno uočava upoređivanjem prikazanih odziva napona.



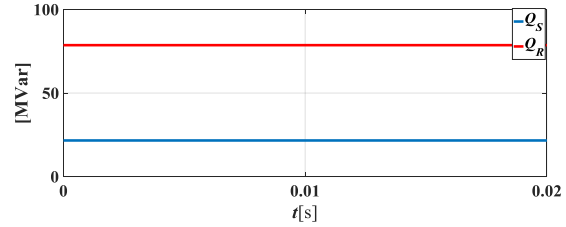
Slika 3.3. *Odzivi struja faze a na početku i kraju voda dužine 100 km*

Na slici 3.3 dat je uporedni prikaz struja faze a na početku i kraju modelovanog voda dužine 100 km. Uočava se veći fazni pomeraj između struja nego što je bio slučaj sa naponima, što se duguje činjenici da sam vod generiše reaktivnu snagu i time nadoknađuje potrebe potrošača.



Slika 3.4. *Odzivi aktivnih snaga na početku i kraju voda dužine 100 km*

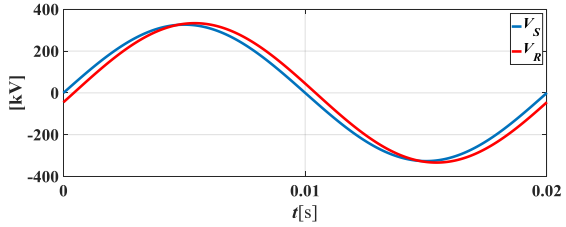
Na slici 3.4 priloženi su odzivi aktivnih snaga na početku i kraju modelovanog voda dužine 100 km. Primećuje se da je ulazna snaga u vod nešto veća od izlazne, zbog postojanja aktivnih gubitaka na samom vodu.



Slika 3.5. *Odzivi reaktivnih snaga na početku i kraju voda dužine 100 km*

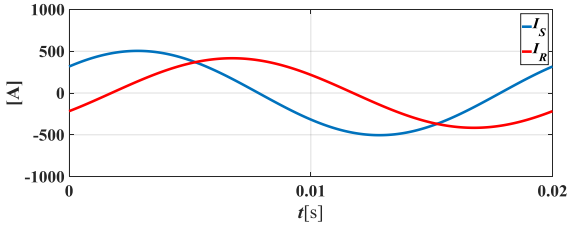
U odzivima reaktivnih snaga sa slike 3.5 na početku i kraju voda dužine 100 km jasno se uočava uticaj generisanja reaktivne snage voda, budući da je izlazna reaktivna snaga veća od ulazne.

3.2. Karakteristični odzivi voda dužine 400 km



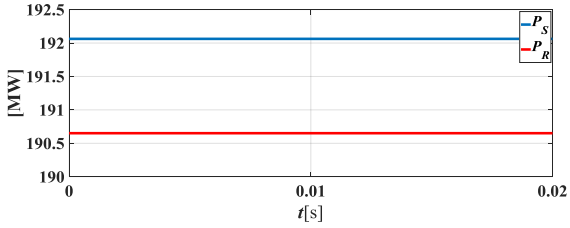
Slika 3.6. *Odzivi napona faze a na početku i kraju voda dužine 400 km*

Na vodu dužine 400 km se primećuje nešto veći fazni pomeraj nego što je to bio slučaj kod voda dužine 100 km, što je i prema očekivanjima, a to se jasno uočava na slici 3.6.



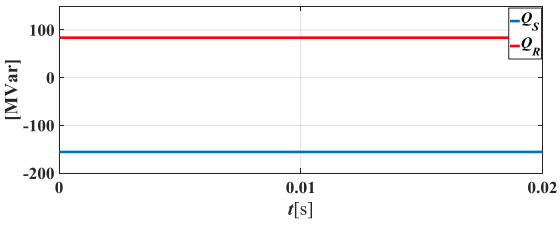
Slika 3.7. *Odzivi struja faze a na početku i kraju voda dužine 400 km*

Fazni pomeraj struja je svakako ponovo izraženiji i kod ovog voda (slika 3.7) u odnosu na fazni pomeraj napona, što se ponovo objašnjava izmenjenim tokom reaktivnih snaga na početku, odnosno na kraju voda.



Slika 3.8. *Odzivi aktivnih snaga na početku i kraju voda dužine 400 km*

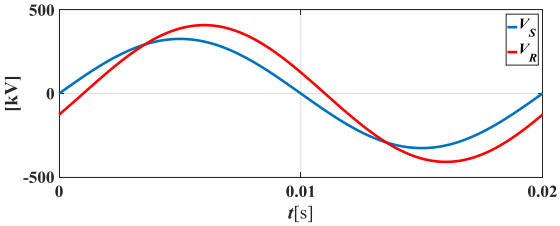
Porastom dužine voda dolazi i do porasta potražnje za akivnom snagom od izvora, usled povećanih gubitaka na samom vodu, što je prikazano na slici 3.8.



Slika 3.9. *Odzivi reaktivnih snaga na početku i kraju voda dužine 400 km*

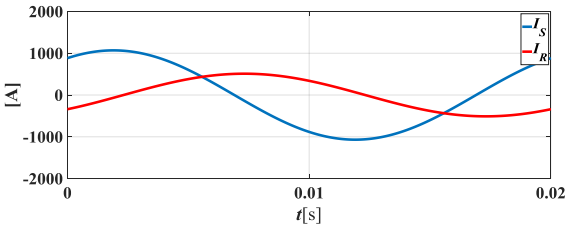
Porast dužine voda je potpuno izmenio tokove reaktivnih snaga, u tolikoj meri da je sada koren mreže postao njen potrošač, što se jasno vidi iz negativne vrednosti reaktivne snage na ulazu u vod, prikazane na slici 3.9.

3.3. Karakteristični odzivi voda dužine 800 km



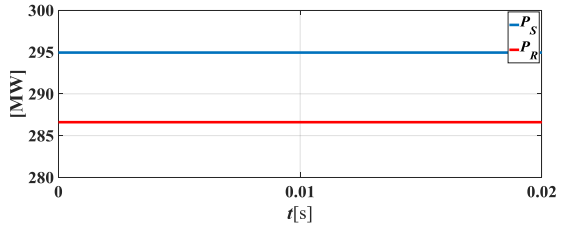
Slika 3.10. *Odzivi napona faze a na početku i kraju voda dužine 800 km*

Na vodu dužine 800 km je primetno izraženiji fazni pomeraj među naponima nego što je to bio slučaj kod prethodnih odziva (slika 3.10), što je i prema očekivanjima, a ujedno se uočava i skok amplitude napona na kraju voda u odnosu na početak. Vod ove dužine spada u veoma dugačke vodove za prenosnu mrežu 400 kV i u ovom radu je analiziran sa čisto teorijskog aspekta, zbog ekstremno izraženih efekata koji u eksploataciji nisu dopušteni.



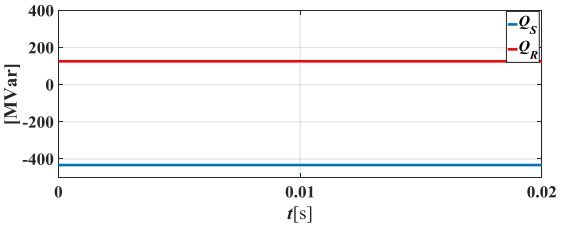
Slika 3.11. *Odzivi struja faze a na početku i kraju voda dužine 800 km*

Na slici 3.11 dat je uporedni prikaz odziva struja faze a na početku i kraju voda. Primećuju se drastične razlike u amplitudama, što se duguje izraženoj reaktivnoj komponenti struje na početku voda, koja isporučuje reaktivnu snagu korenu mreže.



Slika 3.12. *Odzivi aktivnih snaga na početku i kraju voda dužine 800 km*

Povećanjem dužine voda dolazi i do značajnog porasta gubitaka na njemu, što se posledično ogleda i u porastu aktivnih snaga na oba kraja voda, što je prikazano na slici 3.12.



Slika 3.13. *Odzivi reaktivnih snaga na početku i kraju voda dužine 800 km*

Negativni efekti voda veoma velike dužine se, pored odziva napona, najbolje uočavaju u odzivima reaktivnih snaga, prikazanih na slici 3.13. Vod generiše veoma veliku količinu reaktivne snage, koju zatim isporučuje, kako potrošačima, tako i samom korenu mreže, što se ogleda u nedopustivo velikoj negativnoj vrednosti reaktivne snage na početku voda.

4. ZAKLJUČAK

U radu su izložene teorijske osnove modelovanja nadzemnih vodova u visokonaponskim prenosnim mrežama. Zbog potencijalno velikih dimenzija prenosne mreže, u praksi je moguća pojava upotrebe nadzemnih vodova velikih dužina.

Kao posledica toga u eksploataciji EES-a može doći do uspostavljanja radnih režima u kojima dolazi do izražaja efekat kašnjenja, a koji se može negativno odraziti na implementaciju sistema zaštite, a samim tim i na pouzdanost EES-a. Zbog postojanja efekta kašnjenja, potrebno je električne veličine EES-a tretirati kao elektromagnetne talase, pa se na osnovu te činjenice, a uz primenu jednačna telegrafičara, naponi i struje duž dalekovoda tretiraju kao veličine zavisne od prostorne koordinate.

Na osnovu izloženih teorijskih principa je načinjen simulacioni model u MATLAB *Simulink*-u, na kojem su izvršene karakteristične simulacije sa varijacijom dužine modelovanog voda. Ispostavilo se, prema očekivanjima, da porastom dužine voda dolazi do izraženijeg faznog pomeraja među veličinama sa početka, odnosno kraja, ali ujedno i do uspostavljanja nepovoljnog stacionarnog režima sa dramatično izmenjenim tokovima reaktivnih snaga, koje su u praksi nedopustive.

5. LITERATURA

[1] Milan S. Ćalović, *„Osnovi analize elektroenergetskih sistema i mreža“* , Akademska misao, 2004.

[2] *„Electrical Transmission and Distribution Reference Book“* , Westinghouse, Pittsburg, PA, USA, 1950.

[3] M. Ožegović i K. Ožegović, *„Električne mreže: Svezak I“* , Sveučilište u Splitu, Fakultet elektrotehnike, strojarstva i brodogradnje, Split, 1982.

[4] www.mathworks.com

Kratka biografija:

Sanja Špica rođena je u Prijepolju 1996. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Distribuirani energetski resursi odbranila je 2020.godine.

Dejan Jerkan je docent na Fakultetu tehničkih nauka u Novom Sadu, na Katedri za Energetsku elektroniku i pretvarače. Oblast interesovanja su mu modelovanje i dijagnostika električnih mašina, kao i metoda konačnih elemenata.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.317
DOI: https://doi.org/10.24867/10BE32Vucic

PROJEKTOVANJE MERNIH KONDICIONERA

DESIGN OF MESURING CONDITIONERS

Damjan Vučić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Merni kondicioneri su veoma važan deo mernih sistema jer kvalitet i tačnost merenja veoma zavise od kvaliteta realizacije tih ulaznih mernih elektronskih sklopova. Tema ovog rada je teoretska analiza tehnika relizacije mernih kondicionera, a zatim projektovanje operacionog pojačavača kao osnovnog gradivnog bloka mernih kondicionera. Kolo je projektvano i simulirano korišćenjem programskog paketa Cadance. Provereno je da projektovano kolo zadovoljava vrednosti propisane specifikacijom za ekstremne vrednosti parametara okoline kao i za velike varijacije prilikom procesiranja kola na silikonu.*

Ključne reči: *Merni sistemi, kondicioneri, operacioni pojačavači*

Abstract – *Measuring conditioners are a very important part of measuring systems because the quality and accuracy of measurement very much depends on the quality of realization of these input measuring electronic circuits. The topic of this paper is the theoretical analysis of the techniques of realization of measuring conditioners, and then the design of the operational amplifier as the basic building block of measuring conditioners. The circuit was designed and simulated using the Cadance software package. It was checked that the designed circuit meets the values prescribed by the specification for extreme values of environmental parameters as well as for large variations when processing circuits on silicone.*

Keywords: *Mesuring systems, conditioners, operational amplifiers*

1. UVOD

Prelazak sa analognih mernih instrumenata na digitalne merne instrumente je dominantan trend. Ulazna kola kod digitalne merne instrumentacije se uglavnom sastoje od ulaznog multipleksera, *sample & hold* kola, operacionog pojačavača, AD konvertora i mikroprocesora. Bitno je primetiti da se korišćenjem digitalne instrumentacije zanemaruje fizička priroda ulazne veličine jer se nakon digitalizacije signal obrađuje matematičkim operacijama. Taj digitalni signal je veoma otporan na smetnje jer po svojoj prirodi zauzima dve diskretne vrednosti sa kojima je lako manipulisati u matematičkim operacijama.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Josif Tomić, vanr. prof.

Kako je sam hardver standardizovan, rešavanje problema obrade signala se svodi na softveska rešenja programirana na mikroprocesoru.
Tačnost transformacije analognog signala u digitalni predstavlja jednu od najvažnijih stavki. U tom procesu vrši se diskretizacija po vremenu i amplitudi.
Na sreću, u poslednje vreme AD konvertori poseduju brzine od 1MHz i tačnost od 16 bita, po veoma pristupačnoj ceni (oko 30 evra) tako da taj problem više nije aktuelan i ne uzima se u ozbiljnim razmatranjima [1]. Postoje slučajevi kada je potrbna veća brzina od navedene, ali ne tako često.
Po definiciji, metrologija je nauka o merenjima, a instrumentacija je oprema kojom se ta merenja obavljaju. Merenja su osnova prirodnih nauka i ključni element istraživanja i razvoja u svakoj vrsti praktične delatnosti.

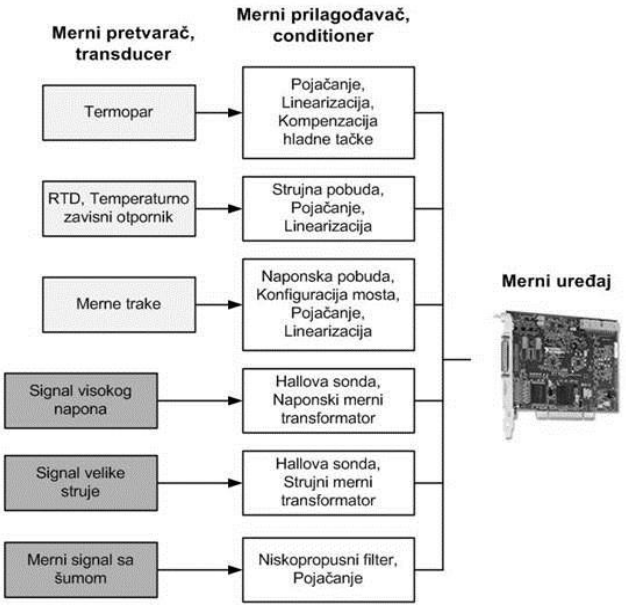
Osnovne karakteristike savremenih merenja su:

- realizacija merenja gotovo svih fizičkih veličina električnim putem,
- nove metoda merenja, proistekle iz simbioze informacionih tehnologija i tehnika merenja,
- tehnološki prelazak na sredstva merenja kod kojih je softver dominantan,
- korišćenje složenih matematičkih algoritama.

2. MERENJE I AKVIZICIJA

Merni pretvarači, *transducer-i*, su uređaji koji konvertuju fizičke fenomene, kao što su: svetlo, temperatura, pritisak ili zvuk, u električni signal koji se može izmeriti. U inženjerskoj terminologiji oni se često nazivaju i senzorima. Najčešći oblici izlaznog električnog signala su napon ili struja.
Za različite fizičke fenomene postoje njima prilagođeni tipovi mernih pretvarača. Tako na primer, pretvarači za temperaturu su termopar, termistor, poluprovodnički senzor; za svetlo foto i vakuumske ćelije, a za zvuk mikrofon [2].

Izbor *transducer*-a je najčešće prvi korak u kreiranju DAQ sistema, (*Data Acquisition System*). *Transducer* najčešće ima efekta u izboru ostalih komponenti u sistemu. Bitno je razumeti rad različitih vrsta pretvarača i koje su njihove prednosti i ograničenja. Nakon konverzije signala *tranducer*-om potrebno je prilagoditi (kondicionirati) signal. Na **slici 2.1** prikazan je izgled kompletnog merno-akvizicionog sistema sa neophodnim mernim pretvaračima i prilagođavačima signala.



Slika 2.1. - Merni system

3. SIGNALI, SENZORI I KONDICIONIRANJE SIGNALA

Prilikom merenja stvarne fizičke pojave, kondicioniranje signala je preduslov za ispravno procesiranje električnih signala sa senzora i pobošanje opšteg kvaliteta podataka [3]. Osnovni tipovi kondicioniranja signala su:

- pobuda mernog senzora,
- pojačanje mernog signala,
- linearizacija mernog signala,
- izolacija mernog signala,
- filtriranje mernog signala.

Pošto merni signali u realnom svetu često imaju male amplitude, kondicioniranje signala može da poboljša tačnost mernih rezultata. Pojačavači povećavaju nivo ulaznog signala tako da bolje odgovara mernom opsegu analogno-digitalnog konvertora (ADC), povećavajući rezoluciju i osetljivost merenja. Dok mnogi DAQ uređaji uključuju *on-board* pojačavače iz ovog razloga, mnogi *transduceri*, kao što su termoparovi, zahtevaju i dodatne pojačavače. Takođe, korišćenje eksternih kondicionera signala, lociranih bliže izvoru signala, poboljšava odnos signal-šum merenja, pojačavanjem nivoa signala pre nego što na njega utiču šum i smetnje iz okruženja [4].

Slabljenje je suprotno od pojačanja. Neophodno je kada je napon koji treba da se digitalizuje iznad ulaznog mernog opsega AD konvertora. Ovaj oblik kondicioniranja signala smanjuje amplitudu ulaznog signala a najjednostavniji kondicioner može biti običan razdelnik napona. Slabljenje je neophodno za merenje visokih napona i struja u elektroenergetici.

Kondicioneri signala mogu da sadrže filtere za smanjenje nepoželjnog šuma u okviru određenog opsega frekvencija. Skoro sve DAQ aplikacije imaju prisutan šum od 50 ili 60 Hz, pokupljen sa električne mreže ili sa električnih mašina. Međutim, većina kondicionera uključuje niskopropusne filtere dizajnirane specijalno da omoguće maksimalno otklanjanje šuma od 50 do 60 Hz. Sledeća česta upotreba filtera je da spreči izobličenje signala – to je fenomen koji se javlja kada je signal *podsempliran* (uzorkovan previše sporo). Nikvistova teorema tvrdi da kada se semplira analogni signal, bilo koja komponenta signala, na frekvenciji većoj od polovine frekvencije sempliranja, pojavljuje se u podacima kao niskofrekventni signal. Ovu distorziju signala možete da izbegnete jedino uklanjanjem svake komponente signala koja je iznad polovine frekvencije sempliranja, pomoću niskopropusnog filtera, pre nego što se signal diskretizuje.

Nepropisno uzemljenje sistema je jedan od najčešćih uzroka problema merenja, uključujući šum i oštećenje uređaja za merenje. Kondicioneri signala sa izolacijom mogu da reše većinu ovih problema. Ovi uređaji prenose signal od izvora do mernog uređaja bez fizičke konekcije, korišćenjem transformatora, optičkih ili kapacitivnih tehnika povezivanja. Pored toga što prekida petlju kroz zemlju, izolacija blokira šumove i odbacuje visoki zajednički napon, te na taj način štiti operatera i skupocenu mernu opremu a merenje čini znatno preciznijim. Digitalni signali takođe zahtevaju periferije za kondicioniranje signala.

Digitalizator, AD konvertor, je obično najskuplji deo sistema za akviziciju podataka. Pomoću multipleksiranja možete da uzastopno dostavljate brojne signale u jedan AD konvertor, a to je jeftin način da se proširi broj mernih mesta u vašem sistemu. Multipleksiranje je neophodno u svim aplikacijama sa velikim brojem senzora.

Kada je potrebno da se mere dva ili više signala u istom trenutku, koristi se simultano sempliranje signala. U tom slučaju potrebno je da svaki kanal ima svoj AD konvertor. Tipični primeri koji zahtevaju simultano sempliranje su: merenje aktivne i reaktivne snage, merenje faznog pomeraja između signala, kao i merenje vibracija.

4. PROGRAMSKI PAKET CADENCE

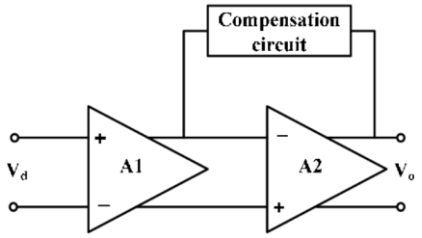
Cadence Design System je americka kompanija koja se bavi razvojem *electronic design automation* (EDA) softvera, ali i inženjerskim poslovima kao što su projektovanje, verifikacija i priprema dizajna za fabrikaciju. Razvija softver za dizajniranje čipova i štampanih ploča (PCB), memorija, analognih kola, SoC periferija, procesorskih jedinica, kao i IP-a. *Virtuoso process design kits* (PDKs) sadrže sve relevantne parametre fabrikacije, od pravila fabrikacije do karakteristika modela, kako bi se sa sigurnošcu moglo reći da to što se isprojektuje može i da se uspešno fabrikuje [5]. Osnova PDKa je precizno određen model komponente po pitanju silicijuma.
Analogno kolo projektovano je u 180nm TSMC tehnologiji korišćenjem posebnog PDK-a firme ams AG. Prilikom dizajnirana korišćeno je više različitih alata u sklopu Cadence programskog paketa, Schematic Editor XL za kreiranje šema i testova, i adexl za simulaciju.

5. PROJEKTOVANI OPERACIONI POJAČAVAČ

Operacioni pojačavač predstavlja jedan od osnovnih gradivnih blokova analognih kola. Pošto se operacioni pojačavači obično koriste kao samostalni blokovi, često je potrebno da se na njima izvrši veliki broj testova da bi se potvrdila stabilnost u širokom rasponu radnih uslova.

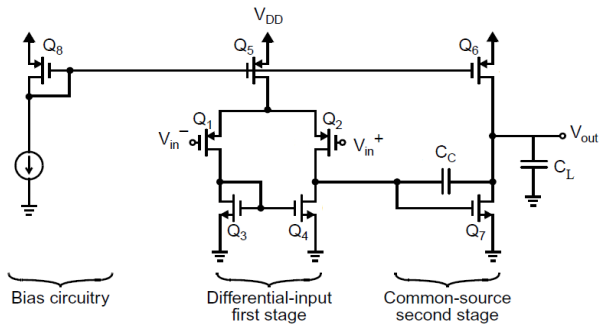
Jedan od osnovnih uslova koje operacioni pojačavač treba da ispuni je veliko pojačanje u otvorenoj povratnoj sprezi. Na pojačavač sa velikim pojačanjem u otvorenoj povratnoj sprezi može da se primeni negativna povratna petlja koja će da stabilizuje pojačanje, ali ga neminovno i smanji. Većina jednostepenih CMOS operacionih pojačavača ne može da obezbedi tako veliko pojačanje, pa su topologije koje se koriste često dvostepene ili višestepene.

U ovom radu opisan je dvostepeni pojačavač. Blok šema takvog kola prikazana je na **slici 5.1**. Jedna od mogućnosti je dodavanje trećeg stepena koji se sastoji od bafera – pojačavača sa jediničnim pojačanjem. Bafer se primenjuje kad je opterećenje koje se postavlja na izlaz operacionog pojačavača rezistivnog tipa. Bafer bi smanjio izlaznu otpornost i omogućio veće varijacije na izlazu [6]. Kad je opterećenje čisto kapacitivno, kao što je i ovom slučaju, bafer se izostavlja. Kao što vidimo na **slici 5.1** kolo sadrži i kompenzaciono kolo (engl. *Compensation circuit*) koje je neophodno kako bi se postigla stabilnost operacionog pojačavača u zatvorenoj povratnoj sprezi.



Slika 5.1. - Blok šema dvostepenog OP

Na **slici 5.2** prikazana je odabrana topologija. Prvi stepen se sastoji od diferencijalnog pojačavačkog stepena sa nebalansiranim izlazom, a drugi od pojačavača sa zajedničkim sorsom. Stujno ogledalo obezbeđuje adekvatnu raspodelu struje do pojačavačkih stepena. Kompenzaciono kolo se pre svega sastoji od kapacitivnog elementa, međutim može da sadrži i otpornik. Budući da se nalazi između ulaza i izlaza drugog stepena velikog pojačanja, on se često naziva Milerov kapacitet, jer je njegovo efektivno kapacitivno opterećenje na prvom stepenu veće od fizičke vrednosti.



Slika 5.2- Odabrana topologija dvostepenog OP

Prednost topologije je da je prilagođena za aplikacije gde se koristi manji napon napajanja, iz razloga što izlazni stepen ne zahteva kaskodu [7]. Pomoću dvostepenog operacionog pojačavača možemo postići veliko pojačanje i veliki izlazni opseg napona. Pre početka projektovanja potrebno je utvrditi specifikacije koje kolo treba da zadovolji. Specifikacije kola su:

• naponsko pojačanje, Av(dB) > 60 dB,
•frekvencija jediničnog pojačanja ili proizvod pojačanja i 3dB opsega GBW >20 MHz,
• fazna margina, PM > 75°,
•maksimalna brzina promene signala na izlazu (engl. Slew rate), SR > 50 V/µs,
• disipacija kola <0,5 mW,
•dozvoljeni opseg napona na ulazu (engl. common mode input range), CMIR (0,2 V - 1,0 V)
•faktor potiskivanja šuma u naponu napajanja, PSRR > 60 dB,
• faktor potiskivanja zajedničkog signala, CMRR > 60 dB
• C_L = 1 pF.

U proračunima se koriste i parametri tehnologije. Tehnologija u kojoj je dizajnirano kolo je 180nm TSMC tehnologija. Tu spadaju sledeći parametri:

- *K_{pn}= 200 µA/V²; K_{pp}= 40 µA/V².*
- *$|V_{tpmin}|$=0,6 V, $|V_{tpmax}|$=0,76 V; V_{tnmin}= 0,64V, V_{tnmax}= 0,8 V [8].*
- *Nominalni napon napajanja je 3,3V, dok je opseg napona napajanja 2,5 V-3,3V.*
- *Temperetaura u kojoj je potrebno da kolo funkiconše je-40 do 80 ºC.*

Potrebno je proračunati DC struje i dimenzije komponenti u bloku. Nakon proračuna slede simulacije rada kola koje će upotrebom složenih modela dati tačniji prikaz funkcionisanja kola i omogućiti informacije na osnovu kojih se kolo može poboljšati.

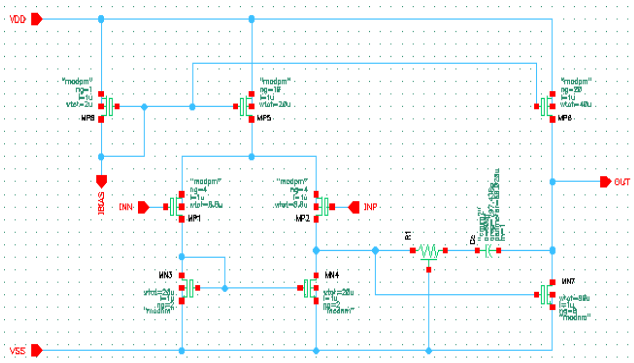
Na osnovu šeme kola dolazimo do sledećih realacija koje opisuju pojačanje prvog, odnosno drugog stepena:

$$A_{V1} = -g_{m1}(r_{ds2}||r_{ds4}) = -2\frac{g_{m1}}{I_5(\lambda_2+\lambda_4)} \quad (1)$$

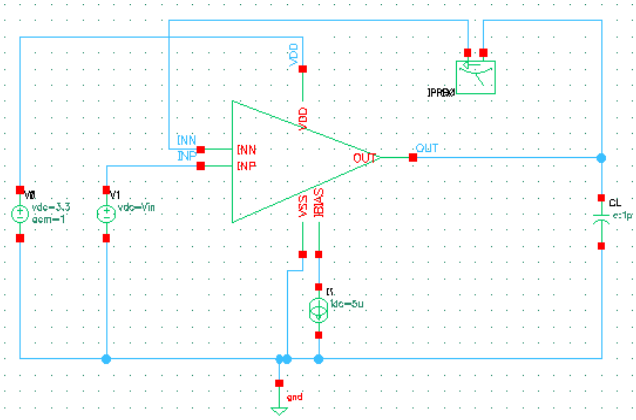
$$A_{V2} = -g_{m7}(r_{ds7}||r_{ds6}) = -\frac{g_{m7}}{I_7(\lambda_7+\lambda_6)} \quad (2)$$

Proračun se započinje iskorišćavanjem vrednosti minimalne fazne margine zadate specifikacijom iz koje se dolazi do frekvencije polova i nula. Zatim se proračunava veličina kompenzacionog kondenzatora, sve struje u kolu i odnosi dimenzija tranzistora. U tim propračunima koriste se jednačine koje opisuju rad tranzistora u odgovarajućem režimu, jednostavni naponski razdelnici i vrednosti definisane iz specifikacije.

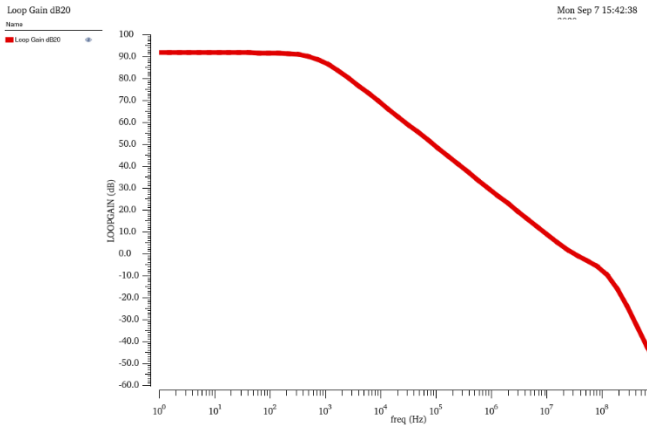
Nakon proračunatih vrednosti kreirana je šema kola (**slika 5.3**) i test šeme **(slika 5.4**) pomoću programskog paketa Cadence. Test šeme su različite i prilagođene za svaki parameter iz specifikacije koji je potrebno proveriti. Simulirane su vrednosti sledećih parametara: CMRR, PSRR, CMIR, SR_rise, SR_fall, fazna margina i pojačanje kola (**slika 5.5**). Na kolu su izvršene i *Monte Carlo* i *corner* simulacije.



Slika 5.3– Šema operacionog pojačavača



Slika 5.4. – Primer test šeme za određivanje pojačanja u zatvorenoj petlji



Slika 5.5 - Pojačanje dvostepenog operacionog pojačavača

Nakon prilagođenja dimezija i vrednosti komponenta u kolu sve vrednosti iz specifikacije su zadovoljene. Promene dimenzija komponenti dovode do promene određenih karakterisitika. U nastavku sledi nekoliko efekata primećenih tokom simulacije kola.

Povećavanjem odnosa dimenzija tranzistora Q5 povećava se struja I_5, što uslovljava povećanje fazne margine i *SR*, međutim smanjuje se pojačanje (ne u svim slučajevima) i povećava se potrošnja. Povećanjem odnosa dimenzija tranzistora diferencijalnog para povećava se i pojačanje prvog pojačavačkog stepena, međutim smanjuje se fazna margina. Povećanjem odnosa dimenzija Q6 i Q7 povećava se pojačanje. Povećanjem Milerovog kondenzatora smanjuje se *SR* i povećava fazna margina.

6. ZAKLJUČAK

U ovom radu detaljno je teoretski obrađena tema mernih kondicionera. Proučeni su svi osnovni tipovi kondicioniranja signala. Operacioni pojačavač, kao jedan od najvažnijih komponenti kondicionog bloka, dodatno je proučen, a zatim i projekovan u 180nm TSMC tehnologiji.

Projektovan je dvostepeni pojačavač sa *nulling rezistor* kompenzacijom. Simulacijama je potvrđeno da kolo radi u granicama specifikacije u svim *corner* slučajevima, procesnim i *mismatch* varijacijama. Potvrđeno je da kolo radi i sa padom napona na 2,5 V i u temperaturnom opsegu od -40 do 80 celzijusovih stepeni. Projektovani pojačavač ima veoma visoku faznu marginu što ga čini stabilnim.

Optimizovane dimenzije kola su (L svih tranzistora je 1µm): W_{12} = 8,8µm, W_{34} = 20µm, W_7 = 80µm, W_8 =2 µm, W_5 = 20µm, W_6 = 40µm, C_C = 600fF, R_1 = 5590Ω.

7. LITERATURA

[1] J. Tomić, M. Milovanović, Virtualna instrumentacija primenom LabVIEW programa, FTN-GRID, Novi Sad, 2010.

[2] J. Tomić, M. Kušljević, Merenje i analiza signala primenom LabVIEW programa, FTN-GRID, Novi Sad, 2016.

[3] Data Acquisition Fundamentals: Improving Measurement Quality with Signal Conditioning, Measurement Computing, Inc., 2013.

[4] Data Acquisition Handbook, Measurement Computing, Inc., 2012.

[5] Ezra Cohen-Yashar, From Tools to Flow: Linking the chains in Cadence Reference Flow, paper from Tower Semiconductor LTD, 2006.

[6] Tony Chan Carusone, David Johns, Ken Martin, Analog Integrated Circuit Design, 2nd edition,Toronto, 2011.

[7] Behzad Razavi, Design of Analog CMOS Integrated Circuits, Los Angeles, 2001.

[8] Interna dokumenta kompanije ams AG, Grac, Austrija.

Kratka biografija:

Damjan Vučić rođen je u Bačkoj Topoli 1991. godine. Nakon završetka osnovnih akademskih studija na Fakultetu tehničkih nauka zapošljava se u firmi *ELSYS Eastern Europe* u Beogradu na poziciji analog lejaut inzenjera. Karijeru nastavlja 2018.-te godine u firmi *ams AG* iz Graca.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE33Colic

UPOREDNA ANALIZA PROCESA REPLIKACIJE KOD RELACIONIH I BAZA PODATAKA NOVE GENERACIJE

COMPARATIVE ANALYSIS OF THE REPLICATION PROCESS IN RELATIONAL DATABASES AND DATABASES OF THE NEW GENERATION

Stefan Colić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Rad opisuje problem procesa replikacije u softverskim sistemima sa analizom pojedinih replikacionih postavki. Fokus je stavljen na ispitivanje osobina proizvoda nove generacije baza podataka koja nastoji da pruži horizontalnu skalabilnost nerelacionih baza podataka sa visokim stepenom konzistentnosti koje pružaju relacione baze.*

Ključne reči: *PostgreSQL, CockroachDB, replikacija, konzistentnost, mikroservisi, CAP teorema.*

Abstract – *This paper describes the problem of replicating data analyzing several replication setups. Focus is placed on examing a product of the next generation of database systems that tends to provide horizontal scalability of non-relational databases and high level of consistency of relational databases.*

Keywords: *PostgreSQL, CockroachDB, replication, consistency, microservices, CAP theorem*

1. UVOD

Visoka dostupnost je važna karakteristika mnogih softverskih sistema. Nedostupnost, spor odziv ili pogrešno funkcionisanje, mogu imati negativan uticaj na pojedince i organizacije i zato je često bitno da svaka komponenta softvera bude otporna na različite vrste otkaza. Replikacija podataka je jedan od načina kako se mogu povećati dostupnost, pouzdanost i skalabilnost sistema. Jedan od razloga za repliciranje informacija jeste redundantnost, odnosno sprečavanje gubitka istih usled otkaza sistema za skladištenje podataka ili drugih hardverskih komponenti. Takođe benefit postojanja više kopija leži u poboljšavanju performansi usled većeg opterećenja. Korisnici mogu istovremeno pristupati različitim kopijama koje su im geografski bliže što smanjuje odziv i pruža bolju pokrivenost. Nedostatak koji dolazi sa procesom replikacije je zaostajanje prilikom propagacije promena između replika što dodatno uvodi problem konzistentnosti. Postoji opasnost čitanja zastarelih informacija što može dovesti do neželjenih aktivnosti. Ovaj rad analizira osobine replikacionih postavki *PostgreSQL* relacione baze i *CockroachDB* baze koja se klasifikuje pod nazivom *NewSQL*.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.

Glavni razlog posmatranja ove dve tehnologije jeste visok stepen konzistentnosti koje pružaju pri replikaciji i pristupu informacijama [1].

Postavke se posmatraju u kontekstu sistema koji pruža različite oblike plaćanja pri čemu se simulira korišćenje od strane određenog broja korisnika. U tom procesu, praćeno je ponašanje baza podataka sa ciljem dostizanja zaključka o upotrebljivosti oba sistema u sličnim okolnostima.

2. TIPOVI REPLIKACIJE

Svaki čvor u replikacionom procesu može imati ulogu vođe (eng. *Leader*), pratioca (*eng. Follower*) ili ulogu koja predstavlja varijaciju ova dve [2]. Lider je čvor koji prihvata upise i radi širenje promena ka ostalim čvorovima pri čemu uglavnom omogućuje i operacije čitanja, dok pratioci pružaju mogućnosti samo čitanja i prihvataju promene poslate od strane lidera [3]. Postoji više tipova replikacije u zavisnosti od prisutnih uloga i odnosa između čvorova pri čemu razlikujemo [2]:

1. *Single-Leader* replikaciju – Postoji samo jedan lider i može postojati veći broj pratilaca.
2. *Multi-Leader* replikaciju – Za razliku od prethodnog tipa, može postojati više lidera, gde svaki lider može predstavljati pratioca drugog lidera.
3. *Leaderless* replikaciju – Fundamentalno se razlikuje od prethodna dva tipa, jer svi čvorovi imaju istu ulogu, odnosno ne postoji eksplicitni čvor koji je deklarisan kao vođa.

Replikaciona postavka sa jednim liderom ima benefit lakšeg razumevanja i pruža podlogu za visok nivo konzistentnosti zahvaljujući *ACID* svojstvima. Nedostatak jeste u ograničenim mogućnostima jednog čvora koji prihvata upise. Te nedostatke otklanja uvođenje većeg broja lidera za opterećenje (eng. *load*) koje se sastoji iz prilibližnog ili većeg broja upisa u odnosu na operacije čitanja. Negativna strana takvog pristupa jeste kreiranje podloge za konflikte koje je neophodno razrešiti na aplikativnom nivou. Oba načina mogu širenje podataka ka čvorovima rešiti asinhrono, sinhrono ili hibridno. Jedan vid hibridnog pristupa jeste *Leaderless* replikacija gde svi čvorovi mogu prihvatati upise, pri čemu se upis ne smatra uspešnim dok se ne postigne kvorum (eng. *quorum*), odnosno dok određen broj (najčešće većina) ne potvrdi da je replikacija uspešno izvršena.

Varijacija pristupa sa jednim liderom jeste replikacija koja koristi konsenzus algoritam. Ona se oslanja na koncept kvoruma pri čemu otklanja mogućnost nastajanja konflikata i rešava se problem izbora novog vođe u slučaju otkaza prethodnog bez negativnog uticaja na konzistentnost podataka [4]. Najpoznatiji algoritmi ove prirode su *Paxos* i *Raft*.

U radu posmatrane postavke su potpuno sinhrona replikacija kod *PostgreSQL* relacione baze koja se oslanja na prenos sadržaja *Write-Ahead Log*-a putem otvorenog toka (eng. *stream*) ka svakom pratiocu i *CockroachDB* replikacije koja je hibridan pristup svih opisanih tipova. Oslanja se na *Raft* konsenzus protokol i nadograđuje ga konceptom kao što je *Multi-Raft*, odnosno postoje više pojedinačnih *Raft* grupa, što ovoj postavci omogućuje da prihvata upise na više čvorova [5]. Takođe uvode apstrakciju zakupa (eng. *lease*) pri čemu se pružaju konzistentna čitanja sa zaobilaženjem *Raft*-a [6].

3. ARHITEKTURA SISTEMA

Sistem koji se opisuje u ovom radu (koncentrator plaćanja) sastoji se iz većeg broja komponenti koje su pokretane na više identičnih računara u lokalnoj mreži. Arhitektura je prikazana na slici 1.

Slika 1. *Arhitektura sistema koji je korišćen za ispitivanje osobina replikacije PostgreSQL-a i CockroachDB-a*

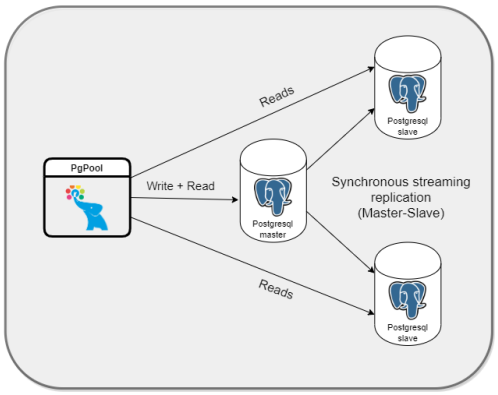
Ulaznu tačku u sistemu čine više instanci *Nginx*-a koji uzimaju ulogu *API Gateway*-a. Njihov zadatak je da *HTTP* zahteve proslede do instanci odgovarajućih mikroservisa na osnovu putanje zahteva. Pre nego što konkretan zahtev prosledi na adekvatno mesto, vrši se autentifikacija i potpomaže se u autorizaciji njegovim prosleđivanjem autentifikacionom servisu koji čita sadržaj *JSON Web Token*-a i pruža odgovor nazad instancama *Nginx*-a koje odlučuju da li zahtev može biti prosleđen dalje. Ovaj mikroservis je realizovan upotrebom *Spring Boot* radnog okvira i zbog svoje učestale upotrebe (pri slanju svakog zahteva), podatke čuva u *H2* bazi koja je smeštena u njegovoj memoriji.

Nakon uspešne autentifikacije, zahtev se prosleđuje nekoj od dostupnih instanci konkretno traženog mikroservisa po kružnom (eng. *round-robin*) principu. Svaki mikroservis se samostalno registruje u registar instanci, *Consul*, koji čuva podatke o svakom i replicira ih i duž klastera koji čine tri instance, upotrebom *Raft* konsenzus protkola. Na osnovu tih podataka, proces pod nazivom *Consul-Template*, osmatra potencijalne promene i vrši ažuriranje konfiguracionog fajla *Nginx*-a pri svakoj novoj registraciji ili uklanjanju servisa iz registra.

Osnovna funkcija ovog sistema je da posreduje između klijenata i različitih servisa za plaćanje. Spram toga, podržano je plaćanje platnom karticom, putem *PayPal*-a i putem *Bitcoin* kriptovalute. Svaki oblik plaćanja je realizovan kao zaseban mikroservis (pokrenut u dve instance) koji prihvata zahteve od korisnika, obrađuje ih, šalje eksternim serivisma i čuva podatke u lokalnoj bazi. U testovima koji su sprovođeni, kreiran je veliki broj zahteva, zato su pozivi eksternim servisima mokovani predefinisanim odgovorima.

Mikroservisi se obraćaju sloju za skladištenje podataka, pri čemu se instance različitih servisa mogu obraćati istoj fizičkoj mašini. S obzirom da se osmatraju osobine baza podataka pod velikim opterećenjem, nije neophodno napraviti separaciju, koja bi u produkciji bila poželjna.

Jedan od razloga upotrebe prethodno opisanih baza podataka jeste u korišćenju istog *PostgreSQL wire protocol*-a, što svodi promenu upotrebljivane baze na promenu konekcionog parametra u konfiguraciji pojedinačnih mikroservisa. Replikaciona postavka u kontekstu *PostgreSQL*-a (prikazan na slici 2) zahteva upotrebu međusloja koji će vršiti rutiranje upita u zavisnosti od toga da li se upisuju ili čitaju podaci sa diska. Upisi su isključivo usmereni ka vođi, dok se čitanja mogu usmeravati na sve tri prikazane replike.

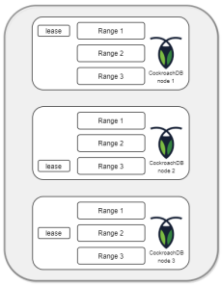


Slika 2. *PostgreSQL streaming replikacija sa upotrebom međusloja u vidu PgPool-II-a*

CockroachDB se klasifikuje u kategoriju *NewSQL* baza koje površinski izgledaju kao relacione baze, zbog svog *SQL* sloja. Na najnižem nivou, koristi se *Key-Value* skladište pri čemu je sadržaj particionisan na opsege (eng. *Range*) koji su raspoređeni po različitim replikama u zavisnosti od replikacionog faktora (primer na slici 3). Svaki opseg predstavlja prethodno opisanu *Raft* grupu. Takođe, svaki čvor u klasteru ujedno igra i ulogu *gateway*-a, odnosno za slučaj da ne sadrži *lease* za određen opseg, zahtev se prosleđuje adekvatnoj replici. Ovakav pristup ne zahteva upotrebu dodatne komponente za balansiranje zahteva s obzirom na postojanje parametra u *JDBC Api*-u koji omogućuje *HikariCP*-u da otvara i održava ravnomeran broj konekcija ka svim čvorovima.

Zahtevi su generisani distribuirano upotrebom *locust* alata, pri čemu su simulirani korisnici koji koriste različite metode plaćanja. U slučaju plaćanja platnom karticom i kriptovalutom, proces obuhvata: proveru podržanosti platne metode, generisanje tokena za plaćanje, plaćanje, nasumična promena statusa transakcije, prikaz detalja i prikaz poslednjih dvadeset izvršenih transakcija.

Mikroservis za podršku plaćanja putem *PayPal*-a dodatno pored navedenih opcija pruža identičan tok za opciju pretplate.



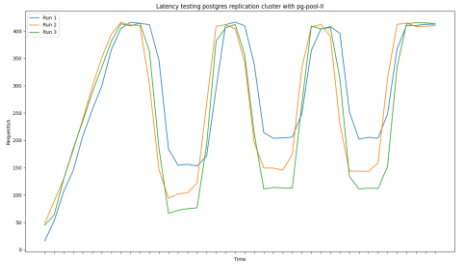
Slika 3. *CockroachDB čvorovi sa 3 opsega*

Sve navedene komponente su pokretane u *Docker* kontejnerima na sedamnaest računara identične konfiguracije u *Swarm Mode*-u. Na svim računarima dodatno su pokretani kontejneri (*CAdvisor, Node-Exporter, Grok-Exporter*) koji omogućuju prikupljanje metrika sa svakog pojedinačnog računara koje su čuvane u *Prometheus Time-Series* bazi. Dodatno je korišćena *Grafana* za elegantan prikaz prikupljenih podataka.

4. REZULTATI I TUMAČENJA

Svaki test pokretan je u istom okruženju sa istim rasporedom na računarima, pri čemu su pojedini testovi pokretani više puta radi preciznijeg prikupljanja informacija. Metrika koja je pritom najviše opažana je broj zahteva po sekundi (*Requests/s)*.

Test povećanog vremena kašnjenja sprovođen je upotrebom *traffic control* alata, pri čemu je simuliran problem sa mrežom od 200, 350 i 500 milisekundi. U *PostgreSQL* postavci sa međuslojem (slika 4), kašnjenja su simulirana na svakom čvoru, pri čemu su nešto lošije performanse dobijene kada je kašnjenje usmereno na lideru, što je očekivano s obzirom da taj čvor i dalje može prihvatati operacije čitanja, a izvršava jedini sve upise.



Slika 4. *Simulacija kašnjenja na PostgreSQL-u sa PgPool-om*

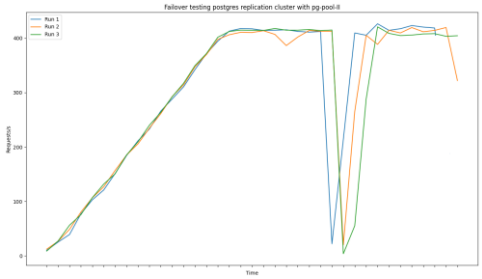
Testovi redom prikazuju 200, 350 i 500 milisekundi na opterećenju koje čini 200 korisnika. Razlog ovolikog pada u broju zahteva po sekundi ogleda se u tome da svaki zahtev izvršava dva do tri upita koji se izvršavaju unutar granica transakcije (*Begin* i *Commit*). Za svaki iskaz se agregira simulirano kašnjenje.

Kod *CockroachDB*-a, kašnjenja dodatno smanjuju broj zahteva po sekundi (slika 5), pri čemu se razlog za to može uzeti iz činjenice da je neophodan veći broj skokova između čvorova. S obzirom da ovakav sistem zbog svoje distribuiranosti u velikoj meri zavisi od optimizovanih upita, broj skokova zavisi od toga šta *cost-based optimizer* odluči da je neophodno uraditi na osnovu prikupljenih statistika tokom izvršavanja pri čemu direktno može uticati na povećano kašnjenje. Takođe, postojanje sekundarnih indeksa je od krucijalne važnosti za ovakav sistem, što dodatno može usporiti upise jer se isti mora održavati. Radi lakšeg prikaza, broj korisnika u ovim testovima je 800.

Slika 5. *Simulacija kašnjenja na CockroachDB-u*

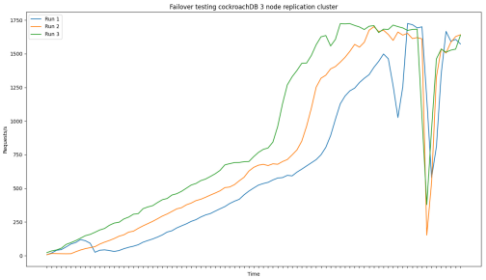
Ono što treba zaključiti je da neće svi zahtevi kasniti. Čitanja se vrše sa čvorova koji sadrže *lease*. Ako je problem van tog čvora, zahtev će vrlo verovatno biti netaknut. Za upise, neophodno je da 51+% potvrdi da je izvršilo replikaciju, što ne mora uzeti u obzir problematičan čvor. U većem klasteru je dodatno manja verovat-noća da se traženi podaci nalaze na problematičnom mestu.

Test otkaza lidera je dao prilično očekivane rezultate. Kod *PostgreSQL*-a, sistem je nedostupan onoliko vremena koliko je rečeno replikama da pokušavaju da kontaktiraju lidera, što je u slučaju testa bilo 12 sekundi (slika 6).



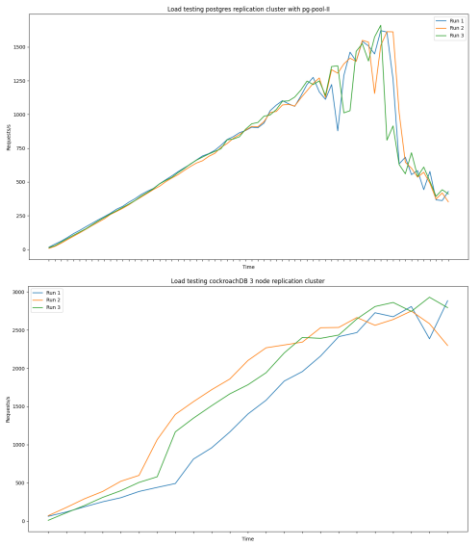
Slika 6. *Simulacija otkaza na PostgreSQL-u*

Sa druge strane, simulacija otkaza je vršena na svakom čvoru u klasteru (slika 7). Koliki je uticaj, zavisi od ukupnog broja *Raft* lidera na pogođenom čvoru, koji može biti varijabilan. Za razliku od *PostgreSQL*-a, ovaj sistem neće biti skroz nedostupan za vreme izbora novog lidera za pojedine opsege. Vreme neophodno za stabilizaciju je oko 9 sek. (vreme za izbor novog lidera i preuzimanja *lease*-a.



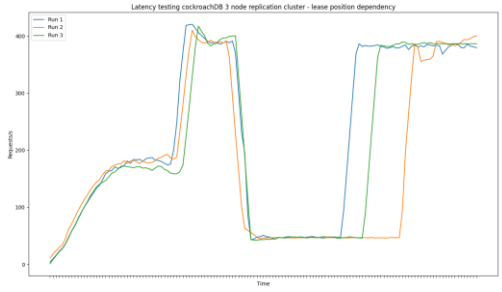
Slika 7. *Simulacija otkaza na CockroachDB-u*

Test velikog broja korisnika se odnosi na ispitivanje ponašanja u onim danima kada je veliko opterećenje sistema, pri čemu se bolje pokazao *CockroachDB* (slika 8) koji zbog distribuirane prirode manje opterećuje procesor i bolje koristi disk (običan HDD).



Slika 8. *Prikaz broja zahteva po sekundi pri većem broju korisnika*

Test praćenja pozicije se odnosi striktno na mogućnost *CockroachDB*-a da zakup (*lease*) prebaci u onu replikacionu zonu odakle dolazi najveći broj zahteva, ako će ta promena dovesti do poboljšanja performansi. S obzirom da čvor sa zakupom zaobilazi *Raft* konsenzus protokol, to će drastično poboljšati performanse operacija čitanja. Test je sprovođen sa jednostavnim servisom koji ima jednostavnu šemu baze radi lakše simulacije i provere navedene funkcionalnosti. Uočeno je da za transfer zakupa neophodno između 30 i 60 sekundi.



Slika 9. *Follow-the-workload funkcionalnost kod CockroachDB-a*

Test periodičnog gašenja i uključivanja *Docker* kontejnera je samo dodatno potvrdio neke opservacije iz prethodnih testova. Otkaz na bilo kom čvoru kod *PostgreSQL*-a potpuno obustavlja sve operacije upisa, dok kod *CockroachDB*-a će pojedini zahtevi ostati netaknuti.

5. ZAKLJUČAK

Tokom testiranja uočene su razne prednosti i jednog i drugog sistema, kao i određeni nedostaci. Sama priroda *Single-Leader* replikacije čini sistem lako razumljivim, ali implementacija kod *PostgreSQL*-a može dovesti do neželjenih okolnosti usled nedovoljnog razumevanja kako ona funkcioniše. Potpuno sinhrona postavka garantuje da kada je odgovor dobijen, sledeći zahtev će uočiti napravljene promene ili novije, nikad starije. Ali se ne rešava problem čitanja zastarelih podataka dok sam proces replikacije traje.

Razlog za to jeste izvršavanje *commit*-a na lideru nakon čega se čeka replikacija ka svim replikama. Pratilac koji prihvati podatke će ih moguće učiniti vidljivim pre nego što vrati odgovor vođi (zavisi od podešavanja), dok kod vođe podaci postaju vidljivi onog trenutka kada sve potvrde pristignu. Za striktno konzistentna čitanja, potrebno je da te operacije budu upućene ka lideru.

CockroachDB u rešava navedene probleme, što uz određena bolja ponašanja kroz testove čine sistem jako primamljivim. Pruža se visok nivo konzistentnosti koji gotovo da u potpunosti odgovara definiciji konzistentnosti iz *CAP* teoreme. Dostupnost u većim klasterima može biti na zavidnom nivou, za razliku od posmatrane relacione baze gde otkaz jednog pratioca negativno utiče na ceo sistem. Klaster je jednostavno namestiti i većina operacija se odvija automatski, bez preke potrebe za manuelnom intervencijom. Ono što je najveći nedostatak ovog sistema jesu perforomanse što zahteva visoku upotrebu indeksa koje mogu iste načiniti još lošijim, kao i visoku optimizaciju *SQL* upita. S obzirom da su podaci distribuirani, *join* operacije su jako neefikasne što čine ovaj sistem skoro neupotrebljivim u situacijama kada su te operacija jako učestale.

6. LITERATURA

[1] „Jepsen Analysis,“ [Na mreži]. Available: https://jepsen.io/analyses. [Poslednji pristup September 2020].

[2] M. Kleppman, „Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable and Maintainable Systems,“ 2017.

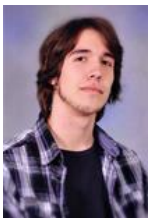
[3] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, „Understanding Replication in Databases and Distributed Systems, **“** *Proceedings - International Conference on Distributed Computing Systems*, 2000.

[4] D. Ongaro, J. Ousterhout, **„**In Search of an Understandable Consensus Algorithm,“ *USENIX Annual Technical Conference*, 2014.

[5] „Scalling Raft,“ [Na mreži]. Available: https://www.cockroachlabs.com/blog/scaling-raft/. [Poslednji pristup September 2020].

[6] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss, P. Bardea, A. Ranade, B. Darnell, B. Gruneir, J. Jaffray, L. Zhang, P. Mattis, „CockroachDB: The Resilient Geo-Distributed SQL Database,“ *SIGMOD, the Association for Computing Machinery*, 2020.

Kratka biografija:



Stefan Colić rođen je u Beogradu 09.11.1995. godine. 2014. godine se upisuje na Fakultet Tehničkih nauka, smer softversko inženjerstvo i informacione tehnologije i diplomira 2018. godine sa prosekom 9.85. Iste godine upisuje master studije na istoimenom smeru.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.738.5:339
DOI: https://doi.org/10.24867/10BE34Vukasinovic

APLIKACIJA ZA REVIZIJU KNJIŽNOG FONDA BIBLIOTEKE

APPLICATION FOR LIBRARY COLLECTION REVISION

Ivan Vukašinović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *Cilj ovog rada je da olakša evidenciju stanja knjiga radnicima u bibliotekama. Umesto dosadašnjeg ručnog popisivanja knjiga, radnici biblioteke mogu upotrebom ove mobilne aplikacije da prođu kroz policu sa njigama i skeniraju kod na poleđini svake od njih. Aplikacija omogućava kreiranje sesije skeniranja koje se mogu kreirati na osnovu autora knjiga, reda na polici ili nekog drugog kriterijuma. Nakon kreiranja sesije otvra se skener koji evidentira svaki validan kod i čuva ga u memoriji telefona. Ukoliko radnik poželi da sinhronizuje stanje sa stanjem koje ima na serveru moguće je pritiskom na dugme za slanje na server.*

Ključne reči: *Skener, Biblioteka, aplikacija*

Abstract – *The aim of this paper is to make it easier for librarians to record the condition of books. Instead of manually listing books, library workers can use this mobile application to go through the shelf with them and scan the code on the back of each of them. The application allows you to create scan sessions that can be created based on the author of the books, the order on the shelf or some other criteria. After creating a session, a scanner opens that records each valid code and stores it in the phone's memory. If the worker wishes to synchronize the state with the state he has on the server, it is possible by pressing the send button for sending to the server.*

Keywords: *Scanner, Library, Application*

1. UVOD

Jedna od softverskih industrija sa najznačajnijim rastom u poslednje vreme je mobilna industrija [1].

U ovom radu biće predstavljeno rešenje mobilne aplikacije koja omogućava evidenciju knjiga u biblioteci skeniranjem koda na svakoj knjizi. Aplikacija se izvršava na iOS i Android operativnim sistemima (OS). Upotrebom jednog koda prikazuje se mogućnost zaobilaženja barijera u pisanju dve različite aplikacije, kako je najčešće do sada bio slučaj.

Umesto dosadašnjeg načina evidencije knjiga prilikom inventara, radnici bilioteka bi mogli upotrebom ove aplikacije uštedeti znatno vreme.

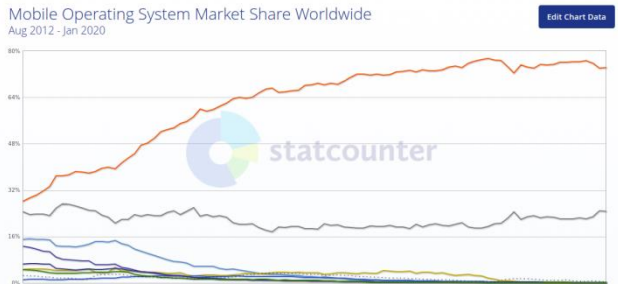
NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.

2. HIBRIDNE MOBILNE APLIKACIJE

Aplikacije koje treba pokrenuti na više platformi obično imaju različite izvorne kodove po platformi. Ovakav razvoj je kompleksan i troši znatnu količinu resursa jer se izvorni kod mora duplirati za različite platforme, što može povećati troškove projekta. Kao odgovor pojavili su se alati koji inženjerima omogućavaju da implementiraju jednu aplikaciju i „izvoze“ je na više platformi. Ova vrsta aplikacija poznata je kao cross-platform aplikacija i ima glavnu prednost u pojednostavljenju životnog ciklusa aplikacije centralizacijom razvoja u jednoj bazi koda [4].

1.1. Razvoj operativnih sistema mobilnih uređaja

Tržište pametnih telefona je u ekspanziji u poslednjih 10 godina. Globalna isporuka pametnih telefona je porasla sa 173.5 miliona jedinica u 2009. godini na 1404.9 miliona jedinica u 2018. Tokom ovog perioda bilo je različitih mobilnih operativnih sistema kao što stu Android, Blackberry, iOS, Symbian i Windows. Danas na tržištu dominiraju dva konkurenta, Android i iOS [2]. Kompanije koje razvijaju mobilne aplikacije imaju problem na koji operativni sistem je najbolje fokusirati se. Čak iako danas vlada tržište Android i iOS operatvnih sistema, može još neko lako preuzeti tržište. Ovo su nesigurne okolnosti koje kompanije moraju uzeti u obzir pri izboru strategije [3]. Na slici 1. prikazano je tržište zastupljenosti operativnih sistema mobilnih telefona.



Slika 1. *Učešće na tržištu mobilnih operativnih sistema širom sveta od avgusta 2012. do januara 2020. [3]*

2. REACT NATIVE

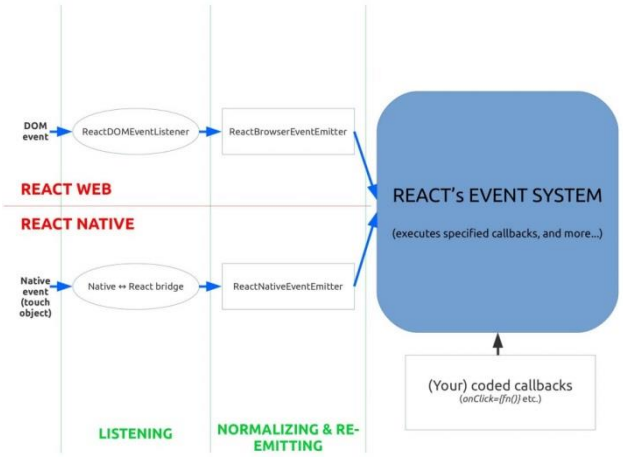
React Native je framework (radni okvir) otvorenog koda koji je Facebook objavio 2015. godine. Godine 2018. imao je drugi po redu najveći broj saradnika u odnosu na bilo koji drugi repozitorijum na GitHub-u. Trenutno podržava Android i iOS.

Programski jezik koji se koristi za razvoj ove tehnologije je JavaScript i prikazuje korisnički interfejs pretvaranjem u matične komponente dajući aplikacijama razvijenim u

React Native izvorni izgled i osećaj. UI je izgrađen sa komponentama platforme poput View, Text i Image koje se direktno preslikavaju u izvorne blokove određene platforme.

2.1. Reakcija na događaje

React i React Native koriste isti mehanizam kada je u pitanju reakcija na događaje (*event handling*). Konceptualno predstavlja veoma intuitivan proces. Svrha je presresti određene događaje (dodir ekrana, klik) i aktivirati određeni *callback koji je kreiran za tu svrhu [5]*.



Slika 2 – Koraci reagovanja na događaje [5]

Mehanizam koji React naglašava je harmonizacija: *cross-browser* za React i *cross-platform* za React Native. Sistem reagovanja na događaje je isti. Oba sistema (DOM i Native) reagovanja na događaje koriste isti kod [5].

Za React Native događaji se primaju preko mosta koji povezuje native kod sa React-om. Ukratko, kad god se stvori prikaz, React takođe prenosi svoj ID broj na native deo, kako bi mogao da primi sve događaje koji se odnose na taj element.

2.2. Pristup senzorima

Jedan od benefita rada u React Native-u je to što dopušta pozivanje matičnih funkcija uređaja uz pomoć JavaScript-a. Problem koji se javlja kada želimo da pristupim određenim senzorima upotrebom ove tehnologije je taj što su obično svi pristupi usko vezani za određenu platformu (iOS ili Android).

Često je potrebno doći na nivo ispod u apstrakciji kako bismo pristupili određenim funkcijama senzora.

Ukoliko je potrebno koristiti žiroskop (Gyroscope) senzor, nailazi se na problem kod podešavanja za obe platforme.

Međutim, već postoje biblioteke koje rešavaju ovaj problem i pružaju API koji je jednostavniji i intuitivniji. Jedna od takvih biblioteka je *react-native-sensors* [6].

Ova biblioteka pruža RxJS bazirani interfejs za akcelerometar, žiroskop i magnetometar.

3. IMPLEMENTACIJA

Aplikacija je izrađena u React Native tehnologiji sa Expo podrškom. Expo je doveo u velikoj meri do ranijeg završetka aplikacije pruživši podršku za korišćenje kamere, vibracije i besplatno objavljivanje aplikacije na svojim serverima tako da je moguće bilo testirati *release* verziju u više navrata.

3.1. Code 128

Code 128 predstavlja jedan od standarda kodiranja teksta, brojeva, brojne funkcije i čitav skup ASCII karaktera. Obično se koristi za nekoliko primena. Naziva se još i ISBT-128, GS1-128, UCC-128, EAN-128 i USS Code 128 [7]. On sadrži 106 različitih štampanih obrazaca barkodova. Svaki odštampani bar kod može imati jedno od tri različita značenja, u zavisnosti od toga koji se od skupova znakova koristi, uz dostupnost tri različita početna znaka Code 128 za programiranje početnog skupa znakova. Funkcije su takođe obezbeđene u simbolici barkoda za prebacivanje između skupova znakova i kodiranje aplikacionih identifikatora. Barkod Code 128 može biti složen za upotrebu zbog različitih skupova znakova, što je osnovni razlog što IDAutomation pruža automatsku funkciju code 128, „Code128()“ u nekoliko alata za font, komponente i aplikacije [7].
Kompletan code 128 je sačinjen od početnog znaka, cifara podataka, modula 103 kontrolne cifre i znaka zaustavljanja što se vidi na slici 3.

Slika 3. *primer barkoda code 128*

3.2. Skener

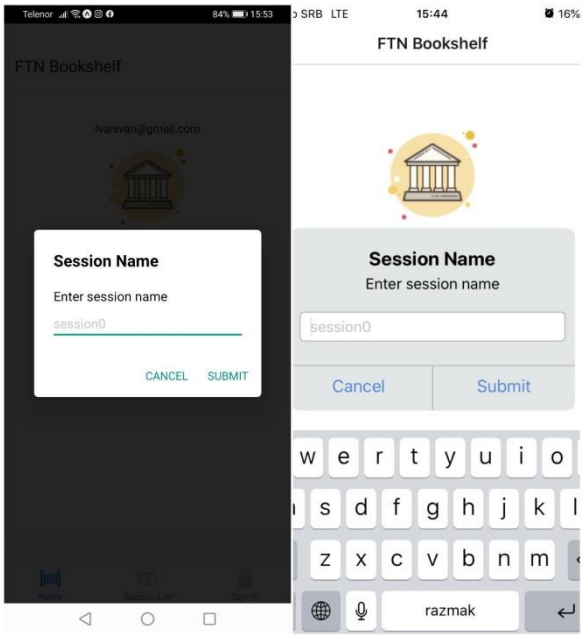
Izborom dugmeta za skeniranje prikazuje se modal u kojem se može izabrati naziv sesije skeniranja ili se može ostaviti prazno nakon čega aplikacija bira naziv na osnovu trenutne liste u memoriji (Slika 4).

U ovom trenutku je moguće i otkazati skeniranje pritiskom na dugme Cancel.

Odabirom dugmeta „Submit“ prelazi se na ekran sa skenerom. Skener je implementiran upotrebom Expo paketa expo-barcode-scanner koji je u znatnoj meri olakšao implementaciju. Ovaj paket podržava i druge vrste kodova kao što su qr, rss14, aztec, code 39,..

U nastavku je prikazan izvorni kod koji predstavlja komponentu za skeniranje barkoda. BarCodeScanner komponenta prima više parametara, među njima su callback funkcija onBarCodeScanned koja se aktivira kada se uspešno skenira kod i tipovi kodova koji su podržani kroz polje barCodeTypes.

Takođe u ovom fajlu se nalazi i SnackBar komponenta čija je uloga da se na kratko prikaže poruka ukoliko je kod uspešno skeniran ili poruka greške ukoliko knjiga već postoji u memoriji (Slika 6).



Slika 4. Modal za unos naziva sesije

Navedene poruke prati vibracija, pa tako ukoliko dođe do uspešnog skeniranja duža vibracija obavesti korisnika, dok ukoliko je došlo do greške, više kratkih vibracija označi da se desila greška.

```
<View
style={{
flex: 1,
flexDirection: 'column',
justifyContent: 'flex-end'
}}
>
<Button title="Finish" onPress={() =>
finishScannig()} />
<BarCodeScanner
onBarCodeScanned={scanned ? undefined :
handleBarCodeScanned}
barCodeTypes={[BarCodeScanner.Constants.BarCodeT
ype.code128]}
style={{ ...StyleSheet.absoluteFillObject,
bottom: 30 }}>
position="top"
/>
</View>
```

Listing 1. Prikaz izvornog koda skenera

4.3. Lista skeniranih sesija i knjiga

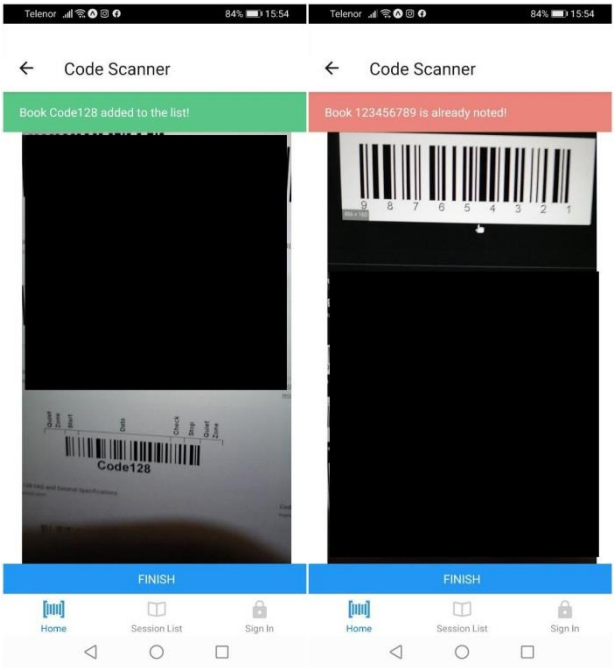
Završetak skeniranja se obavlja pritiskom na dugme „FINISH“. Nakon toga pritiskom na drugi tab na dnu „Session List“ prebacuje se na ekran sa listom sesija. Lista sesija prikazana je na slici 5.

Na ovom ekranu vidimo prikaz svih sesija i dobijamo informaciju koliko knjiga ima u sesiji, naziv sesije, kada je kreirana i da li je sinhronizovana (poslata na server).

Na slici 5 prikazano je kako izgleda lista kod koje ni jedna sesija nije poslata na server.

Krug sa leve strane na svakoj sesiji obojen je sivom bojom, a sa desne strane vidimo ikonicu koja služi za slanje sesije na server.

Slika 5. Lista sesija



Slika 6. Skeniranje koda

5. ZAKLJUČAK

Postoji mnogo mogućnosti za proširenje ovog rešenja. Ukoliko bi se proširo entitet korisnika, dodali novi entiteti i povezali sa bazom podataka stvarne biblioteke, sistem bi se mogao prilagoditi da podržava više mogućih scenarija.

Rešenje može u velikoj meri da olakša posao radnicima biblioteka, s obzirom da ne bi bilo potreban prolazak i upisivanje svake knjige kao što je bio slučaj do sad prilikom inventara.

6. LITERATURA

[1] A Students' Perspective of Native and Cross-Platform Approaches for Mobile Application Development

[2] Holst, A. (2019) Global smartphone shipments forecast from 2010 to 2022 (https://www.statista.com/statistics/263441/global-smartphone-shipmentsforecast/ [2020-02-05 14:12])

[3] Statcounter (2020) Mobile Operating System Market Share Worldwide Aug 2012
– Jan 2020
[4] Bernardes, T.F., Miyake, M.Y.: Cross-platform mobile development approaches: a systematic review. IEEE Lat. Am. Trans. 14(4), 1892–1898 (2016)
[5] The React and React Native Event System Explained: A Harmonious Coexistence
[6] react-native-sensors (https://github.com/react-native-sensors/react-native-sensors)
[7] Code 128/GS1-128 Barcode FAQ & Tutorial (https://www.barcodefaq.com/1d/code-128/)

Kratka biografija:

Ivan Vukašinović rođen je u Prištini 1995. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Primenjene računarske nauke odbranio je 2018.god.
kontakt: ivanvukasino@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42
DOI: https://doi.org/10.24867/10BE35Adamov

SISTEM ZA SPAJANJE BAZA PODATAKA BIBLIOTEČKOG INFORMACIONOG SISTEMA

SYSTEM FOR MERGING LIBRARY INFORMATION SYSTEM DATABASES

Ivan Adamov, *Fakultet tehničkih nauka, Novi Sad*

Oblast – SOFTVERSKO INŽENJERSTVO I INFORMACIONE TEHNOLOGIJE

Kratak sadržaj – *Predmet ovog rada jeste opis sistema za spajanje baza podataka bibliotečkog informacionog sistema BISIS. U prvom delu rada opisan je bibliotečki informacioni sistem BISIS i UNIMARC format. U nastavku je opisana specifikacija sistema, implementacija i rad sistema. Opisana su dva režima rada sistema: pun i inkrementalni. Na kraju rada su opisani dobijeni rezulatati i performanse izvršavanja sistema.*

Ključne reči: *MongoDB, BISIS, spajanje*

Abstract – *The subject of this paper is a description of the system for merging databases of the library information system BISIS. The first part of the paper describes the library information system BISIS and UNIMARC format. The system specification, implementation and operation of the system are described below. Two operating modes of the system are described: full and incremental. At the end of the paper, the obtained results and system execution performance are described.*

Keywords: *MongoDB, BISIS, merge*

1. UVOD

Bibliotečki informacioni sistem BISIS [1] predstavlja sistem za bibliotečko poslovanje. Trenutno obuhvata više od 50 biblioteka širom Srbije, kao i jednu u Austriji, u Lincu. Bibioteke su različitog tipa: javne, visokoškolske, školske i specijalizovane. Podaci ovih biblioteka smeštaju se u zasebne baze podataka.

U okviru ovog sistema postoji sistem za pretragu bibliotečke građe. Kada korisnik unese neki upit, sistem pretražuje zasebne baze podataka i korisniku prikazuje rezultat pretrage. Kako bi se poboljšalo funkcionisanje sistema i izbeglo pretraživanje zasebnih baza podataka, trebalo je sve te podatke spojiti u jednu bazu padataka.
Tema ovog rada jeste implementacija algoritma za spajanje više baza podataka u jednu bazu podataka. Kao testne baze korišćene su baze sledećih biblioteka:
Biblioteka grada Beograda, Gradska biblioteka Novog Sada, Biblioteka šabačka, Biblioteka Milutin Bojić.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.

1.1. Bibliotečki informacioni sistem

Godine 2017. započet je razvoj pete verzije bibliotečkog informacionog sistema BISIS u saradnji sa Bibliotekom grada Beograda, Gradskom bibliotekom u Novom Sadu i Bibliotekom šabačkom. Uvedeni su savremeni tehnološki standardi u bibliotekarstvo u Srbiji, pojednostavljeno je održavanje sistema i omogućena je masovnija primena.

1.2. Najvažnije karakteristike

Najvažnije karakteristike bibliotečkog informacionog sistema BISIS verzije 5.0 [2] su:

- Katalogizacija - potpuno u skladu sa međunarodnim standardom UNIMARC.
- Cirkulacija - cirkulacija (pozajmica) fonda prema praksi srpskih najrazvijenijih biblioteka.
- Izveštavanje - detaljno izveštavanje o građi, pozajmici i drugim aktivnostima.
- Nabavka - automatizovan proces nabavke knjiga povezan sa katalogizacijom.
- OPAC (*online public access catalog*) - pretraživanje fonda za korisnike.
- Objedinjeni katalog - objedinjeni katalog svih biblioteka u sistemu BISIS.
- Digitalni sadržaji - mogućnost skladištenja i pregleda digitalnih sadržaja.
- Uzajamna katalogizacija - preuzimanje zapisa iz drugih biblioteka u sistemu BISIS.
- Otvoreni kôd - kôd BISIS sistema je dostupan svima

1.3. UNIMARC format

UNIMARC (*Universal MARC (Machine Readable Cataloging) format*) [3] je format za obeležavanje i razmenu bibliografskih podataka. Jedan zapis se sastoji od više polja. Svako polje sadrži UNIMARC kôd koji predstavlja identifikator polja i sastoji se od tri cifre. Polja sadrže i podpolja koja se sastoje od identifikatora podpolja i sadržaja podpolja. Identifikator podpolja je predstavljen jednim slovom ili cifrom. Primer kodova UNIMARC formata korišćenih u BISIS sistemu prikazan je u tabeli 1.

Kod polja	Kod podpolja	Značenje
010	a	ISBN
011	a	ISSN
101	a	Jezik knjige
102	a	Zemlja izdavanja
200	a	Naslov knjige
200	e	Podnaslov
700	a	Prezime prvog autora
700	b	Ime prvog autora
210	a	Mesto izdavanja
210	c	Izdavač
210	d	Godina izdavanja
215	a	Broj stranica

Tabela 1. *primer UNIMARC kodova*

2. KORIŠĆENE TEHNOLOGIJE

Aplikacija je napisana u Java programskom jeziku. Podaci bibliotečkog informacionog sistema BISIS čuvaju se u MongoDB [4] bazi podataka. Pored MongoDB baze korišćena je i Redis [5] baza podataka koja služi za povećanje performansi izvršavanja algoritma.

2.1. MongoDB

MongoDB je NoSQL [6] (nerelaciona) baza podataka koja podatke čuva u vidu dokumenata u BSON (*Binary JSON*) [7] formatu koji u velikoj meri liči na JSON (*JavaScript Object Notation*) [8] format. U MongoDB se koriste pojmovi: baza podataka, kolekcija i dokument. Baza podataka može imati nula ili više kolekcija. Jedna kolekcija može imati nula ili više dokumenata. Kolekcija ne određuje šemu baze podataka i ekvivalentna je tabeli u relacionim bazama podataka. Dokument predstavlja jedan zapis u BSON formatu. Dokument u kolekciji ekvivalent je jednom redu u tabeli u relacionoj bazi podataka. Polje dokumenta je isto što i kolona kod relacione baze. Svaki dokument ima jedinstveni identifikator koji je tipa *ObjectId.*

2.1.1. JSON i BSON formati

JSON je tekstualni format i najčešće se upotrebljava za komunikaciju na vebu. Jednostavan je i lako se čita.

BSON predstavlja JSON u binarnom formatu. Kasnije je proširen novim tipovima podataka kako bi poboljšao određene nedostatke JSON formata koji u velikoj meri povećavaju mogućnosti pretrage nad MongoDB bazom. Kroz dokumente u BSON formatu brže se prolazi nego kroz dokumete u JSON formatu. BSON ima brzu serijalizaciju i deserijalizaciju i zauzima više prostora od JSON-a jer čuva dodatne podatke kao što je dužina polja, što omogućava brz prolazak kroz dokument.

2.2. Redis

Redis je veoma brza, NoSQL, *in-memory* baza podataka i najčešće se koristi za keširanje podataka. Podaci se čuvaju u obliku ključ-vrednost i moguće je podesiti vremenski interval čuvanja podataka (posle isteka vremenskog intervala podaci se brišu). Kao vrednost se mogu staviti razni tipovi podataka kao što su: string, lista, mapa, set, sortirani set, i drugi. Redis je projekat otvorenog koda i dostupan je svima.

3. SPECIFIKACIJA SISTEMA

Podaci o bibliotečkim građama pojedninačnih biblioteka BISIS sistema čuvaju se u zasebnim bazama podataka. BISIS sistem obuhvata i takozvani OPAC, sistem za pretraživanje bibliotečkih građa. Taj sistem omogućava pretraživanje bibliotečkih građa po: naslovu, autoru, ključnim rečima, izdavaču i godini izdavanja uz mogućnost dodavanja logičkih operatora: *i, ili, ne*. Kada korisnik sistema unese neki tekst i klikne *enter*, izvršavaju se sledeći koraci:

1. Sistem u pozadini pretražuje određeni Elasticsearch indeks [9].

2. Sistem dobija rezultat u kojem je navedeno u kojim sve bazama se nalazi zadati tekst.

3. Sistem šalje nove upite ka pojedinačnim bazama iz dobijenog rezultata iz koraka 2.

4. Sistem dobija podatke iz pojedinačnih baza i vraća rezultat krajnjem korisniku.

Cilj ovog rada je poboljšanje rada sistema za pretragu, tačnije poboljšanje koraka 3. Umesto da sistem šalje zahteve ka pojedinačnim bazama podataka, treba napraviti jednu bazu koja će čuvati sve podatke bibliotečkog informacionog sistema. Cilj zadatka je implementacija algoritma za spajanje baza podataka u dva režima:

1. Pun režim - spajanje baza podataka “od nule”. Predstavlja spajanje svih podataka pojedinačnih baza u jednu bazu unije.

2. Inkrementalni režim - spajanje novokreiranih podataka iz pojedinačnih baza sa unija bazom.

Kao testne baze koristiti baze sledećih biblioteka: Biblioteka grada Beograda (BGB), Gradska biblioteka Novog Sada (GBNS), Biblioteka šabačka (BS), Biblioteka Milutin Bojić (BMB). Zapise treba sačuvati u bazu unije (UNION). Za kriterijum poređenja koristiti:

1. ISBN

2. ISSN

3. Naslov, autor, izdavač, godina izdavanja

U rezultat ulaze podaci baza i to redom po prioritetu (ako se jedna bibliotečka građa nalazi u dve baze i zapisi u tim bazama imaju različite vrednosti za neko polje, uzeće se vrednost polja iz baze koja ima viši prioritet):

1. BGB

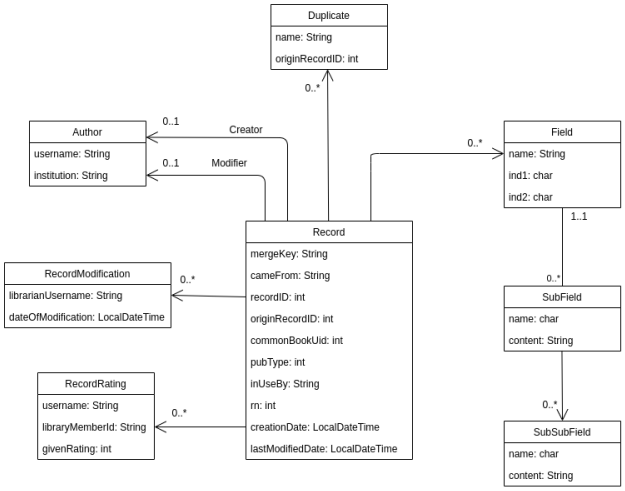
2. GBNS

3. BS

4. BMB

3.1. Dijagram klasa

Klasa *Record* opisuje bibliotečku građu kao i dodatne informacije vezane za pojedinačan zapis (slika 1).



Slika 1. *Dijagram klasa*

4. IMPLEMENTACIJA SISTEMA

Algoritam je implementiran u Java programskom jeziku. Algoritam radi u dva režima: pun i inkrementalni.

4.1. Pun režim

Pun režim predstavlja spajanje baza podataka biliotečkog informacionog sistema od "nule", tj. spajanje svih podataka u jednu bazu unije. Spajanje zapisa se vrši prvo po ISBN-u, zatim po ISSN-u i na kraju po naslovu bibliotečke građe.

Sistem prvo preuzima sve zapise iz pojedinačnih biblioteka koji imaju ISBN, zatim sve zapise koji imaju ISSN, a nemaju ISBN, i na kraju sve zapise koji imaju naslov, a nemaju ISBN i ISSN. Posle preuzimanja svake grupe zapisa radi se spajanje.

Prilikom spajanja baza kada neka bibliotečka građa već postoji u *union* bazi, treba da se uradi ažuriranje tog zapisa. Potrebno je imati Java objekat zapisa da bi se uradile određene operacije poređenja i spajanja dva objekta. Dakle, potrebno je uraditi novi upit ka bazi i dobaviti određeni zapis. Umesto da se upit šalje ka *union* (Mongo) bazi, u Redis bazu se smeštaju svi zapisi koji se upišu u *union* bazu. Ovim načinom se dobija veliko ubrzanje rada algoritma.

Bibliotečka građa se jedinstveno identifikuje ključem zapisa i služi za proveravanje da li bibliotečka građa već postoji u *union* bazi. Primer ključa zapisa za bibliotečku građu "Seobe" dat je na listingu 1.

```
8619018086#_#seobe#nolit#crnjanski milos#1990
```

Listing 1. *Primer ključa zapisa*

Ključ zapisa se sastoji od:

1. Transformisani ISBN, ISBN bez "-" znakova i ako je 13-ocifreni broj izbačena su prva tri broja (978).

2. Transformisani ISSN, ISSN bez "-" znakova.

3. Tranformisani naslov, naslov pretvoren u mala slova latinicom.

4. Tranformisano ime izdavača, ime izdavača pretvoreno u mala slova latinicom.

5. Tranformisano ime autora, ime autora pretvoreno u mala slova latinicom.

6. Godina izdavanja.

7. Separator "#" - služi da odvoji vrednosti.

8. Prazno polje "_" - ako bibliotečka građa ne sadrži neku od vrednosti (ISBN, ISSN, naslov, ...) stavlja se znak "_".

Prilikom pokretanja punog režima, *union* baza je prazna. Prvi korak predstavlja dodavanje BGB zapisa, jer BGB baza ima najveći prioritet. Dodavanje velike količine podataka u Mongo bazu najbrže se izvršava u *batch* režimu. *Batch* režim predstavlja dodavanje zapisa u grupama, npr. dodavanje 1000 zapisa odjednom. Postoje duplikati istih zapisa u pojedinačnim bazama bibliotečkog informacionog sistema. Ovaj sistem vodi računa da se isti zapisi iz istih baza ne upisuju više puta u *union* bazu. BGB zapis koji treba da se sačuva u *union* bazu proširuje se dodatnim podacima kao što su: id zapisa u originalnog bazi i ključ zapisa. Zapis se dodaje u *batch* kontejner i JSON zapis se smešta u Redis bazu. Kada se skupi određen broj zapisa, zapisi se upisuju u *union* bazu u *batch* režimu.

Nakon dodavanja BGB zapisa dobavljaju se zapisi iz ostalih baza (GBNS, BS, BMB) po definisanom prioritetu. Za svaki dobavljeni zapis se generiše ključ zapisa i na osnovu tog ključa se radi provera da li bibliotečka građa već postoji u *union* bazi. Ako zapis ne postoji, dodaće se isto kao i BGB zapis, a ako postoji uradiće se ažuriranje zapisa.

Zapis koji se već nalazi u *union* bazi se dobavlja iz Redis baze na osnovu ključa zapisa i radi se spajanje zapisa iz baze manjeg prioriteta sa zapisom u *union* bazi. Umesto da se radi klasično ažuriranje (*update* upit), prvo se izbrišu svi zapisi koji treba da se ažuriraju i zatim se dodaju ažurirani zapisi. Na ovaj način se postiže veliko ubrzanje rada algoritma.

4.2. Inkrementalni režim

Inkrementalni režim predstavlja spajanje novokreiranih zapisa sa *union* bazom. Zamišljeno je da se pokreće periodično (npr. jednom nedeljno). Kod inkrementalnog režima se umesto pretraživanja zapisa po ISBN-u, ISSN-u i naslovu pretražuju svi zapisi čije je vreme kreiranja veće od vrednosti poslednjeg pokretanja ovog režima rada. Prilikom dodavanja novog zapisa u *union* bazu u punom ili inkrementalnom režimu za svaki zapis se čuva i njegov ključ zapisa. U inkrementalnom režimu se na osnovu ključa zapisa proverava da li zapis postoji u *union* bazi. Ako ne postoji, treba da se doda, a ako postoji, treba da se ažurira. Sistem ima zaštitu od spajanja istog zapisa više puta sa *union* bazom. Kada se završi spajanje snima se poslednje vreme kada je izvršeno spajanje. Poslednje vreme spajanja čuva se u običnom tekstualnom fajlu.

Za razliku od punog režima u ovom režimu rada ne koristi se Redis baza podataka, a provera da li zapis već postoji radi se tako što se šalje upit ka Mongo bazi, dok se ažuriranje zapisa radi na klasičan način (*update* upit).

5. PERFORMANSE

Bitna karakteristika algoritma je i brzina izvršavanja. Prilikom prve implementacije sistemu je bilo potrebno od 10 do 15 dana da spoji četiri baze podataka, da bi se nakon toga u *union* bazi nalazilo preko 400.000 zapisa. Određenim poboljšanjima došlo se do konačne i trenutne verzije algoritma koja četiri baze podataka spoji za oko 220 sekundi.

Kod je bio podeljen na logičke celine i onda se vršilo merenje vremena izvršavanja i analiza logičkih celina. Prvobitno se dodavanje i ažuriranje zapisa radilo jedan po jedan. Uvođenjem *batch* režima postiglo se značajno ubrzanje. Proveravanje postojanja zapisa u *union* bazi kod punog režima se radilo tako što se slao kompleksni upit ka *union* (MongoDB) bazi. Uvođenjem Redis baze podataka i keširanjem dodatih zapisa brzina rada algoritma povećala se oko 22 puta. Ažuriranje zapisa slanjem *update* upita izvršavalo se značajnije sporije nego korišćenjem hibridnog pristupa. Svi zapisi koje je potrebno ažurirati čuvaju se u nekoj listi, zatim se prvo izbrišu iz *union* baze, pa se dodaju. Dakle umesto *update* upita koriste se *delete* + *insert*. Ovaj pristup je doneo veliko ubrzanje rada algoritma.

6. ZAKLJUČAK

Bibliotečki informacioni sistem BISIS predstavlja veoma značajno unapređenje bibliotečkog poslovanja. Sistem se unapređivao i unapređivaće se kako godine budu odmicale. Sistem za spajanje baza podataka predstavlja jedan korak u unapređenju rada sistema.

Početni korak prlikom razvoja sistema bio je analiza tehnologija koje se koriste u bibliotečkom informacionom sistemu kao i analiza karakteristika UNIMARC formata koji služi za reprezentaciju bibliografske građe. U radu su opisane tehnologije za razvoj sistema za spajanje baza podataka, kao i UNIMARC format sa datim primerima. Data su poređenja, prednosti i mane JSON i BSON formata podataka. Naredni korak bio je razvoj samog sistema. Sistem omogućava formiranje jedinstvene baze UNIMARC zapisa. Korišćenjem određenih metoda uočeni su zapisi koji predstavljaju identična izdanja i implementiran je algoritam za spajanje tih zapisa. Prilikom spajanja baza ispoštovana su pravila prioriteta baza. Poređenje zapisa je *case* i *script insensitive*, dakle podaci se svode na isti format - "ošišanu" latinicu malim slovima. ISBN i ISSN formati su svedeni na isti format - desetocifreni broj bez crtica.

Nakon završetka inicijalne verzije sistema, sistem u punom režimu je radio dobro, ali sporo. Velika ubrzanja postignuta su keširanjem podataka i uvođenjem Redis baze podataka, korišćenjem *batch* režima rada i hibridnog ažuriranja (*delete* + *insert* umesto *update* upita). Krajnja verzija sistema radi dobro i brzo. Sistem je otporan na loše podatke, dobro uočava duplikate i lako ga je proširiti na rad sa više baza podataka.

Dalji pravci razvoja sistema se ogledaju u proširivanju sistema na rad sa više baza podataka i optimizaciji zauzeća memorije koje zauzimaju keširani podaci.

7. LITERATURA

[1] https://bisis.rs/biblioteke.html

[2] https://bisis.rs

[3] https://www.ifla.org/publications/unimarc-formats-and-related-documentation

[4] https://en.wikipedia.org/wiki/MongoDB

[5] https://en.wikipedia.org/wiki/Redis

[6] https://en.wikipedia.org/wiki/NoSQL

[7] https://en.wikipedia.org/wiki/BSON

[8] https://www.json.org

[9]https://www.elastic.co/blog/what-is-an-elasticsearch-index

Kratka biografija:

Ivan Adamov rođen je 30.06.1996. u Zrenjaninu. Srednju Elektrotehničku i građevinsku školu "Nikola Tesla" završio je 2015. godine u Zrenjaninu. Iste godine upisuje Fakultet tehničkih nauka u Novom Sadu, smer Softversko inženjerstvo i informacione tehnologije. Studije završava 2019. godine. Iste godine upisuje master akademske studije na istom fakultetu, smer "Elektronsko poslovanje". Master rad odbranio je 2020. godine.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

УДК: 621.39
ДОИ: https://doi.org/10.24867/10BE36Makaji

ANDROID ARCHITECTURE COMPONENTS

ANDROID ARCHITECTURE COMPONENTS

Алексеј Макаји, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У раду су описане Android компоненте архитектуре и њихова имплементација уз коришћење шаблона чисте архитектуре како би се добила робусна, модуларна, тестабилна и одржива апликација. Дато имплементирано пројектно решење је апликација за праћење листе производа које треба купити и које се могу поделити између пријатеља у реалном времену. Апликација је написана у Kotlin програмском језику за Android платформу. У закључку рада су изнете предности и мане компонената архитектуре које су коришћене уз чисту архитектуру.*

Кључне речи: *Компоненте архитектуре, Android, Kotlin, чиста архитектура*

Abstract – *The thesis describes Android architecture components and their implementation using clean architecture pattern for developing a robust, testable, modular and maintainable application. The given implemented project solution is an application for tracking lists of products to buy which are shareable among friends in real time. The application is written in Kotlin programming language for the Android platform. The conclusion contains advantages and disadvantages of Android architecture components used with the Clean architecture pattern.*

Keywords: *Architecture components, Android, Kotlin, Clean architecture*

1. UVOD

Развој нативне *Android* апликације (*native Android application)* даје најбоље перформансе за разлику од хибридне и веб мобилне апликације. Међутим, њена мана је дуже време потребно за развој саме апликације као и захтевање већег знања програмера из разлога што програмер поред програмског језика J*ava* мора да познаје и *Android* радни оквир (*Android framework*).

Да би се решио споменути проблем нативне *Android* апликације, компанија *Google* која највише доприноси развоју *Android* платформе, понудила је компоненте архитектуре (*architecture components*) које за циљ имају да олакшају програмеру развој апликације.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Стеван Гостојић, ред. проф.

Ако неко улази по први пут у програмирање нативне *Android* апликације, компоненте архитектуре треба да служе као неке смернице по којима би се програмер водио, односно коришћење истих представља најбољу праксу како би се написала робусна, тестабилна и продукционо квалитетна апликација [1].

2. КОМПОНЕНТЕ АРХИТЕКТУРЕ

Библиотеке које чине компоненте архитектуре су следеће [2]:

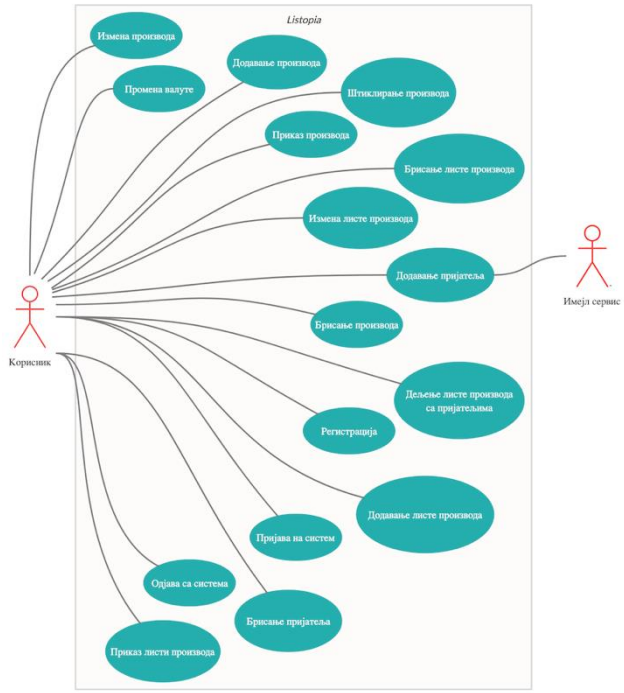
- *Везивање података (data binding)* – декларативно везивање *UI* компонената у распоредима за изворе података
- *Животни циклус (lifecycle)* – управљање животним циклусом компонената активности, фрагмената итд.
- *Живи подаци (livedata)* – обавештавање погледа (*views*) када се деси нека промена над базом података
- *Навигација (navigation)* – управљање свим неопходним стварима за навигацију у апликацији
- *Парцијално повлачење података (paging)* – парцијално повлачење података на захтев извора података *(data source)*
- *Room* – апстрактни слој над *SQLite* базом података за робусни приступ подацима из саме базе података
- *ПогледМодел (ViewModel)* – управљање подацима повезаним са корисничким интерфејсом на начин који је усмерен на животни циклус компонената
- *Радни менаџер (workmanager)* – управљање позадинским пословима (*background jobs*) у апликацији.

3. СПЕЦИФИКАЦИЈА ЗАХТЕВА

Да би софтверско решење било успешно и произвело очекивано решење, потребно је специфицирати захтеве апликације који треба да буду испуњени. Захтеви се могу класификовати на функционалне и нефункционалне захтеве.

3.1 Функционални захтеви

Функционални захтеви су илустровани уз помоћ дијаграма случаја коришћења (*Use-case diagram) (слика 1)*.



Слика 1. Дијаграм случаја коришћења

Сви релевантније случајеве коришћења за апликацију су:

- *Пријава на систем*
- *Регистрација*
- *Одјава са система*
- *Приказ листи производа*
- *Додавање листе производа*
- *Измена листе производа*
- *Брисање листе производа*
- *Приказ производа*
- *Штиклирање производа*
- *Додавање производа*
- *Измена производа*
- *Брисање производа*
- *Дељење листе производа са пријатељима*
- *Додавање пријатеља*
- *Брисање пријатеља*
- *Промена валуте*

3.2 Нефункционални захтеви

Нефункционални захтеви за имплементирано пројектно решење су следећи:

- *Софтверски захтеви*
- *Инсталациони захтеви*
- *Безбедност*
- *Перформансе*
- *Поузданост*
- *Одрживост*

4. СПЕЦИФИКАЦИЈА ДИЗАЈНА

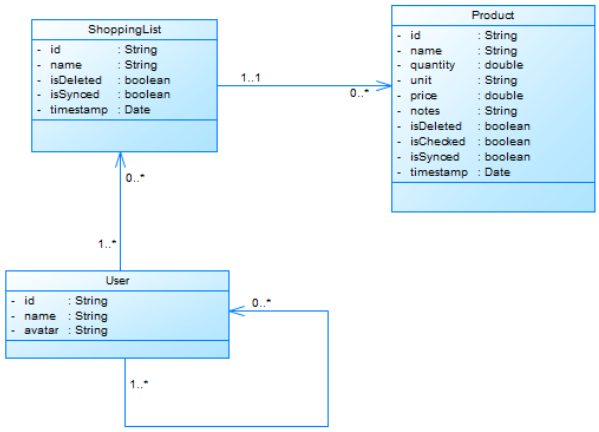
У овом одељку описан је дизајн система са динамичким и статичким моделима система, као и архитектура апликације.

4.1 Дијаграм компоненти

Систем је подељен у две целине, где једну целину представљају уређаји са *Android* оперативним системом, а другу *Firebase cloud* сервиси. Корисник на инсталираној апликацији извршава акције у интеракцији са *UI* контролерима који комуницирају са ПогледМоделима, а они припремају податке за приказ позивајући функције које имплементирају пословну логику апликације. Те функције позивају репозиторијум путем његовог интерфејса, где репозиторијум одлучује да ли ће се обрада података извршити са локалном базом података или са сервером. У случају да се врши комуникација са локалном базом података *Room*, она се врши путем дефинисаног локалног интерфејса базе података, где постоје *CRUD* операције као и додатни упити за извлачење података. У случају да се врши комуникација са сервером, она се врши путем дефинисаног *Retrofit* интерфејса за комуникацију са сервером.

4.2 Дијаграм класа

Дијаграм класа (слика 2) описује модел података који је потребан за дато софтверско решење за вођење евиденције о томе које производе треба купити, који могу бити подељени између пријатеља. На дијаграму се налазе ентитети потребни ради испуњења функционалности система.



Слика 2. *Дијаграм класа*

4.3 Архитектура

Да би се искористиле све компоненте архитектуре, најпогоднији шаблон је управо *MVVM*, односно модел-поглед-погледмодел са чистом архитектуром како би апликација била што модуларнија и лакша за одржавање.

4.3.1 *MVVM* шаблон

MVVM (*Model-View-ViewModel*) односно модел-поглед-погледмодел шаблон је измишљен од стране архитеката из *Microsoft* компаније како би се преместила пословна логика из разних погледа, односно *UI* контролера у неке друге класе [3].

4.3.2 Чиста архитектура (*Clean architecture*)

Шаблон чисте архитектуре је представио *Robert C. Martin* (*Uncle Bob*) [4], која за циљ има развијање модуларне, флексибилне и лако тестабилне апликације. Њом се превазилазе проблеми тешког одржавања кода. Чиста архитектура има за циљ да раздвоји

битне тачке у апликацији, што се постиже раздвајањем апликације на више слојева.

4.3.3 Слојеви архитектуре

Разликујемо три слоја архитектуре [5] где:

- Презентационом слоју припадају ПогледМодели и *UI* контролери односно активност, фрагменти и произвољни погледи.
- Доменском слоју припадају случајеви коришћења који имају за циљ да изврше тачно одређен задатак, односно, одређену пословну логику.
- Слоју података припада репозиторијум који служи за обраду података, односно одлучује из ког извора података ће се подаци добавити или обрадити.

4.3.4. Један извор података

Како апликација подржава офлајн и онлајн мод, постоји опасност да подаци који се приказују буду неконзистентни ако се подаци добављају са више различитих извора. Из тог разлога архитектура поштује правило једног извора података (*single source of truth*) где је то локална база података *Room*. Правило каже да подаци за приказ кориснику треба да се добављају само са једног извора података [5].

5. ИМПЛЕМЕНТАЦИЈА

У овом одељку описана је имплементација сваке *Android* компоненте архитектуре за дато софтверско решење. Такође, описана је структура пројекта као и библиотеке које се користе за реализацију апликације.

Cloud функције и *Firestore Cloud* база података укратко су описани јер се користе на серверској страни за имплементацију датог решења.

5.1 Структура пројекта

Структура пројекта је организована по пакетима који су смештени унутар врховног пакета који има назив апликације и секције *Gradle Scripts* где се налазе фајлови и скрипте за покретање апликације, као и увезивање екстерних зависности попут библиотека и других модула.

5.2 Коришћене библиотеке за реализацију пројекта

Поред библиотека компонената архитектуре које су већ биле описане у раду, коришћене су и следеће библиотеке за реализацију пројектног решења: *Dagger2, Material Design, Support libraries, Kotlin coroutines, Retrofit2, OkHttp3, FirebaseUI, Firebase Messaging, Preference, Glide, Stetho, Timber.*

5.3 *Cloud* функције

Како *Cloud* функције могу бити написане у различитим окружењима, за ово пројектно решење су функције написане у *Node.js* окружењу у *TypeScript* програмском језику. У пројектном решењу функције су *HTTPS* функције које гађа *Android* апликација путем *HTTPS* протокола где функције врше одређену пословну логику и враћају *HTTPS* одговор. Функције комуницирају са *Cloud Firestore* базом података где су смештени серверски подаци за мобилну апликацију [6].

5.4 *Cloud Firestore* база података

Cloud Firestore база података је скалабилна *NoSQL* база података направљена од стране *Google*-а. Модел података базе подржава флексибилну хијерархијску структуру података. Подаци се чувају у документима који имају поља мапирана на њихове вредности. Документи су организовани по колекцијама, на основу којих се пишу упити [7].

5.5 Имплементација компонената архитектуре

У претходном тексту је описана свака појединачна *Android* компонента архитектуре, а надаље ће бити приказана њихова имплементација која је примењена у датом пројектном решењу.

5.5.1 *Room*

Имплементација локалне базе података *Room* захтева да се одреде класе које представљају ентитете у бази података. Сваки ентитет мора имати примарни кључ који се одређује анотацијом ***PrimaryKey***. Једном када се дефинишу ентитети, како би се манипулисало подацима неопходно је за сваки ентитет креирати објекат за приступ подацима *DAO*. Потребно је још иницијализовати *Room* базу и повезати ентитете са објектима приступа података.

5.5.2 Животни циклус, живи подаци и ПогледМодел

Да би класа наследила особине ПогледМодела, потребно је да дата класа наследи *ViewModel* класу, која има најбитнију методу, а то је *onCleared* која се аутоматски позове када се поглед у ком је био креиран ПогледМодел уништи. Тиме се чисти меморија и инстанца класе ПогледМодела се уништава, а све је то омогућено јер су погледи свесни животног циклуса. Како је ослушкивање повезано са животним циклусом, подаци ће се ослушкивати све док је поглед у активном стању.

5.5.3 Навигација

Да би се имплементирала навигација, потребно је прво креирати навигациони граф. Приликом креирања навигационог графа, може се користити дизајн опција за преглед и рад у *Android Studio* радном окружењу где додавање дестинација, подешавање параметара, додавања рута и других ствари генерише аутоматски *XML* код. Други начин јесте ручно писање *XML* кода. Сваки фрагмент представља једну дестинацију, где су дефинисане акције над њима, што представља руту до друге дестинације. Позивањем те акције, прелази се са једне дестинације на другу уз прослеђене параметре по потреби.

5.5.4 Парцијално повлачење података

Библиотека нуди *PageList*-у која је направљена да ради са извором података и нуди класу да се на основу извора података креира живи податак са *PageList*-ом како би се олакшала имплементација парцијалног учитавања података.

5.5.5 Радни менаџер

Први корак ка имплементацији радног менаџера јесте креирање *Worker* класе, односно класе која треба да обави неки позадински посао када су испуњени задати услови. Класа за синхронизацију производа треба да наследи *Worker* класу која има једну круцијалну методу коју треба изменити, а то је метода *doWork* која се позива једном када су задати услови испуњени и радни менаџер спреман да изврши задатак.

5.5.6 Везивање података

Да би се омогућило коришћење библиотеке везивања података, потребно је у *gradle* фајлу у ***android*** блоку кода укључити опцију везивања података са ***buildFeatures {dataBinding true}***. Да би везивање података било доступно у распоред фајловима, потребно је ставити *layout* таг као коренски елемент у *XML* фајлу, чиме се компајлеру назначава да изгенерише аутоматски класу која ће бити доступна програмеру у коду преко које ће моћи приступити елементима и варијаблама дефинисаним у *XML* фајлу.

6. ДЕМОНСТРАЦИЈА

Listopia је мобилна апликација која за циљ има да омогући лак начин праћења листе производа које треба купити, које могу бити подељене између пријатеља или чланова породице. Свака промена листе и производа, као и штиклирање производа се манифестује у реалном времену ако је листа подељена између пријатеља под условом да је интернет конекција доступна. Сви подаци се чувају у *Firestore Cloud* бази података.
Апликација може да ради у онлајн и офлајн режиму. Такође, корисник може да буде пријављен на систем чиме добија додатне функционалности апликације, а то су чување и ишчитавање листе производа са система, додавање и брисање пријатеља као и дељење листе производа између пријатеља. Ако корисник није пријављен на систему, његови подаци се чувају само у локалној бази уређаја, све док се корисник не улогује на систем, чиме ће се аутоматски синхронизовати његове листе производа.

7. ЗАКЉУЧАК

Библиотеке описане у овом раду су тако дизајниране да заједничким коришћењем знатно олакшавају имплементације разних програмских решења као и да решавају недостатке *Android* платформе.
У раду је примећено да су од испробаних библиотека, библиотеке животни циклус, живи подаци и Поглед-Модел изразито погодне када се користе заједно у пројекту. У случају када је у софтверском решењу потребно парцијално учитавање података из локалне базе, установљено је да је *Room* библиотека пожељна за коришћење са библиотеком парцијалног учитавања података јер подржава повратни тип података као извор података што знатно смањује имплементацију датог решења.
У случају да у пројекту има потребе да се неки задаци изврше одложено или ако постоји неки услов да се ти задаци изврше, препорука је да се користи библиотека радни менаџер, што данас и *Google* предлаже.
Како библиотека навигације нуди нови приступ навигацији, визуелни приступ омогућује програмеру да има увид у то које су му све дестинације, односно екрани доступни за навигацију из жељеног екрана.
Битно је напоменути да је архитектура апликације веома битна да би се добила крајње квалитетна, модуларна и лако одржива апликација, а чиста архитектура уз *Android* компоненте архитектуре омогућава управо то.

8. ЛИТЕРАТУРА

[1] *Android developers* - https://developer.android.com/ (последњи приступ 23.02.2020.)
[2] Android jetpack - https://developer.android.com/jetpack (последњи приступ 14.08.2020.)
[3] *MVVM* - https://code.msdn.microsoft.com/How-to-implement-MVVM-71a65441 (последњи приступ 23.02.2020.)
[4] The Clean Architecture by Robert C. Martin (Uncle Bob) - https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (последњи приступ 11.07.2020.)
[5] Препоручена архитектура - https://developer.android.com/jetpack/guide (последњи приступ 18.07.2020.)
[6] *Cloud* функције - https://cloud.google.com/functions/docs/concepts/overview (последњи приступ 14.08.2020.)
[7] *Firestore* база података - https://firebase.google.com/docs/firestore (последњи приступ 14.08.2020.)

Kratka biografija:

Алексеј Макаји рођен је у Бачкој Тополи 1991. год. Дипломски рад на Факултету техничких наука из области Електротехнике и рачунарства – Рачунарство и аутоматика одбранио је 2016. год.

Контакт: makso.the.one@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.43
DOI: https://doi.org/10.24867/10BE38Banovic

PRORAČUN LOKALNE RASPODELE SNAGE GUBITAKA U MAGNETNOM KOLU IPMSM-A PRIMENOM FEA

LOCAL IRON LOSS POWER DENSITY CALCULATION OF IPMSM USING FEA

Milica Banović, Dejan Jerkan, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je napravljen pregled najzastupljenijih modela proračuna gubitaka u magnetnom kolu električnih mašina. Na primeru sinhronog motora sa utisnutim stalnim magnetima (IPMSM), uz primenu metode konačnih elemenata (FEA), izvršen je proračun lokalne raspodele gustine snage magnetnih gubitaka. Pored ukupnih gubitaka, izračunate su i njegove pojedinačne komponente (histerezisni gubici i gubici usled vrtložnih struja), kako za stator, tako i za rotor. Korišćeni su standardni Jordan-ov model i modifikovani Jordan-ov model.*

Ključne reči: *FEA, IPMSM, gubici u gvožđu, histerezisni gubici, gubici usled vrtložnih struja.*

Abstract – *This paper gives an overview of most prominent iron loss calculation models used in electrical machine analysis. Local iron loss power density distribution of analysed IPMSM is calculated using FEA. Besides total iron losses, both hysteresis and eddy current losses contribution components are analysed and calculated separately for stator and rotor. Standard Jordan model and modified Jordan model were used.*

Keywords: *FEA, IPMSM, iron losses, hysteresis losses, eddy current losses*

1. UVOD

Savremena istraživanja se intenzivno sprovode na polju razvijanja visoko efikasnih električnih pogona koji koriste sinhrone motore sa stalnim magnetima (IPMSM) i od kojih se zahteva velika gustina snage i optimizovane performanse u širokom opsegu brzina.

Zbog toga se velika pažnja posvećuje analizi i predikciji lokacije i intenziteta gubitaka u električnim motorima. Jedni od ključnih gubitaka u mašini jesu gubici u gvožđu tj. gubici u magnetnom kolu mašine.

Kod IPMSM gubici u gvožđu su najintenzivniji u zupcima i jarmu statora i površinskom delu jarma rotora, u okolini vrhova magneta, tamo gde je i promena magnetnog polja najizraženija. Pri radu na velikim brzinama, u oblasti slabljenja polja, posebno dolaze do izražaja gubici u gvožđu, koji tada mogu postati i najdominantnija komponenta gubitaka u mašini.

Inženjerski pristup nastoji da gubitke u gvožđu empirijski tretira razdvojeno (gubici usled makroskopskih vrtložnih struja i histerezisni gubici) u cilju pokušaja razdvajanja različitih fizičkih uticaja usled promene frekvencije i varijacija magnetnog polja. Šta više, empirijski modeli gubitaka su pokazali veoma dobru korelaciju sa merenjima i pružaju veoma brz i jednostavan način procene gubitaka u gvožđu zbog čega se i dalje, najčešće koriste pri dizajnu električnih mašina.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dejan Jerkan, docent.

2. MODELI PRORAČUNA GUBITAKA U GVOŽĐU

Analitički modeli gubitaka se uglavnom zasnivaju na empirijskoj *Schteinmetz*-ovoj jednačini ili na *Jordan*-ovom modelu. Osnovna prednost analitičkih modela je njihova jednostavnost i malo procesno vreme. Ovi modeli se pokazuju kao pouzdani u uslovima do vrednosti magnetne indukcije 1,5 T i frekvencije 700 Hz.

2.1. Modeli koji se zasnivaju na *Schteinmetz*-ovoj jednačini

Originalni *Schteinmetz*-ov model [1] gubitke u gvožđu po jedinici zapremine p $[mW/cm^3]$ izražava empirijskom jednačinom:

$$p = C_m f^{\alpha} \hat{B}^{\beta}, \tag{1}$$

gde je $\hat{B}$ amplituda i f frekvencija prostoperiodičnog signala magnetne indukcije. U *Schteinmetz*-ovoj jednačini figurišu tri *Schteinmetz*-ova koeficijenta C_m, α i β, koji predstavljaju empirijski određene parametre materijala fitovanjem *B-H* histerezisne krive. Eksponenti α i β su racionalni brojevi u opsegu $\alpha\epsilon[1,3]$ i $\beta\epsilon[1,2]$ .Najveća mana ovog modela jeste što je namenjen samo za situacije kada je magnetna indukcija prostoperiodičan signal, što često nije slučaj, naročito kod rotacionih mašina sa promenljivom geometrijom vazdušnog zazora izazvanom obrtanjem.

Modifikovani *Schteinmetz*-ov model [2] predstavlja jedan način proširenja originalnog modela za složenoperiodične talasne oblike. Osnovna ideja ovog modela je uvođenje ekvivalente frekvencije koja zavisi od izvoda $\frac{dB(t)}{dt}$ kao:

$$f_{eq} = \frac{2}{\Delta B^2 \pi^2} \int_0^{\mathrm{T}} \left(\frac{dB(t)}{dt}\right)^2 dt\,, \tag{2}$$

gde je $\Delta B = B_{max} - B_{min}$ i T period signala elektromagnetne indukcije. Modifikovana jednačina srednje snage gubitaka u gvožđu po jedinici zapremine p_v $[mW/cm^3]$ se izražava jednačinom:

$$p = C_m f_{eq}^{\alpha-1} \hat{B}^{\beta} \frac{1}{\mathrm{T}}. \tag{3}$$

DC premagnetizacija se može uzeti u obzir korekcionim koeficijentom. Generalizovani *Schteinmetz*-ov model [5] iskazuje snagu gubitaka u gvožđu kao jednoznačnu funkciju magnetne indukcije:

$$p = \frac{1}{T} \int_0^T C_m \left|\frac{dB(t)}{dt}\right|^{\alpha} |B(t)|^{\beta-\alpha}\, dt. \tag{4}$$

Ova jednačina ne uzima u obzir prethodne vrednosti elektromagnetne indukcije i ova jednačina je DC senzitivna tj. ne zahteva uvođenje korekcionog faktora usled postojanja DC komponente.

2.2. Modeli koji se zasnivaju na *Jordan*-ovom modelu

U [3] predstavljen je *Jordan*-ov model (pod kojim se veoma često u literaturi podrazumeva *Schteinmetz*-ov model) u kom su ukupni gubici u gvožđu razdvojeni na dva dela, na statičke histerezisne gubitke p_h i dinamičke gubitke usled vrtložnih struja p_{ed}. Srednja vrednost snage gubitaka u gvožđu po jedinici mase [*W/kg*] tokom jednog perioda osnovnog harmonika magnetne indukcije $T = \frac{1}{f}$ jednaka je:

$$p = p_h + p_{ed} = c_h \hat{B}^{\alpha} f + c_{ed}^{sin} \hat{B}^2 f^2 \qquad (5)$$

U ovom modelu histerezisni gubici se smatraju proporcionalni površini statičke ($f \to 0$) histerezisne petlje. Histerezisni gubici rezultat su unutrašnjih procesa u feromagnetskom materijalu koji rasipa energiju e_h tokom jednog perioda T, uzimajući u obzir periodičnu promenu magnetne indukcije. e_h zavisi od amplitude magnetne indukcije $\hat{B}$, ali i od prisustva viših harmonika u složeno-periodičnom signalu $B(t)$, koji će se oslikati u prisustvu manjih histerezisnih petlji unutar histerezisne petlje osnovnog harmonika. Za prostoperiodičnu promenu magnetne indukcije $B(t)$, $e_h = c_h \hat{B}^{\alpha}$, gde je c_h parametar materijala, a vrednost koeficijenta $\alpha \epsilon [1.5, 2]$.

Gubici usled vrtložnih struja p_{ed} proporcionalni su kvadratu amplitude prostoperiodične magnetne indukcije $\hat{B}$ i kvadratu frekvencije f. c_{ed}^{sin} je složeni koeficijent i računa se kao $c_{ed}^{sin} = \frac{\sigma \pi^2 d^2}{6\rho}$, gde je σ – električna provodnost materijala lima, ρ – gustina materijala i d – debljina lima.

Gubici usled vrtložnih struja se mogu izračunati uz primenu Maksvelovih jednačina kao:

$$p_{ed} = \frac{\sigma d^2}{12\rho} \left(\frac{dB(t)}{dt} \right)^2. \qquad (6)$$

Ovaj model je jedan od najčešće korišćenih u praksi i veoma često, kada je nesinusoidalnost talasnog oblika magnetne indukcije izražena, gubici usled vrtložnih struja se računaju kao suma gubitaka usled vrtložnih struja za svaki red harmonika ponaosob. Tada, srednja vrednost snage gubitaka u gvožđu po jedinici mase [*W/kg*] jednaka je:

$$p = p_h + p_{ed} = c_h \hat{B}^{\alpha} f + \sum_{i=1}^{n} c_{ed}^{sin} \hat{B}_i^{\,2} f_i^2, \qquad (7)$$

gde je i red harmonika. *Pry* i *Bean* [4] su predložili jedan način za korekciju *Jordan*-ovog modela, uvođenjem empirijskog korekcionog faktora μ_{exc} (*excess loss factor*). Modifikovana je jednačina ukupnih gubitaka u gvožđu, za sinusoidalni talasni oblik magnetne indukcije, kao:

$$p = p_h + p_{ed} = c_h \hat{B}^{\alpha} f + \mu_{exc} c_{ed}^{sin} \hat{B}^2 f^2, \text{ gde je } \mu_{exc} = \frac{p_{\text{mereno}}}{p_{\text{proračunato}}} > 1. \qquad (8)$$

Drugi pristup za poboljšanje *Jordan*-ovog modela predlažio je *Bertotti* uvođenjem dodatnih gubitaka p_{exc} kao funkciju frekvencije i amplitude elektromagnetne indukcije. *Bertotti*-ev model ukupne gubitke u gvožđu tretira kao zbir histereznih gubitaka p_h, gubitaka usled vrtložnih struja p_{ed} i dodatnih gubitaka p_{exc}:

$$p = p_h + p_{ed} + p_{exc} = c_h \hat{B}^{\alpha} f + c_{ed}^{sin} \hat{B}^2 f^2 + c_{exc} f^{1.5} \hat{B}^{1.5}. \qquad (9)$$

Bertotti je uveo pojam magnetnih objekata i razvio teoriju kojom je fizički opisao i izveo funkcionalnost dodatnog koeficijenta c_{exc} kao $c_{exc} = \sqrt{SV_0 \sigma G}$, gde je S површина poprečnog preseka lima, G bezdimenzioni koeficijent $G \approx 0{,}136$ и σ električna provodnost lima. V_0 karakteriše lokalnu statističku raspodelu koercitivnog polja uz uvažavanje veličine Vajsovih domena.

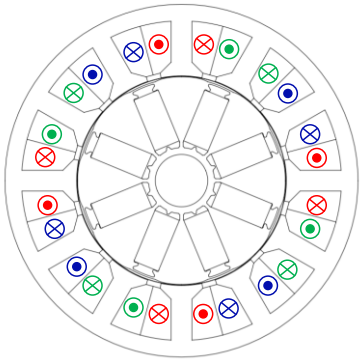
Originalni *Bertotti*-ev model podrazumeva sinusoidalnu magnetizaciju ili vrlo malo prisustvo izobličenja i bilo je potrebno proširiti *Bertotti*-ev model za slučaj nesinusoidalnog signala elektromagnetne indukcije.

Fiorillo i *Novikov* [6], a kasnije i *Barbisio* [7] su razvili originalnu *Bertotti*-evu formulu gubitaka u gvožđu tako da važi za svaki talasni oblik magnetne indukcije.

3. PRAKTIČAN PRIMER

Postupci za proračun gubitaka u gvožđu, koji su zasnovani na *Jordan*-ovim modelima, u ovom odeljku će biti primenjeni i verifikovani na primeru IPMSM. Reč je o motoru namenjenom širokom opsegu brzina, što proračun gubitaka čini još izazovnijim zadatkom.

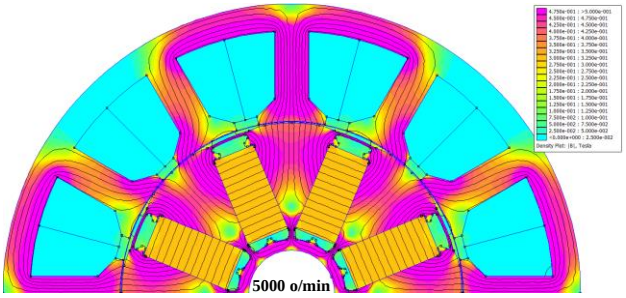
Odabrani sinhroni motor namenjen je za pogon električnih uređaja za široku upotrebu, kao što je mašina za pranje (sušenje) veša. Napaja se iz statičkog energetskog pretvarača, a sam pogon je realizovan bez davača pozicije/brzine tj. reč je o *sensorless* pogonu relativno niskih performansi ali namenjenom za masovnu upotrebu u širokom opsegu brzina i momenta. Motor je osmopolni, trofazni, sa koncentrisanim statorskim namotajem načinjenim od aluminijuma. Sprega namotaja je zvezda, za nominalni napon od 200 *V*. Spoljašnji prečnik statorskog lim paketa iznosi 102 *mm*. Utisnuti stalni magneti su izrađeni od ferita. Nominalna brzina obrtanja motora je 5000 *o/min*. Za brzine preko nominalne pogon je realizovan implementacijom slabljenja polja putem injektovanja negativne struje po *d* osi.



Slika 1. *Izgled poprečnog preseka motora*

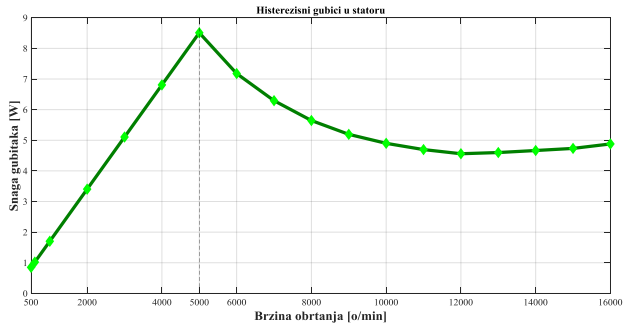
Na slici 1 je prikazan poprečni presek lim paketa analizirane mašine sa principskom dispozicijom statorskog trofaznog namotaja, koji je izveden kao koncentrični. Nad modelom motora, kreiranim u programskom paketu *FEMM 4.2.* [8] izvršene su magnetostatičke simulacije kojima se oponaša rad motora u režimu idealnog praznog hoda, za različite brzine obrtanja, koje se kreću u opsegu 5000 – 16000 *o/min*. Kao odziv ovih simulacija se dobijaju vrednosti vektora magnetske indukcije u svakom od konačnih elemenata, koji se zatim koriste za proračune lokalnih vrednosti snage gubitaka po priloženim modelima.

Na slici 2 je prikazan odziv magnetostatičke simulacije kojom se oponaša rad motora u režimu praznog hoda pri brzini obrtanja od 5000 *o/min*. Ovaj radni režim predstavlja gornju granicu brzine pri kojoj se još uvek ne primenjuje slabljenje polja injektovanjem negativne struje po *d* osi.



Slika 2. *Simulacija za brzinu obrtanja 5000 o/min*

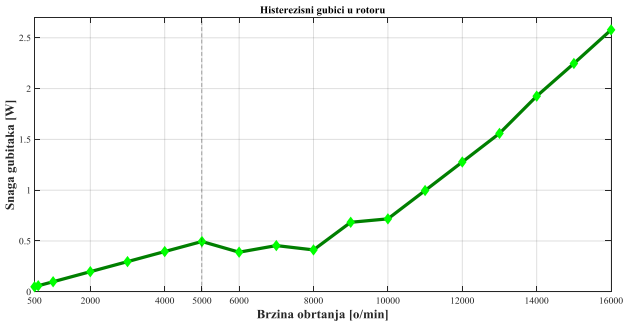
Na slici 3 su prikazani histerezisni gubici u magnetnom kolu statora u zavisnosti od brzine obrtanja. U oblastima brzina do 5000 *o/min* je primetan gotovo linearan porast, dok se u oblastima slabljenja polja za brzine preko 5000 *o/min* beleži njihov blagi pad. U oblasti brzina koje su manje od 5000 *o/min* nije primenjivano slabljenje polja, pa histerezisni gubici rastu sa porastom frekvencije od koje su linearno zavisni. Za brzine preko 5000 *o/min* je primenjeno slabljenje polja injektovanjem negativne struje po *d* osi, usled kojeg dolazi do bržeg pada vrednosti indukcije u odnosu na porast frekvencije, pa otuda i pad histerezisnih gubitaka.



Slika 3 *Histerezisni gubici u statoru po Jordan-ovom modelu proračuna*

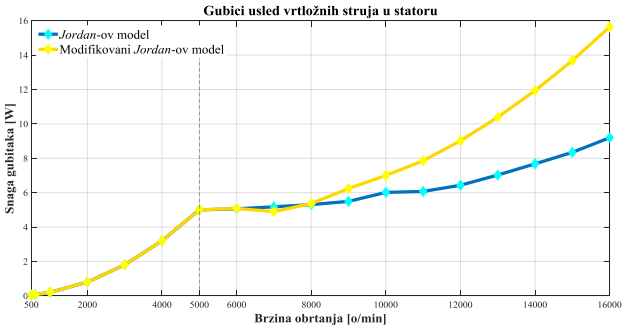
Na slici 4 su prikazani histerezisni gubici u magnetnom kolu rotora u zavisnosti od brzine obrtanja. Ponovo je u oblastima brzina do 5000 *o/min* očit praktično linearan porast ovih gubitaka, dok se u oblastima brzina preko 5000 *o/min*, za razliku od histerezisnih gubitaka u magnetnom kolu statora, beleži njihov značajan porast. Pojašnjene ovog fenomena se pronalazi u činjenici da slabljenje polja, koje se izvodi u cilju smanjenja magnetnog polja u statoru ne manifestuje u značajnoj meri u magnetnom kolu rotora, te se sa porastom frekvencije povećavaju i histerezisni gubici.

Važno je primetiti i da je frekvencija promene magnetnog polja u površinskom sloju rotora, u kojem su inače ove varijacije polja dominantno grupisane, tačno tri puta veća od frekvencije promene polja u statoru, što se objašnjava postojanjem tri statorska žleba po paru polova, koji svojom geometrijom unose varijacije u magnetnom polju rotora tačno tri puta u električnoj periodi.



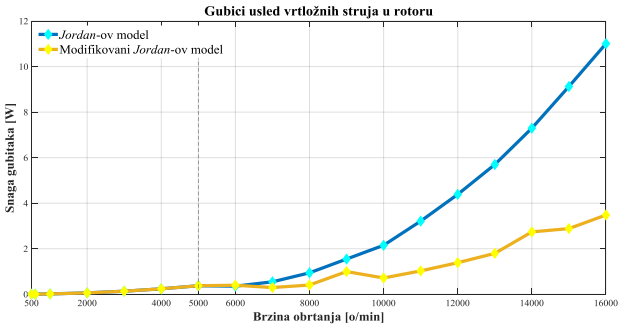
Slika 4 *Histerezisni gubici u rotoru po Jordan-ovom modelu proračuna*

Na slici 5 su prikazani gubici usled vrtložnih struja u magnetnom kolu statora u zavisnosti od brzine obrtanja, primenom *Jordan*-ovog i modifikovikovanog *Jordan*-ovog modela proračuna. Podsećanja radi, modifikovani model praktično unosi razlike samo u proračunu gubitaka usled vrtložnih struja. Primetan je porast gubitaka sa porastom brzine primenom oba modela, jer je zavisnost ovih gubitaka od frekvencije značajno izraženija od histerezisnih gubitaka. Očekivano je i da modifikovani Jordan-ov model daje veće vrednosti gubitaka, jer uvažava udeo harmonijskog izobličenja vektora magnetne indukcije.



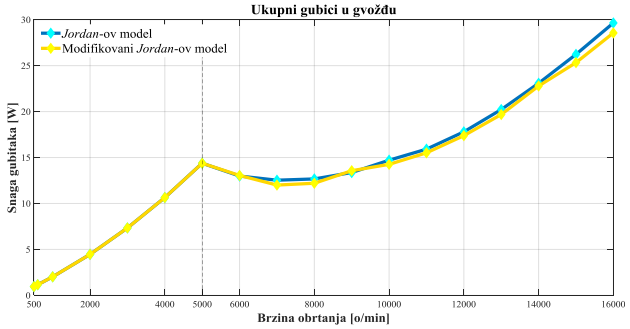
Slika 5 *Gubici usled vrtložnih struja u statoru: Uporedni prikaz Jordan-ovog i modifikovanog Jordan-ovog modela*

Na slici 6 su prikazani gubici usled vrtložnih struja u magnetnom kolu rotora u zavisnosti od brzine obrtanja, primenom *Jordan*-ovog i modifikovanog *Jordan*-ovog modela proračuna. Primetan je značajan porast ovih gubitaka sa porastom brzine, što je i očekivano, u istom smislu kao što je to bio slučaj sa ovim gubicima u magnetnom kolu statora, uz napomenu o tri puta većoj frekvenciji varijacije magnetnog polja u površinskom sloju rotora u odnosu na frekvenciju u statoru.



Slika 6 *Gubici usled vrtložnih struja u rotoru: Uporedni prikaz Jordan-ovog i modifikovanog Jordan-ovog modela*

Na slici 7 su uporedno prikazani gubici u gvožđu za različite brzine obrtanja motora, primenom *Jordan*-ovog i modifikovanog *Jordan*-ovog modela proračuna. Primetno je da dva modela proračuna vrlo malo odstupaju sa aspekta ukupnih gubitaka u magnetnom kolu motora, ali da lokalno daju drugačije rezultate, što je očigledno sa prethodno prikazanih slika. Za motore koji rade u širokom opsegu brzina je važna i lokalna raspodela gubitaka, zbog pravilnog dimenzionisanja sistema hlađenja, tako da se ispostavlja da je od značaja lokalna distribucija gubitaka, koja je vernije prikazana primenom modifikovanog *Jordan*-ovog modela.



Slika 7. *Ukupni gubici u gvožđu: Uporedni prikaz Jordan-ovog i modifikovanog Jordan-ovog modela*

4. ZAKLJUČAK

U radu su prikazani postupci proračuna gubitaka u magnetnom kolu električnih mašina, koji se zasnivaju na *Jordan*-ovim modelima. Ovi modeli su primenjeni u izračunavanju lokalne raspodele snage gubitaka u sinhronoj mašini sa utisnutim stalnim magnetima na rotoru. Pošto je reč o mašini koja tokom rada pokriva širok opseg brzina, veoma je važno poznavati raspodelu gubitaka, koja značajno varira sa promenom brzine obrtanja motora. Priloženi modeli proračuna gubitaka su iskorišćeni u lokalnom obliku, primenom rezultata dobijenih metodom konačnih elemenata. Metoda konačnih elemenata daje izuzetno detaljnu magnetnu sliku motora, nudeći mogućnost rekonstruisanja talasnih oblika vektora magnetne indukcije u svakom od pojedinačnih konačnih elemenata. Na osnovu poznavanja lokalnih vrednosti vektora magnetne indukcije, ali i njegovih talasnih oblika se dobijaju i lokalne raspodele snage gubitaka.

Uporedno su prikazani gubici (histerezisni i usled vrtložnih struja) u magnetnom kolu statora i rotora, kao i ukupni gubici u magnetnom kolu celog preseka motora. Pokazuje se da ima vrlo malih odstupanja u ukupnim gubicima, ali da se pojedinačne komponente razlikuju u zavisnosti od primenjenog modela proračuna. Ovim se potvrđuje da je izuzetno važno poznavati lokalnu distribuciju gubitaka, jer od njih direktno zavisi i lokalna temperatura magnetnog kola. Ukoliko dolazi do lokalnog pregrevanja, o tome se mora voditi računa prilikom projektovanja rashladnog sistema. Primetan je značajan porast gubitaka u rotoru sa porastom brzine obrtanja, a budući da su ovi gubici skoncentrisani u vrlo tankom površinskom sloju magnetnog kola rotora, može se zaključiti da je ovaj deo motora potencijalno najviše termički ugrožen.

5. LITERATURA

[1] C. Steinmetz, „On the law of hysteresis," American Institute of Electrical Engineers., vol. 9, pp. 3-64, 1892. Reprinted: Proceedings of the IEEE, vol. 72, no. 2, 1984

[2] Reinert, J., Brockmeyer, A., & De Doncker, R. W. A. A. (2001). Calculation of losses in ferro- and ferrimagnetic materials based on the modified Steinmetz equation.

[3] H. Jordan, Die ferromagnetischen Konstanten für schwache Wechselfelder, Elektr. Nach. Techn., vol. 1, p. 8, 1924

[4] R. H. Pry and C. P. Bean, Calculation of the energy loss in magnetic sheet materials using a domain model, Journal of Applied Physics, vol. 29, no. 3, pp. 532533, Mar. 1958.

[5] J. Li, T. Abdallah, and C. Sullivan, Improved calculation of core loss with nonsinusoidal waveforms, in Industry Applications Conference, 2001. Thirty-Sixth IAS Annual Meeting. Conference Records, vol. 4, 2001, pp. 22032210

[6] F. Fiorillo and A. Novikov, "An improved approach to power losses in magnetic laminations under nonsinusoidal induction waveform," IEEE Trans. Magn., vol. 26, no. 5, 1990.

[7] E. Barbisio, F. Fiorillo, and C. Ragusa, "Predicting loss in magnetic steels under arbitrary induction waveform and with minor hysteresis loops," IEEE Trans. Magn., vol. 40, no. 4, 2004.

[8] www.femm.info/wiki/HomePage

Kratka biografija:

Milica Banović rođena je u Novom Sadu, 1996. godine. Diplomirala je na Fakultetu tehničkih nauka u Novom Sadu 2019. godine. Master rad iz oblasti Elektrotehnike i računarstva – Energetska elektronika i električne mašine odbranila je 2020. godine.
kontakt: banovicmilicaa@gmail.com

Dejan Jerkan je docent na Fakultetu tehničkih nauka u Novom Sadu, na Katedri za Energetsku elektroniku i pretvarače. Oblast interesovanja su mu modelovanje i dijagnostika električnih mašina, kao i metoda konačnih elemenata.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/10BE39Trninic

SISTEM ZA OBUHVAT I OBRADU PODATAKA IZ HETEROGENIH IZVORA PODATAKA I NJIHOVO SKLADIŠTENJE U JEZERU PODATAKA

SYSTEM FOR ACQUISITION AND PROCESSING OF DATA FROM HETEROGENEOUS DATA SOURCES AND ITS PERSISTENCE IN A DATA LAKE

Milorad Trninić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu predstavljen je sistem za obuhvat i obradu podataka iz heterogenih izvora. Projektovanje pomenutog sistema motivisala je potreba za velikim skupom podataka u cilju treniranja modela mašinskog učenja čiji je kvalitet direktno proporcinalan raznolikosti i količini dostupnih podataka. Sistem omogućava proširivost i skalabilnost komponenti za obuhvat i obradu kako bi zadovoljio zahtev rada sa velikim skupom podataka različite strukture. Svi obuhvaćeni podaci se trajno pohranjuju u jezero podataka u neizmenjenom obliku. Procesi obrade podataka transformišu obuhvaćene podatake u skladu sa potrebama klijenta. Implementirani sistem je dokaz koncepta za obuhvat, trajnu pohranu i obradu velikog skupa podataka sa ciljem pripreme podataka za treniranje modela mašinskog učenja.*

Ključne reči: *Veliki skupovi podataka, distribuirani informacioni sistemi, ETL.*

Abstract – *In this paper, a system for acquisition and processing of data from heterogeneous data sources is presented. Design of this system is motivated by the use of the big data sets in training of machine learning models. Quality of the trained models is directly proportional with data volume and variety. System supports extensibility and scalability of the components in order to meet the needs of processing big data sets which have various structures. All of the acquired data is persisted in a data lake with an unaltered structure. Data processing transforms acquired data to suit the client's needs. System implemented in this paper is a proof of concept for acquisition, persistence and processing of big data sets with the goal of preparing the data for training of machine learning models.*

Keywords: *Big Data, distributed information systems, ETL.*

1. UVOD

Sa razvojem platformi koje nude usluge iznajmljivanja servera i padom cena tih usluga kao i razvojem alata otvorenog koda, obrada velike količine podataka postaje dostupna sve većem broju kompanija. Sve više se daje na značaju analizi podataka o poslovanju i razvija se disciplina nazvana nauka o podacima (engl. *data science*). Ova disciplina podrazumeva rad sa velikim skupovima podataka i oslanja se na definisane protočne strukture. Protočne strukture predstavljaju niz akcija primenjenih nad obuhvaćenim podacima, gde je izlaz jedne akcije ulaz u narednu akciju i imaju za cilj pripremu podataka za analizu podataka. Priprema podataka i postavljanje protočnih struktura naziva se inženjering nad podacima (engl. *data engineering*). Količina podataka koja se generiše povećava se svakim danom, a tradicionalni sistemi za obradu podataka ne mogu da prate taj trend. Glavni razlog za to jeste što tradicionalni sistemi nisu projektovani tako da inherentno podržavaju horizontalno skaliranje. Horizontalna skalabilnost podrazumeva sposobnost komponenti da uvećaju svoju efikasnost sa povećanjem opterećenja sistema dodavanjem novih računara u klaster.

Osnovni cilj ovog rada jeste formiranje horizontalno skalabilne i proširive platforme za obuhvat i obradu velike količine podataka koja će omogućiti dostupnost podataka na jednom mestu. Takođe, važno je obezbediti slabu spregnutost između komponenti (engl. *loosely coupled components*) za obuhvat i obradu podataka. U slučaju jakih veza između komponenti obuhvata i obrade podataka javlja se problem implementacione zavisnosti komponenti.

2. STANJE U OBLASTI

Vodeći klaud (engl. *cloud*) provajderi poput Amazon Veb Servisa (engl. *Amazon Web Service, AWS)*, Majkrosoft Azur (engl. *Microsoft Azure)* i Gugl Klaud Platforme (engl. *Google Cloud Platform, GCP)* poseduju svoje predloge rešenja za obuhvat i obradu podataka uz trajno skladištenje na njihovim implementacijima jezera podataka. Jezera podataka predstavljaju trajna skladišta sirovih podataka. Svako od preloženih rešenja za obuhvat i obradu podataka se sastoji od komponenti za obuhvat, obradu i skladištenje podataka. Razlike predloženih rešenja su posledica razlika između korišćenih komponenti, ali se konceptualno ne razlikuju značajno. Prednosti upotrebe navedenih predloga rešenja za obuhvat i obradu podataka su brojne, polazeći od plaćanja po upotrebi resursa, automatskog skaliranja pa do održavanja klastera od strane provajdera. Problem sa upotrebom rešenja predloženih od klaud provajdera nastaje već kad nekoj od komponenti treba izmeniti implementaciju. Takođe postoji problem i privatnosti podataka, jer mnoge kompanije ne žele da iznose svoje podatke na klaud

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vladimir Dimitrieski.

okruženje. Pored navedenih problema, rešenja klaud provajdera ograničavaju portabilnost sistema. Naime, proces obuhvata i obrade podataka postaje implementaciono zavisan od softverskih rešenja odabranog provajdera.

Pored vodećih klaud provajdera postoji i nekoliko kompanija koje nude softverska rešenja za obuhvat i obradu podataka kao servis (engl. *Software as-a-Service, SaaS*), jedna od njih je Apsolver (engl. *Upsolver)*. Konceptualna razlika Apsolvera i rešenja klaud provajdera potiče od činjenice da je Apsolver gotov servis bez mogućnosti izmene implementacije obuhvata i obrade podataka [1]. Rešenja klaud provajdera nude platformu sa komponentama za obradu i obuhvat podataka, a korisnik je zadužen za implementaciju procesa obuhvata i obrade. Ograničenje dostupnih komponenti zaduženih za proces obrade može negativno da utiče na ceo proces obuhvata i obrade podataka iz razloga što postoje specijalizovana softverska rešenja za određene slučajeve upotrebe koja znatno efikasnije rade od univerzalnog softverskog rešenja.

Arhitektura opisana u ovom radu je projektovana sa ciljem da podrži proširivost što omogućava izmenu, dodavanje ili uklanjanje tehnologija koje implementiraju određene komponente arhitekture. Projektovani sistem takođe omogućava potpunu kontrolu nad konfiguracijom komponenti što omogućava optimizaciju za platformu na kojoj se sistem nalazi. Migracija na neki drugi klaud provajder sa prethodno navedenih platformi i servisa nije moguća ili zahteva značajne izmene u implementaciji, dok se u projektovan sistemu migracija svodi na transfer podataka na novo okruženje. Iz navedenih razloga projektovana je opisana arhitektura za potrebe kompanije u okviru čijeg projekta je i implementiran praktični deo ovog rada.

3. ARHITEKTURA SISTEMA ZA OBUHVAT I OBRADU PODATAKA

Projektovana arhitektura sistema za obuhvat i obradu podataka iz heterogenih izvora podataka može se podeliti na četiri jasno definisana modula. To su: (1) modul za obuhvat podataka, (2) modul za obradu podataka, (3) modul za nadzor logova sistema i (4) modul za komunikaciju s klijentom.

3.1. Modul za obuhvat podataka

Modul za obuhvat podataka preuzima podatke sa izvora podataka i prosleđuju ih modulu za obradu podataka. Ovaj modul sastoji se od dve komponente: komponenta zadužena za obuhvat podataka iz izvora i komponenta zadužena za razmenu poruka.

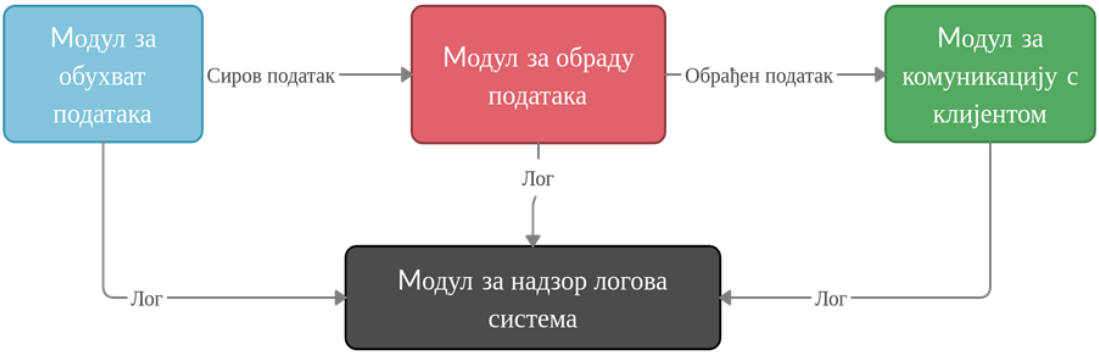
Komponenta zadužena za obuhvat podataka iz izvora ima za zadatak da sakupi podatke iz izvora definisanih od strane klijenta koji koriste obrađene podatke iz ovog sistema. Način obuhvata podataka zavisi od samog izvora podataka, to mogu biti programi koji preuzimaju podatke sa programskog interfejsa aplikacija (engl. *application programming interface, API*), podaci sa internet svega (engl. *internet of things, IoT*) uređaja itd. Najvažniji zahtev koji ova komponenta treba da ispuni je mogućnost ostvarivanja komunikacije sa komponentom za razmenu poruka.

Komponenta zadužena za razmenu poruka ima za cilj prenošenje podataka sa sistema za razmenu poruka u modul za obradu podataka. Postoje dva konceptualno različita načina da se prenos podataka izvrši: direktno i posrednikom tj. brokerom. Direktna komunikacija povećava složenost komponenti koje ostvaruju komunikaciju, jer svaka od komponenti mora biti svesna interfejsa druge komponente sa kojom razmenjuje poruke. Pored dodatne složenosti javlja se i jaka sprega odnosno implementaciona zavisnost između komponenti. Komunikacija putem brokera uvodi pojam teme (engl. *topic*). Tema u ovom vidu komunikacije predstavlja skladište poruka. Na jednu temu mogu se objavljivati poruke ili se na temu može pretplatiti, odnosno čitati poruke sa nje. Navedeni koncept poznat je kao objavi-pretplati (engl. *publish-subscribe*). Upotrebom brokera omogućava slabu spregnutost između komponenti.

3.2. Modul zadužen za obradu podataka

Modul zadužen za obradu podataka sastoje se iz tri komponente: komponenta zadužena za preuzimanje poruka, trajno skaladište podataka - jezero podataka i komponenta zadužena za obradu.

Komponenta zadužena za preuzimanje poruka predstavlja pretplatnika u sistemu razmene poruka. Za svaki od izvora podataka postoji jedna instanca ove komponente koja čita poruke sa komponente za razmenu poruka odnosno odgovarajuće teme. Cilj ove komponente jeste upis sirovih podataka u jezero podataka. Obrada u ovoj komponenti je svedena na dodavanje metapodataka.
Najvažnija osobina ove komponente jeste efikasno čitanje velikog skupa podataka.



Slika 1. *Moduli arhitekute*

Trajno skaladište podataka - jezero podataka u ovoj arhitekturi čini objedinjeno skladište svih podataka obuhvaćenih sa izvora podataka, na koje se naslanjaju odgovarajuće komponente za obradu podataka. U jezero podataka se smeštaju sirovi podaci u njihovom originalnom obliku uz dodatak metapodataka kao što su vreme obuhvata podataka, ime izvora i verzija izvora. Podaci se u trajno skladište upisuju inkrementalno, što znači da se tokom upisivanja novih podataka stari podaci nikada ne brišu niti menjaju.

Komponenta zadužena za obradu podataka u ovoj arhitekturi ima za cilj da predstavi obuhvaćene podatke na način koji klijentu odgovara. Obrada podataka u ovoj arhitekturi je paketna. Obrađeni podaci mogu predstavljati pogled nad sirovim podacima u jezeru podataka ili izvedenu informaciju iz podataka. Neke od najvažnijih osobina ove komponente su dobre performanse paketne obrade podataka i dostupnost brojnih konektora za čitanje i pisanje podataka. Pored procesa obrade ova komponenta zadužena je za slanje obrađenih podataka u modul za komunikaciju sa klijentom.

3.3. Modul za komunikaciju s klijentom

Klijent u ovom sistemu predstavlja svaku eksternu aplikaciju ili korisnika koji je u komunikaciji sa sistemom. Ovaj modul postoji da ne bi dolazilo do remećenja procesa izvršavanja, odnosno da klijent ne može da menja podatke u bilo kojoj komponenti sistema i time potencijalno promeni ishod drugih procesa obrade podataka. Postoji i aspekt skrivanja implementacionih detalja procesa obrade od klijenta iz razloga što klijent ne bi trebalo da vodi računa o načinu implementacije već samo o rezultatima obrade podataka. Kako su klijentski podaci izlazni podaci paketne obrade, komunikacija se obavlja skladištenjem podataka u bazu podataka kojima klijenti mogu da pristupe. Odabrana baza podataka treba da ima sledeće osobine: mogućnost skladištenja velike količine podataka i mogućnost skladištenja polustrukturiranih i nestrukturiranih podataka.

3.4. Modul za nadzor logova sistema

Kako je u pitanju platforma za obradu velikog skupa podataka može se zaključiti da skoro sve komponente imaju više od jednog procesa, odnosno funkcionišu u distribuiranom režimu. Kako zahtevi za resursima rastu, raste i broj procesa pa i čvorova klastera. Ovaj skup komponenti ima za cilj da na jednom mestu okupi sve logove sistema, omogući njihovu pretragu i pregled u korisničkom interfejsu. Najvažnije osobine ovih komponenti su brza pretraga logova i pregledan korisnički interfejs.

4. IZBOR TEHNOLOGIJA ZA IMPLEMENTACIJU KOMPONENTI SISTEMA

Tehnologija koja će predstavljati komponentu za razmenu podataka u ovom sistemu je Apači Kafka (engl. *Apache Kafka*). Kafka je distribuirani sistem za razmenu poruka koji se zasniva na objavi-pretplati načinu razmene poruka uz posredstvo najmanje tri brokera [2]. Brokeri sadrže teme koje se skladište na disku. Teme su sačinjene od skupa poruka. Skup poruka je podeljen u particije koje se repliciraju i time omogućavaju otpornost na otkaze [2]. Kafka može da skalira dodavanjem brokera.

Tehnologija odabrana za komponentu za preuzimanje poruka je Apači Spark (engl. *Apache Spark*). Spark je radni okvir koji omogućava distribuiranu obradu podataka. Ono što je prednost Spark-a u odnosu na druge radne okvire za obradu podataka jeste brzina obrade [3]. Veća brzina obrade velike količine podataka u Spark radnom okviru je omogućena izvršavanjem u memoriji, odloženom evaluacijom (engl. *lazy evaluation*) i optimizacijama plana izvršavanja (engl. *execution plan*). Glavni razlog za odabir Spark radnog okvira za preuzimanje poruka sa sistema za razmenu poruka jeste to što ima brojne konektore za čitanje i pisanje podatka.

Jezero podataka u ovom sistemu predstavlja trajno skladište svih podataka koji su obuhvaćeni sa izvora podataka. Tehnologija odabrana za ovu komponentu je *HDFS*. *HDFS* je distribuirani sistem datoteka koji je deo Hadup radnog okvira za distribuiranu obradu podataka. Blok podataka je najmanja jedinica manipulacije u *HDFS* sistemu datoteka [4]. Replikaciom blokova postiže se tolerancija otkaza [4]. *HDFS* skalira dodavanjem čvorova sa podacima. Razlog za upotrebu *HDFS*-a za jezero podatak jeste mogućnost skaliranja i činjenica da je *HDFS* softver otvorenog koda [4].

Podaci iz jezera podataka se obrađuju paketnom obradom i smeštaju u klijentska skladišta. Tehnologija odabrana za ovaj zadatak je Spark, konkretno njegov *SQL* modul. Ovaj modul na Spark dodaje mogućnost pisanja naredbi u jeziku za strukturirane upite (engl. *structured query language, SQL*) kao i tipizaciju objekata kojima se manipuliše [3]. Spark je odabran za obradu podataka zbog superiornije brzine u odnosu na njemu slične radne okvire poput Hadup MapRidjus (engl. *Hadoop MapReduce)* [3]. Pored navedenih osobina Spark aplikacije imaju dobru interoperabilnost sa drugim tehnologijama za skladištenje podataka [3].

Nakon obrade podaci se smeštaju u klijentska skladišta. Kako je obrada paketna i kako su u pitanju velike količine podataka mogućnosti koje ova komponente treba da ima su: horizontalno skaliranje, efikasno čitanje velike količine podataka i mogućnost rada se polustrukturiranim i nestrukturiranim podacima. Tehnologija koja zadovoljava ove kriterijume je MongoDB [5]. MongoDB je najkorišćenija distribuirana nerelaciona (engl. *NoSQL)* baza podataka [5]. MongoDB kao i sve do sada navedene tehnologije podržava horizontalno skaliranje izvršavanjem procesa horizontalnog particionisanja (engl. *sharding*) [5].

Pošto svaka od navedenih tehnologija radi u distribuiranom režimu očekuje se i veliki broj čvorova u sistemu. Problem sa ovim jeste detekcija loga nekog od procesa bilo koje komponente sistema. Za komponentu koja vrši nadzor sistema odabran je Elastik Stek (engl. *Elastic Stack).* Sastoji se od tri softverska rešenja kompanije Elastik *(engl. Elastic),* u pitanju su Elastiksrč *(engl. Elasticsearch)*, Logsteš *(engl. Logstash)* i Kibana (engl. *Kibana*). Elastiksrč predstavlja skladište teksta loga sa velikom brzinom pretrage. Logsteš je procesor formata logova. Kibana u ovom sistemu služi kao korisnički interfejs za pretragu logova.

5. IMPLEMENTACIJA

Sistem je projektovan za obuhvat i obradu podataka iz heterogenih izvora podataka. Za dokaz koncepta odabrani izvor je VebKroler (engl. *WebCrawler) aplikacija.* VebKroler ima za cilj skupljanje podatak sa veb sajtova čiji su osnovni sadržaj vesti iz oblasti finansija. Napisana je u programskom jeziku Pajton i koristi Selenijum, radni okvir za otvaranje stranica i navigaciju kroz stablo elemenata veb stranice. Razvijene su dve celine *VebKroler* aplikacije. Prva celina obuhvata hiperlinkove od vesti na određenim putanjama. Druga celina obuhvata podatke poput naslova i tela članka, glavne slike, datum objave itd. sa svakog obuhvaćenog hiperlinka. Ove dve celine komuniciraju putem Kafke.

Nakon obuhvata podataka druga celina šalje podatke u vidu poruke na Kafku, temu koja se zove isto kao i domen sa kojeg su podaci obuhvaćeni. Sa ove teme poruke se preuzimaju od strane aplikacije nazvane KrolerKonzjumer (engl. *CrawlerConsumer)* napisane pomoću Spark radnog okvira. Na preuzete poruke dodaju se metapodaci poput vremena obuhvata i imena teme. Obogaćeni podaci čuvaju se na *HDFS*-u u Parke (engl. *Parquet)* formatu.

Podaci sa *HDFS*-a se čitaju u aplikaciji *KarensiPredikt* (engl. *CurrencyPredict)* koja predstavlja implementaciju komponente za obradu podataka. Aplikacija *Karensi-Predikt* ima za cilj da na osnovu teksta vesti odredi koja se valuta spominje u tekstu i kakav je sentiment. Određivanje valuta i sentimenta je svedeno na problem klasifikacije. Za svaku valutu postoji jedan binarni klasifikator koji određuje da li se ta valuta pominje ili ne pominje u tekstu. Korišćeni binarni klasifikator je logistička regresija [6]. Pošto tekst vesti nema strukturu, korišćene su tehnike obrade slobodnog teksta (engl. *Natural Language Processing NLP)* za pretprocesiranje i ekstrakciju odlika (engl. *features*). Neke od pomenutih tehnika su: ukljanjanje stop reči, tokenizacija, svođenje reči na osnovni oblik (engl. *stemming*) i drugo [7].

Nakon određivanja pomenutih valuta u tekstu i sentimenta teksta članka rezultati se smeštaju u odgovarajuću kolekciju MongoDB baze podataka.

6. ZAKLJUČAK

Kompanije neprestano pokušavaju da pronađu najbolje rešenje za obuhvat i obradu podataka koje se uklapa u njihove zahteve po pitanju arhitekture, tehnologija i sigurnosti podataka. Postoje skupovi gotovih alata koji se mogu koristiti u cilju obrade i obuhvata podataka, ali pored svojih prednosti uvode i razna ograničenja. Ograničenja gotovih sistema jesu mogućnost odabira tehnologija. migracija sistema na druge platforme i privatnost podataka. Upotreba tehnologija za koju kompanija ima raspoloživ inženjerski kadar je vrlo važna pri planiranju projekata, što znači da bi kompanija trebalo da zaposli nove ljude koji poznaju nametnute tehnologije ili da postojeći kadar uči nove tehnologije.

Sistem opisan u ovom radu je zamišljen da pomogne pri izvlačenju korisnih informacija iz obuhvaćenih podataka. Izvedene informacije postaju nešto što doprinosi vrednosti kompanije, samim tim se javlja i zahtev za zaštitom izvedenih informacija. Iz navedenih razloga javila se potreba za projektovanjem i implementacijom sistema za obuhvat i obradu podataka iz heterogenih izvora podataka kao i njihovo trajno skladištenje. Skladištenjem podataka u ovakvom sistemu vremenom bi se stvorio veliki skup podataka od interesa za kompaniju koji bi doprineli kvalitetu izvedenih informacija iz tih podataka po ceni skladišta i resursa obrade.

Upotrebom ovog sistema, u kojem su jasno razdvojeni procesi za obuhvat i obradu, mogu se lakše i podeliti zadaci na međusobno nezavisne razvojne timove. Iz svega navedenog zaključuje se da je glavni doprinos ovog rada proširiva i skalabilna arhitektua koja omogućava razdvajanje obuhvata i obrade velikog skupa podataka kao i trajno čuvanje sirovih podataka iz heterogenih izvora podataka.

U okviru daljeg razvoja sistema biće omogućen ispravan rad na više od jednog servera, čime bi se u praksi podržala obrada velikog skupa podataka. Zatim je u planu omogućavanje da komponente za obradu podataka skaliraju po zahtevu posla koji se obavlja kao i izbegavanje usporenja procesa obrade porastom skupa podataka implementacijom inkrementalne obrade.

7. LITERATURA

[1] Yoni Iny, "Upsolver - Technical Whitepaper: The Modern Data Lake Architecture", 2019

[2] https://kafka.apache.org/documentation/ (pristupljeno u avgustu 2020.)

[3] https://spark.apache.org/ (pristupljeno u julu 2020.)

[4] Tom White, *Hadoop: The Definitive Guide, Fourth Edition*, O'Reilly Media, Inc., 2009

[5] Kristina Chodorow, Michael Dirolf, *MongoDB: The Definitive Guide*, O'Reilly Media, Inc., 2015

[6] https://spark.apache.org/docs/latest/ml-guide.html (pristupljeno u avgustu 2020.)

[7] https://en.wikipedia.org/wiki/Natural_language_processing#Common_NLP_Tasks (pristupljeno u septembru 2020.)

Kratka biografija:

Milorad Trninić rođen je u Novom Sadu 1995. godine. Dilpomirao je na Fakultetu tehničkih nauka 2018. godine iz oblasti Elektrotehnike i računarstva – Računarstvo i automatika sa prosečnom ocenom 9,12. Od maja 2020. godine radi kao inženjer podataka.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.4:621.395
DOI: https://doi.org/10.24867/10BE40Jovanovic

PRIMENA ALGORITAMA MAŠINSKOG UČENJA U OFTAMOLOŠKOJ ANALIZI SLIKA OKA KOD PREVREMENO ROĐENIH BEBA

MACHINE LEARNING ALGORITHMS IN OFTALMOLOGY OF PREMATURELY BORN BABIES

Jovana Jovanović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – SOFTVERSKO INŽENJERSTVO I INFORMACIONE TEHNOLOGIJE

Kratak sadržaj – *Ovaj rad daje uvid u algoritme mašinskog učenja koji se mogu koristiti za detekciju Retinopatije kod prevremeno rođenih beba. Za rešavanje problema upotrebljene su konvolutivne neuronske mreže. Izvršena je analiza dobijenih rezultata upotrebom ResNet50, Xception i InceptionV3 modela. Na osnovu izvršenih analiza se došlo do zaključka da skup podatka koji je korišćen nije dobar i da treba da se popravi.*

Ključne reči: *Retinopatija kod prevremeno rođenih beba, konvolutivne neuronske mreže*

Abstract – *This paper provides insight into machine learning algorithms that can be used to detect Retinopathy of prematurity. Convolutional neural networks were used to solve the problem. The analysis of the obtained results was performed using ResNet50, Xception and InceptionV3 models. Based on the performed analyzes, it was concluded that the data set used was not good enogh and needed to be corrected.*

Keywords: *Retinopthy of permaturity, Convolutional neural network*

1. UVOD

Retinopatija kod novorođenčadi (ROP) je očno oboljenje koje se javlja kod prevremeno rođene dece i u najtežem obliku može da dovede do potpunog gubitka vida. Jedna je od vodećih uzroka gubitka vida kod dece. Svake godine 50000 beba u svetu izgubi vid zbog ove bolesti [1]. Nema simptoma kada se prvi put razvije. Može se otkriti samo tokom pregleda kod oftamologa. Rano otkrivanje ove bolesti je od velikog značaja.

Rad se bavi problemom detekcije ROP-a. Izabrano je da se problem reši upotrebom veštačke inteligencije (*eng. Artificial intelligence)*, koja je se sve više koristi u medicini. Međutim, iako se ove metode sve više koriste, još uvek postoji strah da su metode zasnovane na AI sklone greškama i osetljive na greške u podacima. Generalno, nisu tako dobre kao medicinski stručnjaci [2]. Ipak, one se smatraju budućnošću kliničke dijagnostike u oblasti oftamologije i medicine uopšte [3].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dragan Dinu, docent.

Na osnovu prethodnih radova na ovu temu [4, 5, 6], izabran je podskup metoda mašinskog učenja koje su dale najbolje rezultate. Odabrane su konvolutivne neuronske mreže za koje se smatra da su trenutno najperspektivnije za problem detekcije. Tri neuronske mreže koje su odabrane su *ResNet50*, *Xception* i *InceptionV3* model.

Cilj ovog rada je da se primenom izabranih modela mašinskog učenja, odredi koji od njih najbolje klasifikuje podatke iz skupa ulaznih oftalmoloških slika.

2. OFTAMOLOGIJA

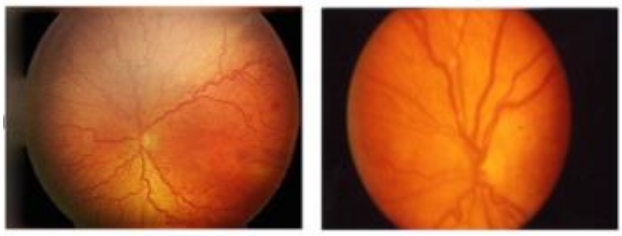
Naziv oftamologija potiče od latinskih reči *Opthalmos*, odnosno oko i *logos*, što u prevodu znači nauka. Takođe, oftamologija je grana medicine koja se bavi proučavanjem ljudskog oka, njegovom anatomijom, funkcijom i bolestima.

Čulo vida je jedno od najbitnijih čula kod čoveka, jer nam daje informacije o spoljašnjem svetu i omogućava nam kvalitetniji život. Uzroci gubitka vida razlikuju se u odnosno na to koliko je neka država razvijena.

2.1 Retinopatija kod dece

ROP (*Slika 1*) je identifikovan kao glavni uzrok slepila kod dece. Kod prevremeno rođenih beba krvni sudovi mrežnjače se ne razvijaju u potpunosti.

ROP je oboljenje koje se odlikuje abnormalnim rastom krvnih sudovi, koji mogu da izrastu iz mrežnjače i prouzrokuju stvaranje ožiljnog tkiva, što dovodi do odvajanje mrežnjače. Ukoliko se ne nadgleda i ne leči u tačno vreme, može prouzrokovati gubitak vida [7].



Slika 1. ROP

3. METODOLOGIJA

U ovom poglavlju je dat kratak opis osnovnih teorijskih pojmova neophodnih za razumevanje korišćenih tehnika.

3.2 Neuronske mreže

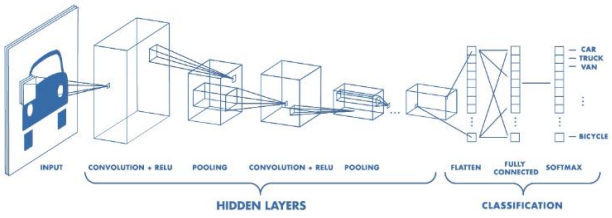
Neuronske mreže su skup algoritama inspirisanih funkcionisanjem bioloških neuronskih mreža mozga, pa se zbog toga nazivaju i veštačkim neuronskim mrežama (*Artificial Neural Network*) [8]. Ova vrsta mreža se sastoji od slojeva u kojima se nalaze neuroni, odnosno čvorovi [9]. Neuroni jednog sloja su povezani sa neuronima iz drugog sloja. Svaki sloj čvorova trenira se na različitom skupu karakteristika, na osnovu rezultata dobijenih u prethodnom sloju. Što se više napreduje sa istraživanjem u oblasti neuronskih mreža, to čvorovi mogu da prepoznaju sve složenije karakteristike, jer objedinjuju i rekombinuju karakteristike iz prethodnog sloja.

3.1 Konvolutivne neuronske mreže

Konvolutivne neuronske mreže su vrsta neuronskih mreža, koje se najčešće primenjuju za analizu vizuelnih slika. Koriste se za detekciju objekata, prepoznavanje lica, klasifikaciju slika i mnoge druge stvari [10].

Konvolutivne neuronske mreže imaju mogućnost hvatanja prostornih i vremenskih zavisnosti na slici primenom filtera i upotrebom operacije konvolucije. Filter je matrica brojeva, koja se prevlači po slici od gornjeg levog ugla ka donjem desnom uglu. Za svaku tačku na slici, izračunava se vrednost na osnovu filtera pomoću operacije konvolucije.

Upotreba filtera i operacije konvolucije smanjuje broj parametara koje neuronske mreža treba da nauči, a omogućava i da se lokacija karakteristika koje se izdvajaju sa slike promeni.



Slika 2. Arhitektura konvolutivne neuronske mreže

Konvolutivne neuronske mreže se sastoje iz dve glavne komponente (*Slika 2*):

1. Komponenta za učenje karakteristika - u ovom delu mreže se izdvajaju bitne karakteristike slike, kao što su ivice, oblici, linije.
2. Klasifikacija - sastoji se od potpuno povezanih slojeva. Služi za dodeljivanje verovatnoća onoga što algoritam predviđa.

4. IMPLEMENTACIJA

U ovom poglavlju dat je osvrt na rešenje postavljenog zadatka, odnosno primena algoritama nad određenim skupom podataka. Kada se rešavaju problemi mašinskog učenja potrebno je odrediti sledeće stvari:

1. Prikupiti ili pronaći skup podataka za problem koji se rešava.
2. Odlučiti se koje algoritme mašinskog učenja koristiti za rešavanje problema.
3. Evaluirati dobijene rezultate.

4.1 Skup podataka

Da bi metode mašinskog učenja bolje „naučile“ željene modele i/ili ponašanje, potrebno je posedovati što veću količinu podataka. Prvi bitan korak koji je potrebno preduzeti jeste formirati skup podataka koji će se koristiti za obučavanje algoritama. Skup podataka se sastoji od parova ulaznih i izlaznih vrednosti, gde će ulazne vrednosti da budu slike, a izlazne klasa kojoj slika pripada. Postoje dve klase, a to su detektovan je ROP i nema ROP-a.

U ovom radu su korišćena dva skupa podataka, nazvaćemo ih skup A i skup B. Skup B predstavlja modifikovani skup A. Sve slike su dobijene pomoću širokookutnog uređaja za snimanje, poznatog kao RetCam. Slike su prikupljene u bolnicama u Srbiji. Rezolucija slika je 640x480 piksela. Skup A se sastoji od 6021 slike, od toga 4448 slika koje pripadaju klase da ima ROP-a i 1573 slike iz suprotne klase. Skup podataka B, kako je već rečeno je malo promenjen i ima 7231 sliku, od toga 4601 sliku sa ROP-om i 2630 slika na kojima nije dijagnostikovana bolest.

Da bi se uspešno obučio i evaluirao model, skupovi podataka su podeljeni na 2 celine, a to su trening skup i test skup, u razmeri 80:20. Trening skup je dalje podeljen i na validacioni u razmeri 25% validacioni skup i 75% trening skup. Za podelu trening skupa korišćena je tehnika stratifikacije, da bi na ravnomeran način u oba skupa bio srazmeran broj instanci obe klase.

Zbog ograničenih resursa u pogledu memorije, podaci su učitavani u paketima (batch-evima) od po 64 slike. Takođe, sve slike su smanjene na veličinu od 240x320 piksela i normalizovane su, vrednosti boja na slici su svedene u raspon od 0 do 1.

4.1 Konvolutivne neuronske mreže

Modeli konvolutivnih neuronskih mreža koje su korišćene u ovom radu su izabrani na osnovu prethodnih radova koji su napisani na ovu temu.

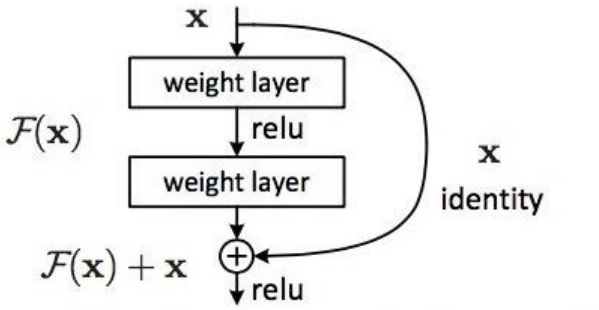
4.1.1 Rezidualne neuronske mreže

Rezidualne neuronske mreže ili skraćeno ResNet su specifična vrsta neuronskih mreža. Predstavljaju veoma duboke neuronske mreže, koje koriste rezidualne konekcije [11].

Rezidualna mreže ima arhitekturu konvolutivne neuronske mreže, koja može da podrži stotine ili više konvolutivnih blokova i blokova identiteta.

Rešavaju problem gradijenta koji nestaje. Neuronske mreže se treniraju putem povratnog širenja, koje se oslanja na gradijentni spust da bi se pronašle optimalne težine koje minimiziraju funkciju gubitka. Kada se doda više slojeva, ponovljeno množenje njihovih derivata na kraju čini gradijent beskrajno malim, što znači da dodatni slojevi neće poboljšati performanse ili ih čak smanjiti.

Svi blokovi u mreži, konvolutivni ili blokovi identiteta su rezidualni blokovi, a primer jednog rezidualnog bloka može se videti na slici *Slika 3*.



Slika 3. Prikaz jednog rezidualnog bloka

Rezidualni blokovi sadrže vezu koja preskače slojeve između, koja je poznata kao veza preskakanja i ona čini jezgro preostalih blokova [11]. Zbog ove veze izlaz iz sloja nije isti.

U radu je korišćen ResNet50 model ovih mreža. On sadrži 50 slojeva.

4.1.2 Xception

Xception je duboka konvolutivna neuronska mreža, koja uključuje dubinski razdvojene konvolucije (eng. Depthwise Separable Convolutions). Razvijena je od strane Google istraživača.

Mreža se sastoji iz tri toka, ulazni, srednji i izlazni. Podaci prvo prolaze kroz ulazni tok, zatim ulaze kroz srednji tok i kroz njega teku osam puta, na kraju prolaze kroz izlazni tok. Svaki sloj konvolucije i odvojene kovolucije (eng. SeparableConvolution layers) praćen je sa normalizacijom [12].

4.1.3 InceptionV3

InceptionV3 je treća verzija konvolutivnih neuronskih mreža razvijenih od strane Google-a. Sastoji se od 11 početnih modula, gde se svaki modul sastoji od slojeva grupisanja i konvolutivnih filtera sa ispravljenim linearnim jedinicama kao funkcijom aktivacije [13].

4.2 Obučavanje

Obučavanje se radi nad instancama trening skupa. Jedna obrada svih instanci se naziva epoha. Obučavanje se sastoji od 4 koraka, a to su:

1. Kreiranje inicijalnog stanja modela neuronske mreže
2. Izračunavanje izlaza za instancu ulaznih podataka
3. Ocenjivanje dobijenih izlaza
4. U ovom koraku sve zavisi od dobijenog izlaza. Ukoliko je izlaz dovoljno tačan, onda je kraj, u suprotnom se prelazi na narednu instancu podataka i ponavlja se korak 2.

Modeli su kreirani upotrebom Keras biblioteke. Ukinut je poslednji sloj i dodati su novi slojevi. Dodat je jedan sloj za grupisanje da bi se smanjio broj parametara, ravan sloj (Flatten layer), da bi podaci bili pogodniji za klasifikaciju. Zatim su stavljena dva potpuno povezana sloja sa ReLu kao funkcijom aktivacije i jos jedan sa softmax funkcijom i oni predstavljaju deo za klasifikaciju.Nakon što se postavi inicijalno stanje modela i dobije rezultat za jednu instancu iz skupa podataka, potrebno je oceniti koliko je rezultat zadovoljavajući. Ova ocena se naziva funkcija gubitka (*eng. loss function*) i direktno utiče na sposobnost modela da ''nauči'', bez obzira na to koji algoritam za ažuriranje stanja se koristi. Zato je bitno adekvatno obrazovati funkciju gubitka. Pošto je problem klasifikacione prirode iskorišćena je Cross entropy funkcija. Nakon odabira funkcije gubitka, potrebno je odabrati odgovarajući algoritam za učenje.

Ovaj deo se zove i optimizacija modela i ima za cilj pronalazak ekstrema funkcije gubitka. Za funkciju optimizacije upotrebljena je funkcija RMSProp, zato što bira različit korak učenja za svaki parametar. Takođe, pored funkcije optimizacije potrebno je odrediti i korak učenja, koji predstavlja konstantu koja određuje brzinu učenja same mreže. Promena koraka učenja rađena je po principu povratnog poziva (eng. Callback), upotrebom metode ReduceLROnPlateau, koja menja brzinu učenja kada se detektuje da nema promena za dati broj epoha u obučavanju.

5. REZULTATI

U ovom poglavlju predstavljeni su rezultati dobijeni primenom izabranih modela konvolutivnih neuronskih mreža nad A i B skupom podataka. U tabeli *Tabela 1* su prikazani rezultati za svaki model pojedinačno. Za merenje performansi modela iskorišćeni su tačnost, F1 mikro mera i funkcija gubitka. Tačnost predstavlja delić predviđanja koje je model uspešno klasifikovao. F1 mikro mera predstavlja koji procenat pozitivnih predviđanja je tačan.

Na osnovu prikazanih podataka može se zaključiti da modeli daju bolje performanse za skup podatka A, što znači da povećanjem instanci podataka u skupu B, modeli nisu uspeli da povećaju svoju efikasnost, kao što je bilo očekivano.

Nad skupom podataka A, ResNet50 model ima najbolju tačnost i F1 meru, ali i najveću funkciju gubitka, pa je zbog toga model sa najboljim performansama za skup A, InceptionV3 model.

Nad skupom B najbolje se pokazao Xception model. Kada se obuhvate rezultati za oba skupa, model sa najboljim performansama je Xception,

Tabela 1. Prikaz dobijenih rezultata (vrednosti su izražene u %)

	Skup A			Skup B		
	ResNet50	Xception	InceptionV3	ResNet50	Xception	InceptionV3
Tačnost	76	74.9	75	69	72	70
F1 mikro mera	76	74.9	75	69	72	70
Funkcija gubitka	54	53.7	53.6	60	57	59

6. ZAKLJUČAK

U ovom radu koriste se metode mašinskog učenja za detekciju ROP bolesti nad dva skupa podataka. Za rešavanje problema, upotrebljena su tri modela kovolutivnih neuronskih mreža, ResNet50, Xception i InceptionV3.

Tabela 2. Klasifikacioni izveštaj dobijen primenom Xception modela nad skupom A (vrednosti izražene u %)

	Preciznost	Odziv	F1 mera	Broj instanci u skupu
0	0.53	0.42	0.47	314
1	0.81	0.87	0.84	889
Tačnost			0.75	1203

Dobijeni rezultati nastali evaluacijom modela nad testnim skupovima prikazani su u delu o rezultatima. Xception model se najbolje pokazao od svih modela, ali kada se detaljno pregleda izveštaj klasifikacije prikazan u tabelama *Tabela 2* i *Tabela* 3, na osnovu odziva može se zaključiti da jako loše klasifikuje slike koje nemaju ROP.

Pokušano je da se problem reši dodavanjem novih slika koje nemaju ROP u skup podataka B, ali kako je prikazano u tabeli*Tabela 3* problem nije rešen. Pa se na osnovu toga zaključuje da je potrebno ispraviti skup podataka.

Takođe, skupovi podataka su labelirani od strane jednog oftamologa, što nije dobro, jer postoji samo jedno mišljenje. Za buduće radove sa prikazanim skupovima podataka, potrebno je prvo srediti podatke.

Tabela 3. Klasifikacioni izveštaj dobijen primenom Xception modela nad skupom B (vrednosti izražene u %)

	Preciznost	Odziv	F1 mera	Broj instanci u skupu
0	0.78	0.34	0.47	526
1	0.71	0.95	0.81	920
Tačnost			0.71	1446

7. LITERATURA

[1] O. D. Saugstad, „Oxygen and retinopathy of prematurity,“ Journal of Perinatology , pp. S46-S50, 2006.

[2] M. A. Zarbin, „Artificial Intelligence: Quo Vadis?,“ Translational Vision Science & Technology, t. 9, pp. 1-1, 2020.

[3] J. A. H. a. D. T. Azar, „The AI Revolution and How to Prepare for It,“ Translational Vision Science & Technology, t. 9, pp. 16-16, 2020.

[4] R. J. Y. C. L. Z. J. H. Y. W. W. D. J. Z. Z. Y. Jianyong Wang, „Automated retinopathy of prematurity screening using deep neural networks,“ EBioMedicine, 2018.

[5] Y. C. J. Z. R. J. Z. Y. F. I. Junjie Hu, „Automated Analysis for Retinopathy of Prematurity by Deep Neural Networks,“ Transations on Medical Imaging, 2019.

[6] W. L. Q.-q. D. C. C. a. Y. S. Yan Tong, „Automated identification of retinopathy of prematurity by image-based deep learning,“ Eye and Vision, 2020.

[7] N. S. A. O. D. S. A. S. J. K. Sanja Knežević, „Analysis of Risk Factors in the Development of Retinopathy of Permaturity,“ 2011.

[8] L. Hardesty, „Neural Networks,“ 4 April 2017. [Na mreži]. Available: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. [Poslednji pristup 10 September 2020].

[9] C. Nicholson, „A Beginner's Guide to Neural Networks and Deep Learning,“ Pathmand, [Na mreži]. Available: https://wiki.pathmind.com/neural-network#element. [Poslednji pristup 10 September 2020].

[10] M. Stewart, „Simple Introduction to Convolutional Neural Networks,“ Towrds Data Science, 17 Feb 2019. [Na mreži]. Available: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac. [Poslednji pristup 10 September 2020].

[11] H. Mujtaba, „Introduction to Resnet or Residual Network,“ 9 Sep 2020. [Na mreži]. Available: https://www.mygreatlearning.com/blog/resnet/. [Poslednji pristup 16 September 2020].

[12] F. Chollet, „Xception: Deep Learning with Depthwise Separable Convolutions,“ Computer Vision and Pattern Recognition, 2016.

[13] S.-H. Tsang, „Review: Inception-v3 — 1st Runner Up (Image Classification) in ILSVRC 2015,“ 10 Sep 2018. [Na mreži]. Available: https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c. [Poslednji pristup 16 September 2020].

Kratka biografija:

Jovana Jovanović rođena je u Zvorniku 1996. god. Master rad na Fakultetu tehničkih nauka iz oblasti Softversko inženjerstvo i informacione tehnologije, odbranila je 2020. godine.
kontakt: jovana.jovanovic@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK : 004.3
DOI: https://doi.org/10.24867/10BE41Degenek

ПОРЕЂЕЊЕ ANGULAR, REACT.JS И VUE.JS JAVASCRIPT ПРОГРАМСКИХ ОКВИРА НА ПРИМЕРУ УПРАВЉАЊА СТАЊИМА КЛИЈЕНТСКИХ АПЛИКАЦИЈА

COMPARISON OF ANGULAR, REACT.JS AND VUE.JS JAVASCRIPT FRAMEWORKS ON THE EXAMPLE OF CLIENT APPLICATION STATE MANAGEMENT

Арсеније Дегенек, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У овом раду дат је опис, као и поређење шаблона за ефикасно управљање стањима клијентских апликација у три тренутно најпопуларнија JavaScript програмска оквира. На основу детаљне анализе и примене библиотека специфичних за сваки програмски оквир, изведени су закључци о томе колико је сваки од њих погодан за коришћење. Сврха читаве анализе јесте доћи до одговора којом стратегијом и уз помоћ ког програмског оквира је најлакше структурирати клијентску апликацију, тако да она остане погодна за одржавање упркос својој комплексности. У циљу ефикасног поређења, употребом сваког програмског оквира креирана је по једна прототипска апликација. Све оне су базиране на истом скупу функционалности, осмишљених тако да дају реалну слику о предностима и манама примењеног шаблона.*

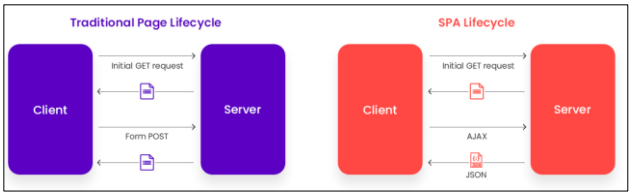
Кључне речи: *Програмски оквир, библиотека, управљање стањима*

Abstract – *This paper provides a description, as well as a comparison of patterns for efficient state management of client applications in three currently most popular JavaScript frameworks. Based on a detailed analysis and implementation of framework-specific libraries, conclusions were drawn about how suitable each of them is for use. The main goal of the whole analysis is to decide which strategy and framework are the most convenient to structure the client application, so that it remains easy for maintenance, despite its complexity. Using each framework, one prototype application was created for the purpose of more efficient comparison. All of them are based on the same set of functionalities, designed to give the real picture of the advantages and disadvantages of the applied pattern.*

Keywords: *Framework, library, state management*

1. УВОД

Као последица велике популарности веб апликација последњих година, захтеви за модерним и интерактивним корисничким интерфејсом су порасли.
То је довело до замирања традиционалног приступа развоја коришћењем вишестраничних апликација (енг. ***Multi-Page Applications***). Уместо њих, појавиле су се једностраничне апликације (енг. ***Single Page Applications***). Оне су омогућиле раздвајање клијентске и серверске стране, због чега је серверски део постао истовремено употребљив и за апликације које се покрећу на платформама мобилних уређаја. Такође, једностраничне апликације симулирају природно понашање претраживача, што значи да страница неће бити поновно учитана приликом добављања новог садржаја са сервера. Разлика између ова два приступа дата је на слици 1.



Слика 1. Вишестраничне и једностраничне апликације

Ослањајући се на овај концепт, дошло је до појаве различитих програмских оквира базираних на JavaScript језику. Они представљају основу са предефинисаним функционалностима, коју је могуће искористити за бржи и квалитетнији развој једностраничних апликација. Сви овде анализирани програмски оквири су засновани на компонентама, као основним градивним јединицама које заједно формирају изглед и понашање странице приказане у претраживачу.

1.1. Мотивација

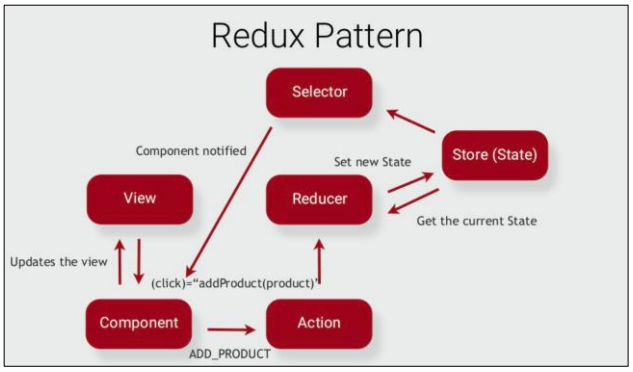
Употреба програмских оквира знатно је олакшала развој комплексних веб апликација. Ствари као што су рутирање, интеракција са DOM-ом (енг. ***Document Object Model***), комуникација са сервером, употреба форми и слично знатно су упрошћене и добро документоване. Међутим, проблем се јавља у ситуацијама које није било лако предвидети приликом пројектовања програмског оквира. У сва три наведена програмска оквира, компоненте међусобно комуницирају на два могућа начина:

- Child-To-Parent комуникација.
- Cross-Component комуникација.

У првом случају, две компоненте не могу да комуницирају директно, без да њихова комуникација иде преко посредника, који је у овом случају родитељска компонента. Ово за последицу има веома

НАПОМЕНА:
Овај рад је проистекао из мастер рада чији ментор је био др Драган Дину, доцент.

велико уплитање зависности, које постаје готово немогуће контролисати, јер је зарад комуникације између две компоненте потребно увести трећу. У другом случају, компоненте комуницирају преко дељене магистрале. Комуникација је базирана на емитовању и прихватању догађаја (енг. ***Events***). Ипак, магистрала временом постаје оптерећена, непрегледна и неконзистентна са становишта података, јер компоненте могу несметано да јој приступају.
Решење за овај проблем лежи у употреби Redux шаблона. Настао с циљем олакшавања комуникације између компоненти, овај шаблон уводи нове појмове који су објашњени у наставку. Идеја је да на нивоу апликације у сваком тренутку постоји само један извор истине у формату JSON објекта, односно једно стање (енг. ***State***). Ново стање се креира од нуле при свакој промени било којег његовог дела и оно је садржано у складишту (енг. ***Store***). Такође, једини начин да се оно промени јесте отпремањем акције (енг. ***Action***). Акције су функције које имају интеракцију са редукторима (енг. ***Reducer***) и омогућавају извршавање асинхроног кода. Редуктори су једине методе које директно могу да мењају стање. На крају, круг се затвара тако што компоненте бивају информисане о промени стања преко селектора (енг. ***Selector***). На слици 2 приказана је структура Redux шаблона.



Слика 2. Ток података у Redux шаблону

Дакле, у наставку ће бити анализиране и поређене имплементације овог шаблона у Angular, React.js и Vue.js програмским оквирима.

2. ПРЕГЛЕД СТАЊА У ОБЛАСТИ

Сходно популарности коју JavaScript програмски оквири имају последњих година, постоји велики број истраживања како појединачног програмског оквира, тако и њихових међусобних односа. Међутим, ниједно истраживање везано за њихово поређење не даје акценат на област управљања стањима.
У оквиру рада [1] дато је детаљно поређење ова три програмска оквира по популарности, тежини савладавања и перформансама, али без осврта на управљање стањима приликом дефинисања наведених критеријума.
Рад [2] се такође бави поређењем поменутих програмских оквира, али као предмет поређења узима уграђени сет функционалности, подршку и величину заједнице која стоји иза сваког од њих.
Треће истраживање [3] даје врло детаљан осврт на сва три програмска оквира, али улази у детаље интеракције са DOM-ом, а то није од велике користи за потребе овог истраживања.
Што се тиче самог Redux шаблона, стручна литература постоји и даје веома прецизно објашњење како он функционише и у којим случајевима га треба користити [4]. Оно што свакако фали, а често се доводи у питање јесте који програмски оквир изабрати, уколико је донета одлука да се за управљање стањем клијентске апликације користи Redux шаблон.

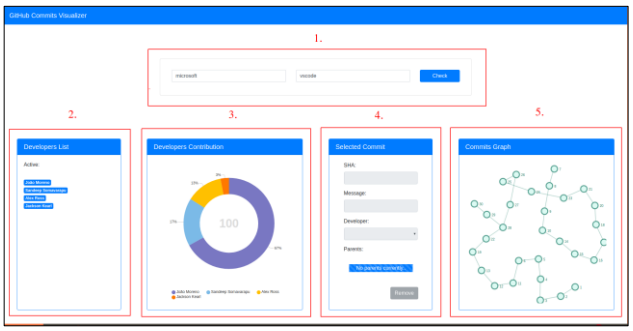
3. МЕТОДОЛОГИЈА

С обзиром да је предмет истраживања поређење Redux-а и његова анализа у различитим срединама, најтачнији и најверодостојнији закључци изведени су из директне имплементације самог шаблона у сва три поменута програмска оквира. За развој React.js и Vue.js апликација коришћен је Javascript језик, односно његова ES6 спецификација. У случају Angular-а, изворни код писан је у TypeScript-у.

3.1. Софтверска имплементација

Приликом спецификирања захтева прототипског софтвера, акценат је био на креирању апликације која ће у позадини моћи да имплементира све концепте које обухвата Redux, а да има примену и да са визуелног аспекта буде погодна за коришћење. Из тог разлога, апликација садржи дијаграме и графове, чија стања се чувају на централизованом месту (Store) и сведоче о исправној функцији примењеног шаблона. Апликација је функционално једнака за све програмске оквире, али постоје минималне разлике у корисничком интерфејсу јер сваки од њих подржава различите пакете за визуелизацију података. Скуп подржаних функционалности (слика 3):

- Преузимање последњих тридесет промена (енг. ***Commit***) са подразумеване гране жељеног GitHub репозиторијума.
- Приказ свих програмера који су починили промене и филтрирање промена по програмеру који их је направио.
- Приказ ангажовања сваког програмера на дијаграму (Pie Chart).
- Приказ почињених промена у структури графа.
- Модификација одређене промене и њено брисање.



Слика 3. Апликација и компоненте

Библиотеке које имплементирају Redux у сваком од програмских оквира:

- ngrx (Angular),
- redux (React.js),
- vuex (Vue.js).

3.2. Интеграција са Redux библиотекама

С обзиром на то да Redux не долази уграђен ни у један програмски оквир, један од критеријума поређења био је његова интеграција са већ постојећом апликацијом која користи неки од традиционалних приступа за управљање стањима. У сва три случаја, језгро шаблона интегрише се као пакет преко npm-a (енг. ***Node Package Manager***). Оно што се разликује јесте постављање Store-a тако да му буде могуће програмски приступити. Такође, додатна својства шаблона као што су нпр. комуникација са сервером или ослушкивање промена у неким програмским оквирима су подржана као део основног пакета, док се у осталим морају накнадно инсталирати.

3.3. Нормализација података

Препорука је да се Redux шаблон користи искључиво у комплексним клијентским апликацијама које изискују велику међусобну зависност између компоненти и раде над великом количином података. Такве апликације у 99% случајева комуницирају са сервером и од њега добијају податке. С обзиром да серверске апликације већином раде над релационим подацима, те релације пренеће се и на клијентску страну, сваки пут када подаци стигну. Сходно томе, иако није имплементирана приликом инсталације основног пакета Redux-a, нормализација података се сматра неизоставним делом у његовој употреби. Њена намена је да нам обезбеди такву структуру Store-a да нам његово коришћење на дуге стазе не донесе више штете него користи.
Под овим се подразумева реорганизација JSON објекта тако да је олакшана претрага и повлачење података и спречавање њиховог дуплирања (сва референцирања иду преко јединственог идентификатора). У случају свих програмских оквира коришћена је ***normalizr*** библиотека. Она захтева креирање шема на основу којих се структура Store-a постави приликом првобитног уписа података. Библиотека је базирана на ES6 спецификацији, али постоји њен подскуп који се користи за TypeScript.

На основу наведеног се закључује да је ова библиотека од изузетне важности, али се различито укапа у зависности од језика којим је имплементирана логика апликације. Због тога, интеграција са њом и једноставност коришћења API-ја које она нуди, коришћени су као други критеријум за поређење ових програмских оквира.

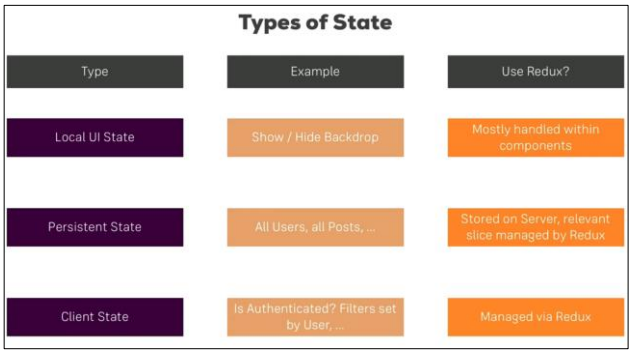
3.4. Модуларизација Store-a

Још један битан критеријум представља могућност поделе Store-a на модуле. Ово је важно из два разлога. Први је што Store постане изузетно непрегледан и тежак за одржавање при великој количини података, упркос томе што су они нормализовани. У том случају, има смисла поделити га на мање делове тј. модуле, тако да сваки од њих садржи податке који припадају истој логичкој целини. Са становишта шаблона, овде се ништа не мења. Сви модули на крају ће бити спојени у један велики Store, као и до сада. Ово је само структурално побољшање, које може бити од велике користи.
Други разлог заснива се на чињеници да готово никада неће бити потребно нормализовати све податке који се налазе у Store-у. Ствари као што су флегови, подаци о ауторизацији и слично неће бити нормализовани, те нема потребе мешати их са остатком података. Због тога нам подршка модуларизације омогућава да поред логичких целина, апликационо стање раздвојимо и на основу тога који његови делови су нормализовани, а који не.

3.5. Остали критеријуми поређења

Поред издвојених критеријума, коришћено је још мањих, који су од значаја приликом избора програмског оквира за подржавање Redux шаблона. Читљивост и прегледност изворног кода након интеграције са Redux-ом свакако је први следећи критеријум вредан разматрања. Треба нагласити да оно што се овде пореди није читљивост кода пре и после употребе шаблона. Сама чињеница да је шаблон настао, већ нам говори да је изворни код знатно побољшан. Овде се мисли на међусобно поређење читљивости код ова три програмска оквира, након што сваки од њих имплементира шаблон, над истим сетом функционалности.
Друга битна ствар је процес дебаговања Store-a. Ово је од изузетне важности за брзо и ефикасно отклањање проблема. У сва три случаја, алати за дебаговање се морају додатно инсталирати у претраживач и увезати са апликацијом. Међутим, они нису исти за све програмске оквире, што даје додатни простор за поређење.
На крају овог поглавља, треба додати да поред увођења Redux шаблона у апликацију, постоје њени делови који неће бити садржани у оквиру глобалног стања. Овде пре свега спадају флегови који су везани искључиво за изглед саме компоненте и нису од значају другим компонентама. Њихово измештање у Store не доноси никакве бенефите, него напротив уводи компликације у цео процес, те се сматра лошом праксом. На слици 4 приказано је који типови

података иду у глобално стање, а који остају везани за компоненту.



Слика 4. Употреба Redux-а у зависности од типа података

4. ЗАКЉУЧАК

Након имплементације шаблона који омогућују лакше и поузданије управљање стањима клијентских апликација и његовог поређења кроз три JavaScript програмска оквира, изведени су закључци везани за одабир најпогоднијег од њих. На почетку, одмах, треба нагласити да не постоји методологија која ће по свим критеријумима прогласити један програмски оквир као најбољи избор. У том случају, не би постојала остала два и не би имала толико широку употребу.

Са становишта интеграције са Redux библиотеком, највише посла је било при изради апликације у React.js програмском оквиру. Разлог за ово је што Redux библиотека није уско везана за програмски оквир као друге две, те је поред њеног укључивања било потребно укључити и библиотеку за њено конектовање са програмским оквиром. Такође, за рад са селекторима потребно је укључити додатну библиотеку, али то је последица оптимизације која не постоји у остала два програмска оквира.

Када је реч о нормализацији података, за очекивати је било да ће тај процес бити много лакши у React.js и Vue.js програмским оквирима, јер је библиотека базирана на ES6 спецификацији. Упркос томе што постоји њена подршка за TypeScript, она у тренутку израде Angular апликације није била довољно развијена, те није подржавала креирање сложенијих акција над нормализованим Store-ом. Из овог разлога, библиотека је враћена на верзију коју користе друга два програмска оквира, али као последица тога шеме и функције за нормализовање нису креиране у духу TypeScripta.

Процес поделе Store-а на модуле могућ је у сва три програмска оквира. Ипак, најједноставније га је било имплементирати у Vue.js-у, јер у њему не постоји концепт редуктора, који је као последица модуларизације трпео промене у остала два програмска оквира.

Програмерски алати за дебаговање су исти у случају Angular i React.js програмских оквира. Прегледни су и једноставни за употребу. Vue.js има свој, засебан алат, али упркос томе што није компатибилан са осталим програмским оквирима, нуди готово све исте функционалности.

На крају, треба навести главне адуте сваког од наведених програмских оквира. Angular је дефинитивно прави избор за изградњу стабилне апликације отпорне на велику количину багова, захваљујући статичкој типизацији коју омогућава TypeScript. Ово се наравно одражава и на Store, јер се он такође писан у овом језику.

Са друге стране, redux библиотека коју користи React.js је дефинитивно најразвијенија од све три, првенствено због чињенице да је овај шаблон првобитно настао за потребе овог програмског оквира. То за последицу има дужи процес савладавања библиотеке, али и њено много ефикасније коришћење јер су функционалности поприлично оптимизоване у позадини.

Vue.js је настао као програмски оквир који је покупио најбоље од претходна два и покушао то да поједностави. То је приметно и у случају управљања стањима. Његовом употребом сам процес развоја ће ићи брзо. Коришћење vuex библиотеке је упрошћено и што је најважније, притом нису изгубљене функционалности.

ЛИТЕРАТУРА

[1] Elar Saks *"JavScript frameworks: Angular vs React vs Vue"*. [Преузето: 20.09.2020]

[2] Eric Wohlgethan *"Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React adnd Vue.js"*. [Преузето: 20.09.2020]

[3] Mattias Levlin *"DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue and Svelte"*. [Преузето: 20.09.2020]

[4] Nir Kaufman *"Thinking in Redux"*. [Преузето: 20.09.2020]

Кратка биографија:

Арсеније Дегенек рођен је у Бачкој Тополи 1995. године. Мастер рад на факултету техничких наука из области Електротехнике и рачунарства – Рачунарство и аутоматика одбранио је 2020. год.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE42Perlinac

ANALIZA PROGRAMSKOG JEZIKA PYTHON

PYTHON PROGRAMMING LANGUAGE ANALYSIS

Tamara Perlinac, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *U prvom delu rada ukratko su objašnjeni osnovni pojmovi o programskim jezicima uopšteno i napisane neke opšte stvari o Python-u, a zatim je izvršena sintaksna analiza programskog jezika Python, i opisani su osnovni elementi i koncepti ovog programskog jezika. U drugom delu rada implementirana je studija slučaja, u okviru koje je obađen primer rada sa fajlovima, otvaranja fajla i isčitavanja podataka iz njega, a zatim smeštanja isčitanih podataka u kolekcije programa i kasniju manipulaciju podacima i pristup istim. Studija slučaja implementirana je u programskim jezicima Python i JAVA, analizirane su obe implementacije i povučena paralela između njih.*

Ključne reči: *programski jezik, Python, JAVA, sintaksa, manipulacija podacima, kolekcije, datoteke*

Abstract – *The first part of this paper briefly explains basic terms of programming languages in general and writes some general things about Python, then performs a syntatic analysis of the Python programming language and describes the basic elements and concepts of this programming language. The second part of this paper is case study implementation , which contains an example of working with files, opening a file and reading data from it, then placing the read data in program collections and subsequent data manipulation and access. This case study was implemented in programming languages Python and JAVA, both implementations are analyzed and compared with each other.*

Keywords: *programming language, Python, JAVA, syntax, data manipulation, collections, files*

1. UVOD

Kada se govori o programskim jezicima, kao jedna od definicija može se izdvojiti ta da programski jezik predstavlja formalni jezik, koji sadrži set instrukcija koje proizvode različite vrste izlaza, i koji se koriste za pisanje programa pomoću kojih se vrši implementacija raznih algoritama. Opis programskog jezika deli se u dve osnovne komponente: sintaksu (formu) i semantiku (značenje). Postoje razne podele programskih jezika, prema različitim kriterijumima – programski jezici niskog i visokog nivoa (prema stepenu zavisnosti od računara), interpretirani i kompajlirani (prema načinu izvršavanja), jezici opšte namene i specifični za domen (prema opsegu korišćenja). Programski jezik Python predstavlja interpretirani jezik visokog nivoa i opšte namene, osmišljen krajem 80-ih godina XX veka, čija je implementacija započeta 1989. godine u Holandiji, od strane Gvida van Rosuma. Python omogućava razvrstavanje programa u module, koji mogu biti ponovo iskorišćeni u drugim Python programima.

On sadrži veliku kolekciju standardnih modula, koji se mogu koristiti kao osnova novih programa, ili kao primeri za početak učenja programskog jezika Python. Neki od standardnih modula omogućavaju čitanje/pisanje datoteka, sistemske pozive, kreiranje GUI-ja, vezu ka relacionim bazama podataka, itd. Python je proširiv, što znači da je moguće dodati novu funkciju ili modul, koje je potrebno napisati u nekom drugom programskom jeziku (obično je to programski jezik C). Podržava više paradigmi programiranja – u prvom redu imperativnu, objetno-orijentisanu i funkcionalnu paradigmu, i može biti iskorišćen u različite svrhe. Tema prvog dela ovog rada je sintaksna analiza programskog jezika Python – obrada ugrađenih i ostalih tipova podataka, sintakse komentara, kontrole toka programa (petlje), definisanje funkcija, klasa, obrada grešaka i izuzetaka, unosa i ispisa podataka, čitanja i pisanja datoteka i rad sa kolekcijama. U drugom delu rada implementirana je studija slučaja, koja za cilj ima da prikaže upotrebu ovih obrađenih elemenata na primeru, u programskim jezicima Python i JAVA, i upoređivanje dveju implementacija.

2. SINTAKSNA ANALIZA JEZIKA

2.1. Tipovi podataka

Python poseduje određeni broj ugrađenih tipova podataka, koji se mogu podeliti u sledeće kategorije: tekstualni tipovi (str), numerički tipovi (int, float, complex), tipovi sekvence (list, tuple, range), tip mapiranja (dict), tipovi skupa (set, frozenset), boolean tip (bool), binarni tipovi (bytes, bytearray, memoryview). Tip podatka promenljive u Python-u ne mora biti eksplicitno naveden, ali može. Prilikom dodeljivanja vrednosti promenljivoj, Python joj sam dodeli i tip – u slučaju x = „apple“ Python će promenljivoj x dodeliti tip str, dok u jednakosti x = 6, promenljiva x dobija tip int.

Prilikom inicijalizacije promenljive x može se i eksplicitno navesti njen tip, i u tom slučaju navedene jednakosti bi izgledale ovako: x = str(„apple“) i x = int(6). Python dozvoljava i da se promenljivoj koja je inicijalizovana kao promenljiva jednog tipa, kasnije dodeli vrednost drugog tipa, i to dovodi i do same promene tipa podatka koji nosi promenljiva (tipa promenljive). Tip podatka svakog objekta može se preuzeti pozivom type() funkcije.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio doc. dr Petar Marić.

2.2. Komentari

Komentar je razumljivo objašnjenje ili napomena koje se dodaje izvornom kodu programa radi lakšeg razumevanja od strane osobe koja čita programski kod. Programeri takođe koriste komentare kao napomene sebi, obično da opišu logiku pri pisanju nekih komplikovanih funkcija i sl., kako bi se lakše snašli i podsetili posle nekog vremena, kada dođe na red održavanje programa. Oni mogu biti jednolinijski i višelinijski (blok) komentari, i deo programa koji predstavlja komentar ignorisan je od strane interpretera. Jednolinijski komentar u Python-u obeležava se sa # (tarabom), dok višelinijski komentari tehnički ne postoje u ovom programskom jeziku. Postoje docstring-ovi – string koji služi za dokumentaciju Python-a, i on predstavlja literal string koji se obično koristi u klasama, modulima, funkcijama ili definicijama metoda, koji im se pridružuju kao tehnička dokumentacija, i olakšavaju razumevanje funkcionalnosti svake veće celine koda. Početak i kraj docstring-a obeležavaju se sa tri dvostruka navodnika “””, i smešteni su u __doc__ atributu klase, funkcije, modula ili metode za koju su vezani.

2.3. Kontrola toka programa

Upravljanje tokom programa predstavlja redosled po kome se naredbe, instrukcije i pozivi funkcija izvršavaju. U programskom jeziku Python nema blokova, koji se u većini ostalih programskih jezika koriste za grupisanje iskaza, već se grupisanje vrši uvlačenjem redova – svi iskazi koji se nalaze u istom bloku treba da imaju isti količinu belog prostora od početka reda. Izjave pomoću kojih se može vršiti kontrola toka programa u Python-u su sledeće: IF...ELIF...ELSE – definišu kontrolnu strukturu, koja može sadržati više blokova izjava od kojih će se izvršiti maksimalno jedna i to ona koja zadovoljava uslov naveden nakon neke od reči IF ili ELIF, ili nakon reči ELSE ukoliko nijedan uslov nije zadovoljen, FOR petlja – služi za iteriranje kroz neku sekvencu, i pomoću nje može se izvršiti jedna ili više izjava, jednom za svaku stavku sekvence kroz koju se iterira ovom petljom, range() funkcija – generiše aritmetičku progresiju i olakšava iteriranje sekvencom brojeva, while petlja – uslovna petlja koja se izvršava sve dok je uslov koji sledi nakon ključne reči while zadovoljen. Za kontrolu toka programa koriste se i break izjava – služi za prekidanje daljih iteracija for ili while petlje u okviru koje se napiše i zaustavlja izvršavanje petlje pre nego što se završi iteracija kroz sve stavke, continue izjava – prekida samo trenutnu iteraciju kroz petlju, i nastavlja iteriranje sa sledećom iteracijom, pass izjava – koristi se da bi sintaksa jezika bila zadovoljena, na mestima gde program ne zahteva nikakvu akciju i else klauzula na petljama – označava deo programskog koda koji će se izvršiti kada se petlja završi.

2.4. Definisanje funkcija

Funkcija predstavlja blok organizovanog koda za višekratnu upotrebu, koji se koristi za izvršavanje jedne akcije. Pomoću njih se programski kod može podeliti u organizovane blokove i učiniti bolje organizovanim i ponovo iskoristivim. Što se tiče programskog jezika Python, ključnom rečju def nagoveštava se da će uslediti definicija funkcije. Nakon nje, sledi naziv funkcije, a posle njega u okviru malih zagrada lista parametara funkcije. Linija definicije funkcije završava se dvotačkom, i od naredne linije kreću izjave koje sačinjavaju telo te funkcije (moraju biti pravilno uvučene). Izvršenje funkcije uvodi novu tablicu simbola koja se koristi za lokalne promenljive te funkcije. Tačnije, svaka dodela vrednosti promenljivoj u okviru funkcije čuva se kao nova vrednost u njenoj lokalnoj tablici simbola, i kad god se koristi neka promenljiva, prvo se pretražuje da li postoji u lokalnoj tablici simbola funkcije u okviru koje se zahteva njena vrednost, pa onda dalje. Definicija funkcije uvodi naziv funkcije u tabelu simbola programa. Return izjava služi za vraćanje vrednosti iz funkcije.

Funkcije koje nemaju povratnu vrednost u Python-u vraćaju None – ugrađeno ime za povratnu vrednost funkcije kada nema return izjave. Postoje različite vrste parametara funkcije: obavezni (pozicioni), parametri sa podrazumevanom vrednošću, parametri ključnih reči. Pri pozivu funkcije, neophodno je proslediti sve obavezne parametre, dok se parametri koji imaju podrazumevanu vrednost mogu, ali i ne moraju proslediti. Lambda izrazi su male, anonimne funkcije čija je sintaksa: lambda argumenti: izraz.

Mogu imati proizvoljan broj parametara, a izraz može biti samo jedan. Obično se koriste za manje, jednostavnije delove programskog koda, kako bi uprostile i skratile kod.

2.5. Klase

Klasa predstavlja proširivi šablon programskog koda koji služi za kreiranje objekata, podešavanje početnih vrednosti za stanje klase (atribute) i implementaciju ponašanja (funkcije ili metode). Klase predstavljaju sredstvo za objedinjavanje podataka i funkcionalnosti. Svaka instanca klase (objekat) može imati pridružene atribute klase, čije vrednosti određuju trenutno stanje instance, i metode definisane nad klasom, koje se pozivaju nad instancom klase i uglavnom definišu ponašanje koje vrši promenu stanja instance nad kojom su pozvane. Definicija klase počinje ključnom rečju class, zatim sledi naziv klase. U okviru definicije klase pišu se izjave koje najčešće predstavljaju definicije funkcija (ali nije obavezno da budu). Funkcije koje se definišu u okviru klase su njeni članovi i nazivaju se metodama klase – funkcije koje pripadaju nekoj klasi i može im se pristupiti samo preko objekta te klase. Kada se definiše funkcija u okviru klase (metod), kao prvi parametar se prosleđuje self – ključna reč koja označava samu instancu klase (nad kojom je metod pozvan), i pomoću nje pristupa se svim atributima i drugim metodama klase. Konstruktor klase jeste funkcija __init__ koja služi za kreiranje objekta i inicijalizaciju početnih vrednosti atributa. Python podržava i mehanizme nasleđivanja i višestrukog nasleđivanja – prilikom definicije klase potrebno je nakon naziva klase u malim zagradama proslediti jednu ili više klasa koje su roditeljske toj klasi. Sve klase nasleđuju Object klasu.

2.6. Greške i izuzeci

Postoje najmanje dve vrste grešaka: sintaksne greške – greške koje nastaju pri kompajliranju programa usled nepoštovanja definisane sintakse jezika (pogrešnog redosleda sekvence znakova ili tokena) i prekidaju izvršenje programa i izuzeci – greške koje nastaju u toku izvršenja programa i nisu bezuslovno fatalne greške –

njima se može rukovati, pošto su oni predvidivi. Nekima od izuzetaka je već rukovano od strane programa, ali većinom nije pa je potrebno da programer sam obradi izuzetak i rukuje njime na željeni način. Postoje različiti tipovi izuzetaka, a kao deo poruke o grešci uvek je ispisano o kom tipu izuzetka se radi. Izuzecima se u Python-u rukuje pomoću try-except klauzule, gde su u okviru try bloka piše deo koda u kom može doći do nekog tipa izuzetka, a nakon ključne reči except navode se tipovi izuzetaka kojima se rukuje, a zatim i način na koji se to može uraditi. Moguće je napisati i korisnički definisane izuzetke – potrebno je naslediti klasu Exception.

2.7. Input i output

Podaci koji predstavljaju izlaz programa mogu biti ispisani u ljudski-čitljivoj formi ili upisani u neki fajl, radi trajnog čuvanja istih. Funkcija za ispis podataka jeste funkcija print(), dok se za upis podataka u fajlove koristi funkcija write(). Formatiranje ispisa može se vršiti pomoću formatiranih string literala, format() metode ili ručno. Računarska datoteka ili fajl, predstavlja često korišćen resurs za trajno skladištenje podataka. Python pruža ugrađene metode koje rade sa fajlovima – open(), pomoću koje se otvara datoteka, kojoj se pri pozivu proslede dva argumenta: prvi sadrži ime datoteke koja se otvara, dok drugi predstavlja režim otvaranja datoteke (govori da li se datoteka otvara za čitanje/pisanje/oba, ili pak za dodavanje novog sadržaja na kraj datoteke ili eksluzivno kreiranje). Režim nije obavezan argument poziva, i ukoliko se ne prosledi koristi se režim čitanja. Metod open() vraća fajl objekat.

Druge značajne metode za rad sa datotekama su: read() – čitanje sadržaja otvorene datoteke, readline() – čitanje jedne linije iz datoteke, write(string) – upisivanje prosleđenog stringa u datoteku, tell() – vraća trenutnu poziciju u fajl objektu, seek(offset, whence) – vrši promenu trenutne pozicije otvorene datoteke. U Python-u se često koristi ključna reč with u kombinaciji sa pozivom metode open(), i prednost njenog korišćenja jeste u tome što je datoteka ispravno zatvorena automatski od strane programa kada se rad sa datotekom završi, čak i ako u nekom momentu dođe do nekog izuzetka.

2.8. Strukture podataka (kolekcije)

Struktura podataka predstavlja oblik organizacije, upravljanja i skladištenja podataka, koji omogućava efikasan pristup podacima i modifikaciju istih. Najčešće korišćene strukture podataka su liste, dok u Python-u postoje i setovi, rečnici, sekvence itd. Liste predstavljaju uređene kolekcije koje su podložne izmenama i koje mogu sadržati duplikate. U Python-u se prazna lista kreira pomoću uglastih zagrada, a ukoliko je potrebno kreirati listu sa nekim inicijalnim vrednostima, onda se one navode razdvojene zarezom u uglastim zagradama. Elementima liste moguće je pristupiti pomoću indeksa – navođenjem naziva objekta liste i broja u uglastim zagradama koji predstavlja indeks, pristupa se elementu liste koji se nalazi na navedenoj poziciji (indeksiranje ide od nule). Python podržava i negativne indekse – ukoliko je indeks negativan broj, onda se pristupa elementima od kraja liste. Postoji velik broj metoda za rad sa listama, od kojih su najvažnije: append(x) – dodaje element x na kraj liste, insert(i,x) – dodaje element x na poziciju određenu indeksom i, remove(x) – uklanja iz liste element čija je vrednost jednaka x ili uzrokuje izuzetak ValueError (ukoliko element ne postoji u listi), pop([i]) – uklanja element liste koji se nalazi na prosleđenom indeksu ili uklanja poslednji element liste (ukoliko indeks nije prosleđen) i vraća ga, clear() – prazni listu, copy() – kreira kopiju liste, reverse() – preokreće mesta elemenata u listi (element sa poslednjeg mesta prebacuje se na prvo, tj. indeks 0, i tako redom), sort(key, reverse) – koristi se za sortiranje liste u rastućem redosledu, a argumenti određuju po kom ključu se sortira lista i da li rastuće ili opadajuće. Ove metode omogućavaju da se liste po potrebi mogu koristiti i kao LIFO i kao FIFO strukture. Korišćenjem list comprehensions mehanizma omogućeno je kreiranje nove liste u kojoj je svaki element rezultat primene operacija na svaki član druge iterabilne kolekcije. Rečnici su strukture podataka koje su indeksirane pomoću ključeva, koji mogu biti bilo kog nepromenljivog tipa podatka. Rečnik zapravo predstavlja skup ključ:vrednost parova, gde je vrednost ključa jedinstvena u okviru tog rečnika. Najčešće korišćen tip podatka za ključeve u rečnicima su stringovi i brojevi. Prazan rečnik se u Python-u kreira pomoću para praznih vitičastih zagrada, a ukoliko je potrebno ubaciti inicijalne vrednosti u rečnik, onda se one navode razdvojene zarezom kao ključ:vrednost parovi u okviru vitičastih zagrada. Vrednostima iz rečnika pristupa se isto kao u listi, samo što se umesto indeksa šalje vrednost ključa u uglastim zagradama. Izraz recnik[“k1”] = “v1” će u rečnik dodati novi par (“k1”, “v1”) – ukoliko vrednost ključa “k1” ne postoji u rečniku ili će pregaziti postojeću vrednost za ključ “k1”, ukoliko vrednost već postoji, dok se uređeni par iz rečnika briše pomoću ključne reči del – del recnik[“k1”]. Prilikom iteriranja kroz rečnike, ključu i njegovoj vrednosti može se pristupiti zajedno, ako se koristi items() metoda nad rečnik objektom (for (key, value) in recnik.items()).

3. STUDIJA SLUČAJA – POREĐENJE UČITAVANJA PODATAKA IZ CSV DATOTEKE I MANIPULACIJE PODACIMA U PROGRAMSKIM JEZICIMA PYTHON I JAVA

CSV (Comma Separated Values) fajl je obična tekstualna datoteka koja sadrži određene podatke, obično razdvojene zarezima. Ove datoteke se koriste kao skladišta podataka, ali isto tako često se koriste za razmenu podataka između različitih aplikacija. Prva linija CSV fajla je linija zaglavlja, koja sa sobom nosi opis semantika podataka koji se nalaze od druge do poslednje linije fajla. Studija slučaja obuhvata primer pisanja programa, koji kao ulaz ima books.csv fajl – fajl koji sadrži podatke o knjigama: autoru, naslovu, godini izdanja, oceni i ceni knjiga, u Python-u i Javi. Program otvara ovaj fajl, učitava podatke iz njega i smešta ih u kolekcije – rečnik, koji kao ključ ima string vrednost autor (ime i prezime autora), a kao vrednost ima listu knjiga koje je taj autor napisao, a zatim prikazuje i opisuje primere iteriranja kroz kolekcije, pristupa i manipulacije podacima, i rad sa njima nakon smeštanja u kolekcije. Python nudi csv modul za rad sa ovim tipom fajla, i on sa sobom donosi mnoge metode i klase koje olakšavaju rad. Otvaranje fajla u Python-u vrši se pomenutom kombinacijom ključne reči with i open metode, bez prosleđivanja režima (otvara se za čitanje što

je po default-u). Prilikom obrade podataka iz csv fajla potrebno je preskočiti liniju zaglavlja. Zatim se iz csv modula koristi klasa DictReader – kojoj se prosledi fajl objekat koji vraća poziv metode open(), i nazivi polja u .csv fajlu. Ovaj objekat mapira informacije u svakom redu .csv fajla u jednu kolekciju tipa rečnik, po ključevima zadatim kroz nazive polja. Dakle, pomoću DictReader-a se od svakog reda .csv fajla kreira jedan rečnik, čiji su ključevi zadati parametrom, a vrednosti se kupe iz svakog reda posebno, i svi oni se smeštaju u listu objekata tipa rečnik. Od ove liste kreira se glavni objekat tipa rečnik, koji će kao vrednost ključa imati autora, a kao vrednost listu knjiga koje je on napisao, i ovo se radi pomoću itertools.groupby funkcije. Ova funkcija kreira iterator koji vraća uzastopne ključeve i grupe za te ključeve iz iterabilne kolekcije koja joj je prosleđena kao prvi argument poziva, i u ovom slučaju će to biti lista objekata tipa rečnik. Drugi parametar poziva funkcije jeste funkcija koja izračuneva ključnu vrednost za svaki element, i upravo ova funkcija zahteva da se iterabilna kolekcija koja se grupiše pomoću groupby funkcije prethodno sortira na osnovu iste ključne funkcije. Zbog ovoga, pre grupisanja liste objekata tipa rečnik i kreiranja glavnog rečnika od njega, potrebno je prvo sortirati listu objekata tipa rečnik po vrednosti za ključ „autor“. Nakon ovoga, postoje razne varijacije na temu kako se može pristupati i manipulisati podacima iz rečnika – prikazati knjige autora Meše Selimovića, ispisati broj knjiga autora Fjodora Dostojevskog, ispisati knjige autora Ive Andrića sortirane po oceni, ili knjige Ive Andrića sortirane po ceni sa ocenom većom od 8 (razni filteri su mogući i dozvoljeni i razlike su minimalne), kao i maksimalno ocenjenu knjigu autora Čarlsa Bukovskog. Ključ rečnika je ime autora, a kako jedna knjiga predstavlja jedan rečnik, tako je vrednost zapravo lista objekata tipa rečnik (lista knjiga) za tog autora. Listi knjiga autora (vrednosti iz rečnika za zadati ključ) pristupa se pomoću get metode – npr listi knjiga autora Meše Selimovića – recnik_knjiga.get(„Mesa Selimovic“), zatim se for petljom prođe kroz sve njegove knjige (objekte tipa rečnik) i ispišu se sve potrebne vrednosti. Ispis broja knjiga određenog autora vrši se pozivom len funkcije nad dobavljenom listom knjiga tog autora, dok se sortiranja u Python-u vrše pomoću ugrađene sorted() funkcije, kojoj se prosleđuju lista koja se sortira, i ključ za sortiranje (zadat pomoću lambda izraza). Sortiranje je podrazumevano rastuće, ukoliko je potrebno uraditi opadajuće onda se sortirana lista u rastućem redosledu samo prosledi reversed funkciji. Primena filtera vrši se tako što se od postojeće liste kreira nova, korišćenjem list comprehensions mehanizma – iz postojeće liste u novu se prebacuju samo oni elementi koji zadovoljavaju određene uslove tj. filtere. Ispis informacija o knjizi je odvojen u posebnu funkciju i poziva se na više mesta u kodu. Za pronalaženje knjige sa najvećom ocenom koristi se max funkcija, kojoj se prosledi lista iz koje traži najveću vrednost (lista knjiga Čarlsa Bukovskog), i ključ po kome to radi – u ovom primeru je to ocena knjige.

Ceo ovaj program u programskom jeziku JAVA implementira se na sličan način, a najveće razlike su u početku – pri otvaranju .csv fajla i smeštanja podataka u kolekcije, dok se primeri manipulacije podacima i pristupa istim ne razlikuju mnogo od Python-a. Prva razlika je što u Javi nema posebnih reader-a koji direktno mapiraju redove fajla u rečnike, pa je potrebno imati pomoćnu klasu Knjiga, gde će biti smeštane informacije o svakoj knjizi posebno. Otvaranje i čitanje fajla u Javi se vrši pomoću BufferedReader-a i FileReader-a, a popunjavanje glavnog rečnika knjiga se vrši ručno – prvo se napravi lista knjiga za autora, a zatim se u rečnik doda ključ-vrednost par. Koriste se objekti tipa HashMap i ArrayList. Listi knjiga za zadatog autora pristupa se pozivom get() metode nad heš-mapom, broj knjiga za autora se dobija pozivom size() metode nad listom knjiga, za definisanje polja nad kojim se vrši poređenje koriste se Javini Comparator-i, dok se sortiranja, filtriranja i pronalaženja maksimuma rade pomoću ugrađenih Collections.sort funkcije, filter() i max() funkcija nad Stream objektima (potrebno je vršiti prebacivanja iz liste u stream i nazad).

4. ZAKLJUČAK

Programski jezici Python i Java pružaju veliki broj metoda, funkcija i klasa za rad sa datotekama i kolekcijama, i olakšavaju isti. U konkretnom primeru rada sa .csv fajlom, Python pruža više klasa i funkcija koje obavljaju veliku količinu posla umesto programera, i jednostavnije su za upotrebu od Javinih klasa koje obavljaju isti posao. Samim tim, količina koda potrebna za implementaciju ovih primera manja je dosta u Python-u nego u Javi, i lakša za razumevanje. Implementacija u Javi zahteva definisanje pomoćne klase Knjiga, dok u Python-u svaka knjiga predstavlja jedan rečnik (posledica prednosti korišćenja DictReader-a). Analizom i izučavanjem programskog jezika Python dolazi se do zaključka da je ovaj jezik veoma pogodan i nudi veliki broj prednosti i razloga za njegovo učenje i korišćenje. Ovaj rad obuhvata samo mali deo mogućnosti i razlika koje ovaj programski jezik može da ponudi u odnosu na ostale. Python nudi ogroman broj biblioteka, modula i framework-a, podržan je od strane velikog broja OS, sintaksa je veoma čitljiva, a kod čist i dobro struktuiran. Podržava nekoliko programskih paradigmi, a primenu nalazi u raznim oblastima: programiranju web aplikacija, desktop aplikacija, mašinskom učenju, programiranju mrežnih servera koji rade sa ogromnim količinama podataka, razvoju interaktivnih kompjuterskih igrica i animacija, itd.

5. LITERATURA

[1] https:/docs.python.org/3/ - zvanična dokumentacija programskog jezika Python

[2] https:/docs.oracle.com/en/java/ - zvanična dokumentacija programskog jezika Java

[3] https://www.tutorialspoint.com/python

Kratka biografija:

Tamara Perlinac rođena je u Novom Sadu 1995. god. Srednju ekonomsku školu završila je 2014. godine, i iste godine upisala OAS na Fakultetu tehničkih nauka, smer Računarstvo i automatika. Diplomirala 2018. god i upisala MAS na istom smeru. Master rad na tehničkih nauka iz oblasti Računarstva i automatike odbranila je 2020.god.
kontakt: tperlinac@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE43Dokic

UPOTREBA GITHAB AKCIJA ZA AUTOMATIZACIJU IZRADE I OCENJIVANJA STUDENTSKIH ZADATAKA

USING GITHUB ACTIONS TO AUTOMATE THE CREATION AND EXAMINATION OF STUDENT ASSIGNMENTS

Jelena Dokić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *Poslednjih godina, na tržištu se pojavljuju mnogi alati za automatizaciju procesa razvoja softvera. Ovaj rad opisuje jedan od takvih alata - Github akcije, kao i prednosti i mane njegovog korišćenja. Opis je dat kroz primer razvoja dodatnih funkcionalnosti u aplikaciji koja služi za automatizaciju procesa kreiranja i ocenjivanja studentskih zadataka.*

Ključne reči: *Github akcije, automatizacija, izrada i ocenjivanje studentskih zadataka*

Abstract – *In the past few years, a lot od software automation tools have appeared on the market. This paper describes the usage of one of those tools – GitHub Actions, with the advantages and disadvantages of its usage. The descriptionis given through the development of features on the application that is used for the creation and examination of student assignments.*

Keywords: *GitHub actions, automation, creation and examination of student assignments*

1. UVOD

Od samog početka razvoja računarstva, ideja je bila da računari i softverski alati koji se na njima koriste olakšaju ljudske aktivnosti koje zahtevaju mnogo uloženog vremena ili resursa da bi se izvršile jednom ili više puta. Ove aktivnosti su vremenom postajale sve kompleksnije, a alati koji ih rešavaju sve moćniji. GitHub akcije su jedan od takvih alata, koji je nastao pre svega nekoliko meseci.

Zadatak ovog rada je da kroz realan primer razvoja softvera otvorenog koda testira primer upotrebe GitHub akcija i sve prednosti i mane koje ovaj alat donosi. Projekat otvorenog koda koji se koristi u ovom primeru je pjisp-assignment-template.

U pitanju je softverski alat koji se koristi za automatizovano kreiranja zadataka za studente korišćenjem šablona na predmetu Programski jezici i strukture podataka na Fakultetu tehničkih nauka u Novom Sadu.

Proces kreiranja zadatka upotrebom pjisp-assignment-template alata sastoji se iz nekoliko koraka, te postoji mogućnost da se napravi greška.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Petar Marić, docent.

Pomoću GitHub akcija, u malopre pomenuti alat se integrišu drugi alati, koji služe za vršenje provera ispravnosti nekih od koraka i koji dodatno olakšavaju proces kreiranja zadatka. Ovi alati predstavljaju programe koji se mogu instalirati i nezavisno od pjisp-assignment-template alata, a i sami za pripremanje izvršne verzije koriste GitHub akcije.

Ovaj rad opisuje sve prethodno pomenute alate i način njihove integracije u celinu upotrebom GitHub akcija. Na kraju samog rada, izvedeni su zaključci o GitHub akcijama stečeni na osnovu njihovog korišćenja u procesu izrade ovog rada.

2. ALATI KOJI SE KORISTE U PROCESU AUTOMATICIJE IZRADE I OCENJIVANJA STUDENTSKIH ZADATAKA

Projekat o kojem ovaj rad govori sastoji se iz grupe alata koje je bilo potrebno napisati i povezati da bi se dodale nove funkcionalnosti u postojeći projekat - pjisp-assignment-template [1]. Pomenuti alat služi za automatizovano kreiranje studentskih zadataka, nihovo puštanje u učionicama u kojima studenti rade zadatke i, na kraju, pregledanje zadataka. Prvenstveno je namenjen da nastavnom osoblju olakša ceo proces od kreiranja do pregledanja kolokvijuma ili zadataka koje studenti rade.

Zadatak nastavnog osoblja je pre svega da iz repozitorijuma projekta koji je šablonski repozitorijum kreira svoj repozitorijum sa zadatkom. Važno je da svoj repozitorijum nazove u skladu sa predefinisanim načinom imenovanja. Nakon toga se u repozitorijumu kreiraju svi potrebni materijali koji su potrebni za generisanje studentskog zadatka.

Zadatak nastavnog osoblja jeste da pokrene generisanje zadatka za jedan od predefinisanih testova na predmetu Programski jezici i strukture podataka. Jedan od alata o kojem ovaj rad govori - pjisp-template-name dodaje mogućnost automatizovanog kreiranja šablona za zadatak za određeni test na osnovu naziva repozitorijuma koji autor zadatka navede, uz proveru ispravnosti naziva.

Po završetku pisanja zadatka i njegovog testiranja, nastavnik ili saradnik je dužan da obavesti drugog nastavnika o tome, da bi on mogao da proveri ispravnost zadatka i odobri korišćenje zadatka u nastavi, naravno ako je dobar i pogodan za studente. Kao olakšicu, autor zadatka može da pokrene alat za testiranje zadatka i proveri ispravnost.

Projekat o kojem ovaj rad govori uvodi isto to testiranje i nakon objavljivanja nove verzije zadatka na GitHub-u. Kao dodatnu proveru,jedan od alata iz ovog projekta - pjisp-diff proverava da li su izmenjene sve potrebne datoteke, da bi zadatak bio kompletan, odnosno da bi imao korektan tekst zadatka, primer rešenja i testove. Ako svi ovi alati vrate rezultat pozitivnog ishoda, u GitHub repozitorijumu se pojavljuje oznaka da su svi uslovi ispunjeni i da je zadatak spreman za pregledanje. U suprotnom, stoji oznaka da zadatak ne ispunjava sve potrebne uslove.

Naredna poglavlja opisuju alate koji su korišćeni da bi se postigla opisana unapređenja pjisp-assignment-template alata, a poslednje poglavlje opisuje njihovu integraciju.

2.1. pjisp-template-name

Alat pjisp-template-name [2] je aplikacija koja služi za proveru ispravnosti naziva repozitorijuma. Napisana je u obliku konzolne aplikacije otvorenog koda u programskom jeziku Python.

Za imenovanje repozitorijuma koristi se predefinisani šablon, koji izgleda ovako:

pjisp-{SCHOOL_YEAR}-{COURSE_ID}-{TEST_ID}-{GROUP_ID}

Alat pre svega proverava da li je dužina naziva ispravna, odnosno da li naziv ima sve celine odvojene znakom "-". Ako nema, ispisuje se poruka "Repository name length not valid" i program izlazi sa izlaznim kodom 1. U suprotnom, program nastavlja sa izvršavanjem. Nakon provere dužine, proverava se ispravnost ostalih tokena, prema malopre opisanom načinu davanja imena. Ako bilo gde dođe do greške, ispisuje se poruka"Repository name not valid. Error on <token>" i izlazi se sa izlaznim kodom 1. U ovom slučaju, <token> predstavlja jednu od celina odvojenih znakom"-". Ako naziv u potpunosti ispunjava šablon, kao izlazna vrednost vraća se identifikator testa, da bi na osnovu toga mogle da se generišu datoteke potrebne za kreiranje tog testa.

2.2. pjisp-diff

Alat pjisp-diff [3] je konzolna aplikacija u programskom jeziku Python, koja služi kao pomoćni alat nastavnom osoblju koji proverava da li su izmenili sve potrebne datoteke pri kreiranju zadataka za studente.

Pored provere da li su promenjene sve potrebne datoteke, ovaj alat proverava i da li su ostale nepromenjene sve pomoćne datoteke koje nije dozvoljeno menjati. Datoteke koje je potrebno menjati zavise od ulaznog parametra, šablona na osnovu kojeg se kreira zadatak.

Datoteke koje je potrebno uvek menjati su assignment_solution.c i assignment.rst.

Ako je vrednost ulaznog parametra jednaka "T12", potrebno je menjati fixtures/stdio-numbers.yaml datoteku.

Ako je vrednost ulaznog parametra jednaka "T34" ili "SOV", potrebno je menjati file-error-input-not-readable.yaml, file-error-output-not-readable.yaml i file-text.yaml datoteke u fixtures direktorijumu.

Datoteka koju nije dozvoljeno menjati je assignment_notes.rst.

Ako su po završetku programa izmenjene sve potrebne datoteke i nisu izmenjene one koje ne smeju da se menjaju, alat vraća 0 kao vrednost izlaznog koda. U suprotnom, alat vraća vrednost 1 uz jednu ili više poruka u obliku: "Please change the <filename> file." ili "Please do not change the <filename> file.". U ovom slučaju, <filename> predstavlja naziv datoteke koja nije izmenjena, a treba da bude ili je izmenjena, a ne treba da bude menjana.

2.3. poetry-publish

Alati koji su opisani u prethodna 2 poglavlja napravljeni su tako da mogu biti instalirani i korišćeni kao Python paketi. Oba paketa su dostupna na PyPI repozitorijumu Python paketa. Za postavljanje na repozitorijum, korišćen je alat Poetry. Pošto je potrebno objaviti novu verziju paketa na repozitorijumu nakon svake značajne izmene u samom paketu, korišćena je poetry-publish [4] GitHub akcija koja pri svakom novom objavljivanju verzije alata na GitHub-u, kreira i novu verziju Python paketa i postavlja je na PyPI repozitorijum.

Akcija poetry-publish je kreirana od strane autora ovog rada pre nego što se razvila ideja za projektom koji je u ovom radu opisan, ali je znatno olakšala kreiranje malopre pomenutih paketa, tako da je imala veoma važnu ulogu i u ovom projektu. Akcija je projekat otvorenog koda i dostupna je za preuzimanje u GitHub prodavnici [5]. Dostupna je pod BSD 3-Clause licencom.

Poetry-publish je GitHub akcija koja služi za kreiranje paketa i njihovo objavljivanje na Python repozitorijumu upotrebom alata Poetry. Kreirana je u obliku Docker akcije. U Docker kontejneru se instaliraju svi potrebni alati za korišćenje Python-a i zatim se pokrenu Poetry komande za kreiranje i objavljivanje paketa:

poetry build - da kreira Python paket

poetry publish - da objavi paket na repozitorijumu koji je prethodno podešen

Paramtri koje ova akcija očekuje su:

- python_version - verzija Python-a koja se instalira u kontejneru i koristi za kreiranje paketa. Ako se ne navede, podrazumeva se da se koristi najnovija verzija. Za bolje performanse, poželjno je koristiti predefinisanu - najnoviju verziju.

- poetry_version - verzija Poetry-a koja se instalira u kontejneru i koristi za kreiranje paketa. Ako se ne navede, podrazumeva se da se koristi najnovija verzija. Navodi se u PIP sintaksi za specifikaciju verzija

- pypi_token - jedini obavezan parametar. Služi kao API token za autentifikaciju pri objavljivanju paketa na PyPI.

- repository_name - naziv repozitorijuma na kojem se objavljuje paket. Ako se ne navede, podrazumeva se da se objavljuje na PyPI repozitorijumu.

- repository_url - adresa repozitorijuma na kojem se objavljuje paket. Ako se ne navede, podrazumeva se da se objavljuje na PyPI repozitorijumu.

Takođe, potrebno je specificirati pyproject.toml datoteku na osnovu koje će se kreirati Python paket. Ova datoteka treba da se nalazi u korenskom direktorijumu repozitorijuma koji koristi ovu akciju.

U pjisp-template-name i pjisp-diff, ova akcija se izvršava samo kada se napravi novi privezak u repozitorijumu u obliku v*.*.*.Tok poslova se sastoji od jednog posla koji se pokreće na poslednjoj verziji Ubuntu operativnog sistema i 3 koraka. Prvi korak preuzima sadržaj repozitorijuma. Drugi korak menja verziju u pyproject.toml datoteci na verziju priveska koji je prouzrokovao pokretanje ove akcije. Poslednji korak koristi poetry-publish akciju za objavljivanje nove verzije paketa. Svi ulazni parametri koriste predefinisanu verziju, samo je pypi_token parametar postavljen na vrednost PyPI tokena koji se preuzima iz GitHub-ovog konteksta okruženja - secrets.

2.4. smoke-test

Još jedan alat otvorenog koda koji je korišćen i izmenjen u ovom radu je smoke_test [6]. To je konzolna aplikacija i API koji služi za testiranje studentskih zadataka na dim.

Ovaj alat se koristi i za testiranje zadataka koje nastavno osoblje piše kao primer tačno urađenog zadatka i samim tim se koristi u okviru pjisp-assignment-template alata. Zadatak koji ovaj alat očekuje treba da bude napisan u programskom jeziku C ili nekom jeziku od kojeg je GCC progrmaski prevodilac sposoban da napravi izvršnu datoteku.

Zadatak se prvobitno kompajlira pomoću GCC-a. Nakon toga zadatak se testira na slučajeve korišćenja opisane u .yaml datotekama koje nastavno osoblje priprema prilikom kreiranja zadatka. Rezultat je u ljudski čitljivom formatu, ali je izlazni kod u svakom slučaju, bilo pri pozitivnoj ili negativnoj evaluaciji studentskog rešenja uvek bio 0. Da bi smoke_test alat mogao da se upotrebi u okviru GitHub akcije koja proverava ispravnost rešenja koje nastavno osoblje kreira, i da bi se izvršavanje akcije prekinulo u slučaju negativne evaluacije studentskog rešenja, izmenjen je izlazni kod na vrednost 1 ako ponuđeno rešenje ne ispunjava sve uslove navedene u .yaml datotekama i ako je prosleđen argument -e kao argument komandne linije. Kada GitHub akcija u bilo kojem koraku dobije rezultat različit od 0, smatra se da taj korak nije uspešno izvršen i ne prelazi se na naredne korake. U slučaju pjisp-assignmen-template alata, to znači da autor zadatka nije dobro napisao primer ispravnog rešenja i da mora da ga ispravi.

2.5. pjisp-assignment-template

Alat pjisp-assignment-template nudi veliki broj funkcionalnosti za kreiranje i proveru ispravnosti kreiranog zadatka za studente. Ove funkcionalnosti su u velikoj meri učestvovale u dodavanju GitHub tokova poslova čije funkcionalnosti su opisane na samom početku poglavlja. Ipak, neke od funkcionalnosti je trebalo malo izmeniti i bilo je potrebno kreirati nekoliko novih.

Za spisak postojećih funkcionalnosti potrebno je pročitati dokumentaciju.

Nove funkcionalnosti koje uvodi projekat o kojem ovaj rad govori su:

- assignment-diff - za proveru da li su izmenjene sve datoteke koje je potrebno menjati i da lisu ostale iste sve datoteke koje se ne smeju menjati.

- assignment-check - za proveru izmena datoteka i proveru ispravnosti primera rešenja

- get-template - za pronalaženje naziva testa na osnovu naziva repozitorijuma i proveru ispravnosti naziva repozitorijuma

Takođe, uvedene su sitne izmene u neke od postojećih funkcionalnosti:

- init - dodatno, nakon generisanja datoteka na osnovu izabranog identifikatora testa, kreira se .template datoteka u koju se smešta identifikator za kasniju upotrebu

- test-solution - dodato je da pre pokretanja funkcionalnosti zahteva da postoji assignment_solution.c datoteka

- assignment-build - dodato je da pre pokretanja funkcionalnosti zahteva da postoji assignment_solution.c datoteka

- assignment-pack - dodato je da se pre pakovanja, pored provere ispravnosti primera rešenja, proveri da li su izmenjene sve datoteke koje je potrebno menjati i da li su ostale iste sve datoteke koje se ne smeju menjati

Kao zavisnosti u projektu dodati su pjisp-template-name i pjisp-diff korišćenjem Pipenv alata. Get-template funkcionalnost poziva pjisp-template-name sa nazivom repozitorijuma kao ulaznim parametrom i identifikatorom testa kao povratnom vrednošću. Assignment-diff poziva pjisp-diff sa identifikatorom testa kao ulaznim parametrom.

Nakon dodavanja malopre pomenutih izmena u kod pjisp-assignment-template alata, uvedene su sve funkcionalnosti potrebne za kreiranje GitHub tokova poslova čije funkcionalnosti su opisane na početku trenutnog poglavlja. Kreirane su dve nove datoteke na mestu koje je predviđeno za pisanje GitHub tokova poslova - github/workflows. Tokovi poslova nazvani su “Project create” i “PJISP assignment” i nalaze se u init-repo.yml i test-solution.yml datotekama respektivno.

2.5.1 Project create

“Project create” je tok poslova koji se pokreće, kako mu i samo ime kaže, izazvan događajem kreiranja projekta, odnosno nakon pravljenja repozitorijuma na osnovu šablon repozitorijuma pjisp-assignment-template. Uloga ovog toka posla je da postavi inicijalno stanje projekta. Na osnovu naziva repozitorijuma, zaključuje se na koji test se repozitorijum odnosi i onda se na osnovu identifikatora testa kreiraju datoteke namenjene tom testu. Identifikator testa može biti T12, T34 ili SOV.

U README.rst datoteku, koja sadrži sve bitne informacije o projektu i datotekama u njemu, postavlja bedž koji daje informacije o tome da li je zadatak kreiran kako treba, a menja se na osnovu rezultata drugog toka poslova - “PJISP assignment”.

Bedž može biti u jednom od tri stanja: “no status” kada nema status, “failing” kada se “PJISP assignment” neuspešno izvrši i “passing” kada se uspešno izvrši.

“Project create” tok posla u sebi ima jedan posao koji se sastoji od šest koraka. Posao se pokreće na Ubuntu operativnom sistemu na najnovijoj verziji.

Prvi korak preuzima sadržaj repozitorijuma.

Drugi korak instalira Python u verziji 2.7 u okruženju pokretača.

Treći korak je zadužen za instaliranje Pipenv-a, alata za upravljanje Python paketima koji pjisp-assignment-template koristi.

Četvrti korak je zadužen za inicijalizaciju datoteka na osnovu naziva repozitorijuma. Naziv repozitorijuma dobija se iz promenjive github.repository koja se preuzima iz konteksta okruženja. Nakon toga se instaliraju zavisnosti korišćenjem Pipenv alata i pokreće se get-template funkcionalnost. Ova funkcionalnost proverava ispravnost naziva repozitorijuma i na osnovu njega pronalazi identifikator testa. Ako je naziv repozitorijuma ispravan, poziva se init funkcionalnost sa identifikatorom testa kao ulaznim parametrom.

Peti korak služi za kreiranje bedža i njegovo dodavanje u README.rst datoteku. Poslednji, šesti korak je zadužen da postavi sve novonastale izmene na GitHub repozitorijum. Nakon postavljanja ovih izmena, sve nove datoteke nalaze se na GitHub repozitorijumu i bedž je vidljiv na vrhu README.rst datoteke. Bedž u početku ima failing vrednost, pošto zadatak nije spreman za studentei očekuje se od nastavnog osoblja da ga kreira.

2.5.2 PJISP assignment

"PJISP assignment" tok posla se pokreće na događaj push ili pull-request na glavnoj - master grani. Zadužen je da svaki put nakon izmene koda na repozitorijumu proveri da li je repozitorijum spreman za pregledanje od strane nastavnika i davanje studentima.

Ovaj tok posla u sebi ima jedan posao koji se sastoji od četiri koraka. Posao se pokreće na Ubuntu operativnom sistemu na najnovijoj verziji. Prvi korak preuzima sadržaj repozitorijuma. Drugi korak instalira Python u verziji 2.7 u okruženju pokretača. Treći korak je zadužen za instaliranje Pipenv-a. Svi koraci osim prvog imaju uslov pod kojim se izvršavaju, a to je da u svom nazivu ne sadrže "pjisp-assignment-template".

Ovaj uslov osigurava se na pjisp-assignment-template repozitorijumu neće pokretati ovi koraci, pošto je on samo šablonski repozitorijum i nikada neće sadržati konkretan zadatak za studente, pa samim tim nema smisla testirati njegovu ispravnost.

Poslednji, četvrti korak je zadužen za pokretanje testiranja zadatka. Prvo se instaliraju zavisnosti projekta korišćenjem Pipenv alata, a onda se pokreće assignment-check funkcionalnost. Ova funkcionalnost prvo pokreće assignment-diff. koji proverava da li su izmenjene sve datoteke koje je potrebno menjati i da li su ostale nepromenjene sve datotekekoje ne bi trebalo menjati. Ako se uspešno izvrši, prelazi se na test-solution funkcionalnost. Ako se ne izvrši uspešno, onda se izvršavanje koraka, a zatim i celog toka posla prekida i kreiranje zadatka se smatra neuspešnim. Test-solution funkcionalnost ispituje ispravnost primera rešenja na osnovu testova opisanih u .yaml datotekama. Potrebno je da se svi testovi uspešno izvrše da bi se zadatak smatrao ispravnim i onda se bedž postavlja na "passing" vrednost.

3. ZAKLJUČAK

Iz prethodnog poglavlja vidimo da se mnoge nove funkcionalnosti oslanjaju na funkcionalnosti koje GitHub Actions servis nudi. Ovaj servis je dosta mlad, nastao je u novembru 2019. godine i to sa sobom donosi neke probleme. Još uvek nije u potpunosti pouzdan, pošto su neke od njegovih funkcionalnosti povremeno nedostupne.

Takođe, zbog brzog razvoja softvera, dokumentacija nije u svakom momentu usklađena sa alatom, pa to nekada može predstavljati problem za programere koji žele da koriste nove funkcionalnosti alata čim se one pojave. I pored pomenutih problema, za potrebe ovakvog projekta, GitHub akcije donese mnoge beneficije i znatno olakšavaju razvoj i korišćenje svih prethodno pomenutih alata. Integracija alata je bila veoma jednostavna, pošto se šablon repozitorijum već nalazi na GitHub-u i onda nije bilo potrebno uvoditi dodatne alate u proces integracije.

Funkcionalnosti koje ove akcije uvode mogu znatno da smanje učestalost grešaka koje nastavno osoblje pravi prilikom kreiranja zadataka za studente. Olakšavaju i procenu spremnosti zadataka za davanje studentima, pošto bedž na početku README.rst datoteke u repozitorijumu zadatka već naznačava da je zadatak prošao sve testove koje alat trenutno pokreće nad njim i da je spreman za pregledanje od strane drugog nastavnika.

4. LITERATURA

[1] Petar Marić, "pjisp-assignment-template", https://github.com/petarmaric/pjisp-assignment-template (pristupljeno u avgustu 2020.)

[2] Jelena Dokić, "pjisp-template-name", https://github.com/JRubics/pjisp-template-name (pristupljeno u avgustu 2020.)

[3] Jelena Dokić, "pjisp-diff", https://github.com/JRubics/pjisp-diff (pristupljeno u avgustu 2020.)

[4] Jelena Dokić, "poetry-publish", https://github.com/JRubics/poetry-publish (pristupljeno u avgustu 2020.)

[5] Jelena Dokić, "publish-python-poetry-package", https://github.com/marketplace/actions/publish-python-poetry-package (pristupljeno u avgustu 2020.)

[6] Petar Marić, "smoke-test", https://github.com/petarmaric/smoke_test (pristupljeno u avgustu 2020.)

Kratka biografija:

Jelena Dokić rođena je u Novom Sadu 1996. god. Gimnaziju je završila 2015. godine, i iste godine upisala OAS na Fakultetu tehničkih nauka, smer Računarstvo i automatika. Diplomirala je 2019. god i upisala MAS na istom smeru. Master rad na Fakultetu tehničkih nauka iz oblasti Računarstva i automatike odbranila je2020.god. kontakt: jelena.dokic@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE44Miseljic

IMPLEMENTACIJA PODSISTEMA SKRIVENE MEMROIJE ZA RISC-V PROCESOR

IMPLEMENTATION OF CACHE SUBSYSTEM FOR RISC-V PROCESSOR

Đorđe Mišeljić, Vuk Vranjković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu je modelovan podsistem skrivene (keš, eng. cache) memorije za RISC-V procesor. Model je pisan u VHDL jeziku te je ciljan za soft-core primenu na FPGA uređajima. Model se sastoji iz dva nivoa keš hijerarhije: prvi nivo je direktno preslikan i razdeljen, dok je drugi N-smerno set asocijativan i unificiran. Model je parametrizovan, te korisnik ima mogućnost da bira kapacitet memorija kao i asocijativnost. Sistem je zajedno sa jednostavnim RISC-V procesorom prvo simuliran pomoću Vivado alata a zatim upakovan u IP jezgro, implementiran i testiran na Zybo razvojnoj ploči.*

Ključne reči: *Skrivena memorija, RISC-V, FPGA, Zybo*

Abstract – *This paper follows a modeling of a cache subsystem for the RISC-V processor. The model was written in VHDL language and is targeted at soft-core applications on FPGA devices. The model is composed of two levels of cache hierarchy: the first level is directly mapped and split, while the second level is N-way set associative and unified. The model is parametrized, so the user can choose cache size as well as associativity. A simulation of a simple RISC-V processor core with a cache subsystem was done in the Vivado development tool, after which it was packaged into an IP core, implemented, and tested on the Zybo development board.*

Keywords: *Cache memory, RISC-V, FPGA, Zybo*

1. UVOD

Poslednjih par decenija, dve arhitekture skupa instrukcija procesora (*eng. Instruction Set Architecture*) su kontrolisale svetsko tržište: x86 i ARM. Obećavajući faktor za promenu trenutnog stanja u hardverskoj industriji je nova, besplatna i otvorena arhitektura pod nazivom RISC-V [1]. Krećući se od osnovnog seta instrukcija, moguće je dodati proširenja koje su specijalno dizajnirana za različite komercijalne i naučne svrhe.

Sa ovim na umu, moguće je da RISC-V nađe primenu kako u oblasti visokih performansi tako i u oblasti uređaja male potrošnje energije, te na taj način napravi otvoreni standard za hardver. Prednost ovoga je razvoj softvera koji će biti kompatibilan i raditi na svim sličnim RISC-V jezgrima [2]. Posebna oblast od interesa za RISC-V arhitekturu su FPGA (eng. *Field Programmable Gate Array*) uređaji.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vuk Vranjković, docent.

Ukoliko neko implementira RISC-V procesorsko jezgro na FPGA čipu, često obezbedi i RTL (eng. *Register Transfer Level*) izvorni kod. Budući da RISC-V ima i punu softversku podršku Linux organizacije, moguće je bez ikakvih izmena koda, besplatno i legalno podići operativni sistem baziran na Linux-u, na bilo kojoj FPGA razvojnoj ploči [3]. Stoga, postoji velika zainteresovanost za dizajn *soft-core* RISC-V procesora, za koje se veruje da će verovatno zameniti ulogu Xilinx-ovog MicroBlaze procesora koji je do sada bio jedno od retkih rešenja za podizanje operativnog sistema na FPGA čipu. Ono što će izdvojiti RISC-V u odnosu na MicroBlaze je softverska podrška koja je uvek bila slaba tačka *soft-core* procesora zbog malog broja korisnika i odžavaoca softvera.

Ne sme se zaboraviti da je brzina pristupa memoriji i dalje usko grlo u performansama procesora. Svaki moderan procesorski čip na sebi ima barem dva nivoa keš memorije, koja zbog težnje da bude što veća, zauzima veliki deo ukupne površine čipa. Dakle, benefiti skrivenih memorija se ne smeju ignorisati, čak i u oblastima procesiranja sa malom potrošnjom energije. Kao i kod ASIC (eng. *Application-Specific Integrated Circuit*) implementacije, *soft-core* procesori na FPGA čipovima mogu iskoristiti različite tipove memorijskih resursa koji su na raspolaganju dizajneru, kao spregu između procesorske logike i operativne DDR RAM (eng. *Dual Data Rate Random Access Memory*). Ovaj rad će predstaviti jedan način iskorištenja konfigurabilnih memorijskih resursa koji su prisutni na svakom savremenom FPGA čipu, kako bi se implementiralo keširanje i zadovoljila memorijska zahtevnost RISC-V procesora.

2. MODEL KEŠ PODSISTEMA

Od osnivanja ideje o keširanju podataka, svaki savremeni memorijski sistem se deli na više memorijskih nivoa, gde se manje a brže memorije postavljaju što bliže procesoru, dok su veće a sporije memorije na periferiji [4]. Niži nivoi su manjeg kapaciteta, optimizovanije fabrikacije tranzistora (veća brzina, potrošnja i površina) te su bliži procesoru za manje kašnjenje na linijama. Uz pomoć dodatnog hardvera (keš kontrolera) proces prebacivanja memorije između različitih nivoa keševa i operativne memorije se izvršava automatski, bez potrebe intervencije programera.

Za datu memorijsku hijerarhiju, gde t_i predstavlja tehnološki zavisno kašnjenje pristupa memoriji na nivou hijerarhije i (krećući od strane procesora), realno vreme označeno sa T_i je veće jer zavisi i od ostalih nivoa hijerarhije memorije. Vreme T_i zavisi od udela pogodaka h_i (eng. *hit rate*) i promašaja m_i (eng. *miss rate*). Ovi brojevi govore koliko se često traženi podatak nalazi u memoriji na nivou i. Ukoliko se desi pogodak, procesor može da preuzme

podatak iz memorije na nivou *i*, a u suprotnom se prelazi na nivo *i*+1. Na sličan način se redom ispituju nivoi *i*+2...*N* dok se ne pronađe željeni podatak. Iz ovoga proizlaze formule 1, 2 i 3 [5].

$$h_{\mathrm{i}} + m_{\mathrm{i}} = 1 \quad (1)$$

$$T_i = h_i \cdot t_i + m_i \cdot (t_i + T_i + 1) \quad (2)$$

$$T_i = t_i + m_i \cdot T_i + 1 \quad (3)$$

Formula br. 3 se dalje može rekurzivno rastavljati zamenom člana T_{i+1} sa zavisnošću sa nivoom *i*+2. Za poslednji nivo hijerarhije memorije važi da je h_N=1 dok je m_N=0 jer se podatak mora nalaziti u njoj, te je $T_N = t_N$. Cilj pri dizajniranju sistema za keširanje je dobiti željeno vreme T_i unutar dozvoljenog budžeta.

2.1. Tipovi memorija na Zybo ploči

Za implementaciju keširanja na Zybo razvojnoj ploči se mogu iskoristiti dva tipa memorijskih resursa koji su prisutni u programabilnoj logici:

Blok RAM: postoji 60 blokova, gde se svaki može konfigurisati kao dvopristupna memorija veličine 36Kb ili dve jednopristupne memorije veličine 18Kb [6]. Čitanje memorijske lokacije se vrši na rastuću ivicu sinhroni-zacionog signala. Može se uključiti izlazni registar, čime se unosi dodatan takt kašnjenja pri čitanju, ali se dobija manje *clk-to-output* kašnjenje. Ukupno je na raspolaganju 2.1Mb ovog tipa memorije.

Distribuirani (*LUT, Lookup Table*) RAM: Od prisutnih 17,600 *slice* modula, 6000 su tipa *slicem*, dok je ostatak tipa *slicel* [7]. Svaki *slicem* modul se može konfigurisati kao 64bita brze RAM memorije [8]. Čitanje iz ove memorije je asinhrono, ali se može dodati izlazni registar za bolju *clk-to-output* karakteristiku. Ukupno je na raspolaganju 375Kb ovog tipa memorije. Brža a manja distribuirana (LUT) RAM će se iskoristiti za prvi nivo, dok sporija a veća blok RAM će se iskoristiti za drugi nivo keša.

2.2. Direktno preslikan keš

Podaci se između nivoa memorijske hijerarhije prenose u blokovima. Veličina bloka od *B* bajtova, znači da će nižih $b=ceil(log_2B)$ bita u adresi referencirati bajt u bloku (eng. *byte in block, bib*). Sa veličinom keša od *C* bajtova, širina adrese keš memorije će biti $c= ceil(log_2C)$ bita. Nižih *c* bita u adresi traženog podatka će indeksirati lokacije u kešu. Kod direktno preslikanog keša, kada se blok preuzima iz operativne memorije (eng. *fetch*), on se smešta na memorijske lokacije u kešu koje se poklapaju sa donjih *c* bita adrese. Može se zaključiti da postoji $2^{(31-c)}$ različitih blokova koji se mogu nalaziti na istoj lokaciji u kešu. Kako bi se znalo koji je blok trenutno na nekoj lokaciji u kešu, kada se on preuzima operativne memorije, gornjih (31-*c*) bita pod nazivom "tag" se čuva u pomoćnoj memoriji za čuvanje tagova (eng. *tag store*). Ova memorija se često implementira u istoj tehnologiji izrade kao i keš. U ovom radu, prvi nivo keša je implementiran kao direktno mapiran keš sastavljen od LUT RAM-a.

2.3. Set asocijativan keš

Najveća mana keširanja sa direktnim preslikavanjem je da se lako može napraviti sekvenca pristupa memoriji koja proizvodi udeo pogodaka jednak nuli - naizmeničan pristup dvema memorijskim lokacijama sa istim indeksom. Kako bi se ovo izbeglo, keš se pravi da bude N-set asocijativan. Keš se podeli u N manjih jednakih delova, svaki sa svojom memorijom za čuvanje tagova, te u istom trenutku može da čuva N podataka sa istim indeksom. Sa većom asocijativnošću se udeo pogodaka povećava na račun dodatne logike i brzine rada. Potrebno je imati N komparatora za poređenje tagova, te je potrebno pomoću dodatne kombinacione logike proslediti podatak iz jednog od N smerova. U ovom radu je implementiran parametrizovani N-smerni keš, sastavljen od blok RAM modula na FPGA čipu.

2.4 Polisa evikcije

Kada se keševi inicijalizuju, svi blokovi su nevalidni, te se N smerova popunjava redno. Ukoliko su svi blokovi u N smerova validni, onda se usvaja set pravila na osnovu kojih se odlučuje koji od blokova će biti zamenjen. Najočiglednija polisa je izbaciti najdavnije korišten blok (eng. *Least Recently Used, LRU*). Implementacija ove polise postaje problem za keševe sa većom asocijativnošću. U praksi se pokazuje da udeo pogodaka zavisi od programa na kojima se testira, te je prosečan udeo pogodaka za sličan za LRU i nasumičan odabir bloka [9]. Iz ovog razloga se pribegava tehnikama koje predstavljaju kombinaciju LRU i nasumičnog algoritma koje se zovu *Pseudo LRU* polise. U ovom radu je za N-smerni asocijativan keš drugog nivoa implementirana pseudo LRU polisa pod imenom "Žrtva - Sledeća žrtva" (eng. *Victim - Next victim*) [10]. Od N blokova u asocijativnom kešu samo se prate dva bloka: jedan označen kao žrtva (eng. *victim*) i jedan označen kao sledeća žrtva (eng. *next victim*), dok su svi ostali blokovi obični (*ordinary*).

2.5 Polisa upisa

Kada procesor izvrši naredbu upisa podatka u memoriju, te se desi pogodak, on promeni sadržaj nekog bloka u najnižem nivou keša. Pitanje je: kada ažurirati sledeći nivo keša sa novim podatkom? Ukoliko se keš dizajnira tako da se promeni sadržaj bloka samo najnižeg nivoa, dok blok zadržava prethodnu vrednost u sledećem nivou keša, sledeći nivo će biti ažuriran tek kada se blok eviktuje iz prethodnog nivoa. Ovakav keš se naziva "upis-nazad" (eng. *write-back*) keš. Druga bitna odluka vezana za upise u keševe je: da li se alociraju keš blokovi kada se desi promašaj pri instrukciji upisa? Pri metodi "alociraj pri promašaju upisa" (eng. *allocate on write miss*), keš blok se preuzima iz keša višeg nivoa ili operativne memorije, te se nakon toga izvrši upis. U ovom radu će postojati dva nivoa keša gde će oba biti metode "upis-nazad" i "alociraj pri promašaju upisa".

2.6 Polisa razdeljenosti

Većina savremenih procesora zahteva da se na svaku rastuću ivicu takta iz memorije preuzme instrukcija koju je potrebno dekodovati, a sem toga, potencijalno dodatan pristup memoriji ukoliko se radi o instrukciji upisa ili čitanja iz memorije. Stoga se keš prvog nivoa može implementirati kao jedna dvopristupna ili kao dve fizički odvojene memorije (za instrukcije i podatke). Prvi način se naziva ujedinjeni (eng. *unified*), a drugi se naziva razdeljeni (eng. *split*) keš.

Većina modernih procesora ima implementiranu protočnu obradu podataka, te se pri postavljanju komponenti na čip i optimizaciji rutiranja, desi da je logika faze za prihvat instrukcija fizički udaljena od faze za upis/čitanje podataka. Razdeljeni keš omogućava da se keševi za čuvanje instrukcija i podataka smeste na čipu blizu faze protočne obrade kojoj su potrebni. Na FPGA čipovima veliki problem prestavlja rutiranje, koje je mnogo ograničenije nego u ASIC implementaciji - zbog mnogo manjeg izbora za postavljanje komponenti te ograničenog broja kanala za rutiranje. Na rutiranju se često gubi veći deo performansa implementirane logike. Iz ovih razloga je dobar izbor da se prvi nivo keša implementira kao razdeljeni, a drugi nivo kao unificiran keš. Prvi nivo keša pokazuje najbolje rezultate kada su keševi za instrukcije i podatke jednake veličine [11].

2.7 Polisa inkluzije

Kod inkluzivnog keša, prisusvo bloka u nižem nivou keša implicira postojanje kopije istog bloka u višem nivou keša. Inkluzivan keš žrtvuje ukupan kapacitet keša zbog dobiti na jednostavnosti implementacije keš kontrolera. U ovom radu će dva nivoa keša biti inkluzivna.

3. IMPLEMENTACIJA SISTEMA NA ZYBO PLOČI

Implementacija sistema je izvršena u Vivado alatu kompanije Xilinx. Ciljana razvojna ploča je Zybo, kompanije Digilent, koja na sebi ima Zynq-7000 sistem na čipu (eng. *System on Chip*, *SoC*). Ploča poseduje dva DDR3 memorijska čipa, koja prave 32-bitni interfejs ka memoriji kapaciteta 512MB i propusnim opsegom 1050Mbps. Na Zynq SoC-u postoji već ugrađen memorijski kontroler sa 8 DMA (eng. *Direct Memory Access*) kanala za direktan pristup memoriji. Za potrebe testiranja, sistem za keširanje sa RISC-V procesorom je implementiran u programabilnoj logici, te je jedan od memorijskih kanala iskorišten za komunikaciju sa operativnom memorijom. Interfejs je tipa AXI3-Full širine 64 bita.

3.1. Pakovanje IP jezgra

Kako bi bilo moguće uvesti kontrolne i statusne signale za processor, kao i omogućiti da keš sistem pravilno komunicira sa DDR memorijskim čipom, moraju se modelovati *AXI-Lite Slave* i *AXI-Full Master* interfejsi. 32-bitni RISC-V processor koji podržava osnovni set instrukcija za manipulaciju celobrojnim vrednostima RV32I, je zajedno sa opisanim podsistemom za keširanje oklopljen u IP (eng. *Intelectual Property*) jezgro. Upakovana su i dva dodatna modula koja implementiraju potrebne AXI interfejse te služe kao sprega za komunikaciju sa Zynq SoC-om.

3.2. Blok dizajn u Vivado alatu

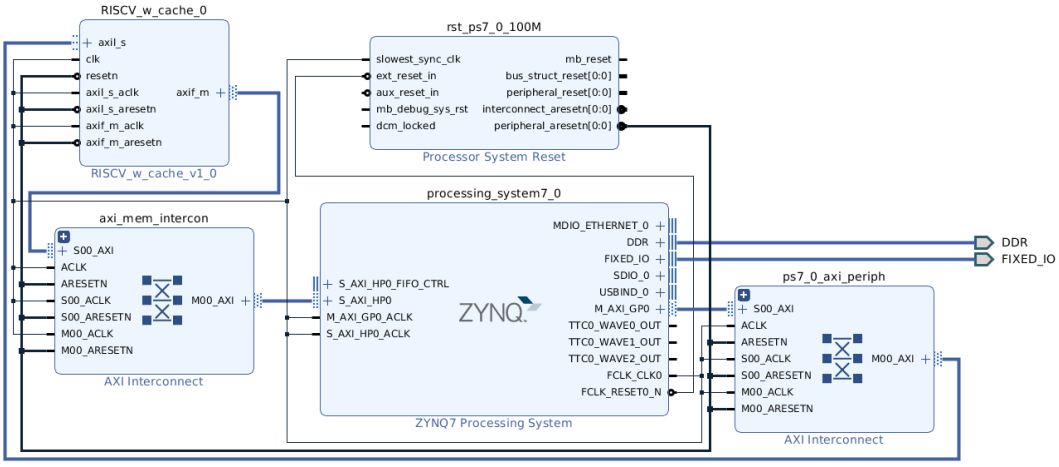
Prethodno opisano IP jezgro je povezano sa Zynq SoC-om kao na slici 1. 32-bitni AXI4 Master interfejs jezgra RISCV_AXI je povezan sa 64-bitnim AXI3 Slave interfejsom memorijskog kontrolera preko interkonekt modula. Interkonekt pravi spregu između AXI4 i AXI3 protokola, te vrši pakovanje podataka u 64-bitni interfejs. Dodatno je omogućena funkcija interkonekta za unutrašnji FIFO (eng. *First In First Out*) bafer, kako bi potencijalna memorijska kašnjenja unutar transakcije bila neprimetna iz tačke gledišta RISCV_AXI jezgra. Sistem je implementiran i testiran sa sledećom konfiguracijom parametara:

- *BLOCK_SIZE* = 64; - veličina keš bloka je 64B.
- *LVL1_CACHE_SIZE* = 1024*;* - veličine keševa prvog nivoa su 1KB (za instrukcije i podatke).
- *LVL2_CACHE_SIZE* = 4096*;* - veličina svakog smera u drugom nivou keša je 4KB.
- *ASOCIATIVITY* = 4; - drugi nivo poseduje četri smera u set asocijativanom kešu.
- *TS_BRAM_TYPE = HIGH_PERFORMANCE;* - koristi se dodatan izlazni registar u memorijama za čuvanje tagova drugog nivoa.

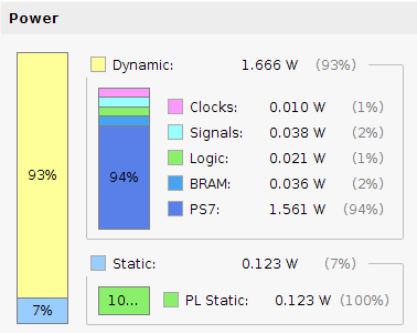
Frekvencija rada sistema je podešena na 100MHz. Sinteza i implementacija su uspešno završene. Iskorištenost resursa nakon implementacije se može videti u tabeli 1, a potrošnja energije na slici 2.

Tabela 1. *Iskorištenost resursa*

Resurs	Zauzeto	Dostupno	Zauzeto %
LUT6	5063	17600	28.77
LUTRAM	512	6000	8.53
FF	3308	35200	9.40
BRAM	8	60	13.33
BUFG	1	32	3.13



Slika 1. *Blok dijagram sistema na Zybo ploči*



Slika 2. *Potrošnja energije*

3.3. Testiranje u Vitis alatu

Za svrhe testiranja je u blok dizajnu prikazanom na slici 1, dodat modul *System ILA* (eng. *Integrated Logic Analyzer*). Ovaj modul je povezan na AXI Full interfejs RISCV_AXI modula, te koristi blok RAM da zabeleži transakcije pri testiranju na Zybo ploči. Na ovaj način možemo proveriti da li se transakcije koje inicira sistem za keširanje poklapaju sa zahtevima programa, te dobiti informaciju o realnim kašnjenjima pri pristupu memoriji.

ILA modul je podešen da detektuje transakcije čitanja iz memorije te da u tom trenutku sačuva vrednosti signala na AXI interfejsu. Na osnovu zabeleženih vrednosti se pokazuje da se od zahteva čitanja bloka, do čitanja poslednjeg podataka iz DDR memorijskog čipa utroši približno 42 mašinska ciklusa.

Od trenutka kada se na AXI Full interfejsu postavi adresa zahtevanog bloka, do prijema prvog podatka prođe 26 ciklusa, koji se gube zbog sporog odziva DDR čipa. Ostalih 16 ciklusa je potrebno da se blok od 64B prenese preko magistrale širine 32 bita. Ukoliko se ovo uporedi sa prenosom bloka iz drugog nivoa keša što zahteva 16 mašinskih ciklusa za prenos podataka sa dodatna 3 takta za proveru tagova i ažuriranje pomoćnih bita, očigledno je zašto je memorijski sistem sa više nivoa keševa neophodan za dobre performanse procesora.

Može se takođe zaključiti da za manje blokove, odnos korisnih ciklusa u odnosu na utrošene u čekanju na DDR čip, naglo opada. Slični rezultati se dobijaju za upis bloka podataka u DDR čip kada se ILA podesi u modu za detekciju transakcija upisa bloka podataka. Od inicijalizacije transakcije upisa implementaciju memoriju, do odgovora memorije da je upis izvršen uspešno (eng. *response*), utroši se 38 mašinskih ciklusa.

4. ZAKLJUČAK

Kako bi se premostila razlika između brzine rada procesora i tipičnog DDR3 čipa, u *soft-core* primeni RISC-V arhitekture, mogu se iskoristiti već postojeći memorijski resursi na FPGA ploči kako bi se implementirao jednostavan a efikasan sistem za keširanje. Distribuirani RAM može da obezbedi prvi nivo keša malog kapaciteta ali dovoljno brzog odziva da zadovolji zahteve pristupa memoriji bez zaustavljanja protočne obrade.

Nasuprot tome, blok RAM može da obezbedi dovoljno velik kapacitet keša kako bi se aktivan set podataka aplikacije čuvao na FPGA čipu. HDL model koji je predstavljen u radu je samo jedan način realizacije sistema za keširanje.

Implementirana inkluzivnost, pamćenje prisustva blokova prvog nivoa u pomoćnim bitima, kao i slobodan port u memorijama drugog nivoa keša izgleda kao dobro rešenje za višejezgarni *soft-core* processor, jer predstavlja odličnu osnovu za laku implementaciju MOESI (*Modified, Owned, Exclusive, Shared, Invalid*) ili sličnog algoritma za održanje koherencije između jezgara [12].

5. LITERATURA

[1] K. Asanović, D.A. Patterson, "Instruction Sets Should Be Free: The Case For RISC-V", EECS Dept., U.C. Berkley, CA. UCB/EECS-2014-146. Aug.6, 2014. Dostupno: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.pdf

[2] https://riscv.org/why-risc-v/ (pristupljeno u avgustu 2020.)

[3] https://www.linuxfoundation.org/the-linux-foundation/2018/11/the-linux-foundation-and-risc-v-foundation-announce-joint-collaboration-to-enable-a-new-era-of-open-architecture/ (pristupljeno u septembru 2020.)

[4] M. V. Wilkes, "Slave Memories and Dynamic Storage Allocation". IEEE Trans. Electron. Comput., pp. 270-271, Apr. 1965.

[5] J. C. Hoe, Predavanje, "Memory Technology and Organization" Carnegie Mellon Univ., Pittsburgh, PA, Mar. 2020. Dostupno: http://users.ece.cmu.edu/~jhoe/course/ece447/S20handouts/L14.pdf

[6] Xilinx, *7 Series FPGAs CLB User Guide (UG474)* Dostupno: https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[7] Xilinx, *Zynq-7000 SoC Data Sheet: Overview (DS190)*, 2018. Dostupno: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[8] Xilinx, *7 Series FPGAs Memory Resources User Guide (UG473)*, 2018. Dostupno: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf

[9] L. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer". IBM Syst. J., no. 5, pp.78-101, 1966.

[10] O. Mutlu, Predavanje, "Caches". Carnegie Mellon Univ., Pittsburgh, PA, Feb. 2015. Dostupno: https://course.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=onur-447-spring15-lecture18-caches-afterlecture.pdf

[11] J. Bell, D. Casasent, C.G. Bell, "An Investigation of Alternative Cache Organizations". IEEE Trans. Comput., pp. 346-351, Apr. 1974.

[12] J. Hennessy, D. Patterson, *Computer Architecture : A Quantitative Approach*, 6th ed., San Francisco, CA: Morgan Kaufmann Publishers Inc., 2017, pp 377-388.

Kratka biografija:

Đorđe Mišeljić rođen je na Cetinju (Crna Gora) 1995. god. Završio je opštu gimnaziju u SMŠ Ivan Goran Kovačić 2014. god u Herceg Novom. Bečelor rad na Fakultetu tehničkih nauka, usmerenje Embeded sistemi i algoritmi, odbranio je 2018. godine, kada je izabran u zvanje saradnik u nastavi. Kontakt: djordjemiseljic@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE45Kovacevic

IMPLEMENTACIJA VEKTORSKOG PROCESORA BAZIRANOG NA RISC-V SETU INSTRUKCIJA

IMPLEMENTATION OF A VECTOR PROCESSOR BASED ON A RISC-V INSTRUCTION SET

Nikola Kovačević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu prezentovan je 32-bitni vektorski procesor baziran na RISC-V setu instrukcija. Sistem je implementiran pomoću VHDL jezika za opis hardvera i namenjen je za soft-core primenu na FPGA platformama. Procesor je podeljen na dve celine, skalarno jezgro koje implementira RISC-V integer set instrukcija i vektorsko jezgro koje implementira RISC-V vektorski set instrukcija. Vektorsko jezgro je parametrizovano promenljivim brojem vektorskih linija, što omogućava korisniku da bira između performansi i ukupnog zauzeća resursa. Sistem je testiran na Zybo razvojnoj ploči, pri čemu je Vivado alat korišćen za njeno programiranje, analizu performansi i analizu utrošenih resursa.*

Ključne reči: *RISC-V, vektorski procesor, FPGA, Zybo.*

Abstract – *This paper presents a 32-bit vector processor based on the RISC-V instruction set. The system is implemented using VHDL hardware description language for soft-core applications on FPGA platforms. The processor is split into two parts, the scalar core, that implements the RISC-V integer instruction set, and the vector core that implements the RISC-V vector instruction set. The number of vector lanes inside the vector core is parametrized, so the user can make area-versus-performance trade-offs. The system was tested on a Zybo development board, using the Vivado tool to program it, and analyze resource utilization and performance.*

Keywords: *RISC-V, vector processor, FPGA, Zybo.*

1. UVOD

2004. godine Denardovo skaliranje (*eng. Dennard scaling*) [1] prestaje da važi i frekvencije rada procesora se od tada sve sporije povećavaju.

Ta promena je mikroprocesorsku industriju primorala da pronalazi nova rešenja kako bi povećala performanse procesora i ono što se pokazalo kao prekretnica jeste stavljanje akcenta na paralelizam prilikom obrade podataka. Trenutno, najuspešnija arhitektura za paralelnu obradu podataka jeste GPU (*eng. Graphics processing unit*).

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vuk Vranjković.

No, Vektorski procesori, koji datiraju još od 60-tih godina prošlog veka, jedna su od arhitektura kod kojih je ova vrsta paralelizma takođe izražena, ali su do skora smatrani jako „skupim“.
Jedan razlog je broj tranzistora, ali drugi, možda i bitniji, potreba za DRAM (*eng. Dynamic Random Accesss Memory*) memorijama koje mogu dovoljno brzo da „nahrane“ vektorski procesor podacima [1].

Napretkom tehnologije, potrebom za što većom paralelizacijom obrade podataka, sa što većom energetskom efikasnošću, pojavom RISC-V instrukcijskog seta, ovi procesori su ponovo skrenuli pažnju na sebe. Iz tih razloga ovaj rad se bavi analizom i implementacijom vektorskog procesora baziranog na RISC-V arhitekturi.

2. UVOD U VEKTORSKE PROCESORE

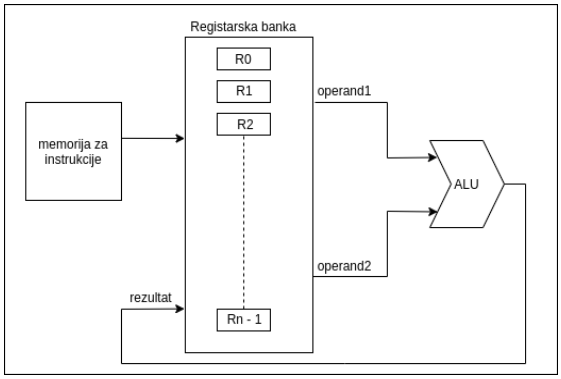
Kako bi povećali performanse, preko granica koje dopušta trenutna tehnologija izrade čipova, sistemske arhitekte pribegavaju različitim vrstama paralelizacije:

- Paralelizam na nivou instrukcija, skraćeno ILP (*eng. Instruction Level Parallelism*), omogućava istovremeno izvršavanje više instrukcija iz jednog sekvencijalnog skupa. Najjednostavniji primer su procesori sa protočnom obradnom [1], kod kojih pre nego što se završi egzekucija prethodne instrukcije, kreće se sa narednom.
- Paralelizam na nivou niti, skraćeno TLP (*eng. Thread Level Parallelism*), omogućava istovremeno izvršavanje instrukcija iz više odvojenih skupova. Najočigledniji primer su multiprocesorski sistemi kod kojih svaki procesor može da izvršava njemu dodeljen skup instrukcija.
- Paralelizam na nivou podataka, skraćeno DLP (eng. *Data Level Parallelism*), omogućava izvršavanje jedne iste operacije istovremeno nad nizovima elemenata. Primer su vektorski procesori koji jednu vektorsku instrukciju primenjuju na više podataka istovremeno.

Od prethodne tri vrste paralelizma, DLP se znatno bolje skalira, jer su podaci nad kojima vektorski procesor vrši određenu operaciju, na osnovu prihvaćene instrukcije, međusobno nezavisni, dok kod arhitektura koje iskorišćavaju TLP i ILP, rešavanje zavisnosti između instrukcija zahteva dodatni hardver.

2.1. Vektorsko procesiranje

Princip rada skalarnih procesora predstavlja dobar uvod u vektorske procesore, te će stoga početak ove sekcije biti posvećen tome. Slika 1 ilustruje pojednostavljenu strukturu skalarnog procesora i na njoj su prikazane 3 ključne komponente: memorija za instrukcije, registarska banka i ALU (aritmetičko logička jedinica). Registarska banka unutar sebe skladišti određeni broj registara i na osnovu prihvaćene instrukcije jedan par tih registara biće pročitan i sproveden do aritmetičko logičke jedinice. Nad njima će se izvršiti određena operacija u zavisnosti od prihvaćenene instrukcije i rezultat će biti smešten u registarsku banku. Struktura vektorskih procesora je jako slična prethodno opisanoj, kao što se može videti na slici 2. Ključne komponente su iste, osim što je registarska banka zamenjena vektorskom registarskom bankom.

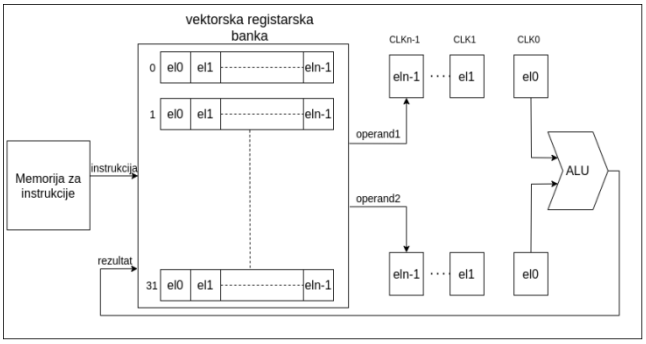


Slika 1. *Pojednostavljena struktura skalarnog procesora*

Razlika između te dve komponente jeste u tome što se registarska banka sastoji od registara određene širine (broj bita je određen arhitekturom procesora) dok se vektorska registarska banka sastoji od vektora, pri čemu svaki vektor unutar sebe sadrži određeni broj elemenata (širina pojedinačnog elementa je takođe određena arhitekturom). Prihvatom vektorske instrukcije, kao što se kod skalarnog procesora čitaju registri registarske banke i izvršava određena operacija nad njima, tako se kod vektorskog procesora, iz vektorske registarske banke, čitaju vektori i jedna ista operacija se izvršava nad elementima unutar njih. Unutar instrukcije se nalazi informacija kojim se vektorima pristupa i svakim taktom biće pročitan jedan par elemenata, počevši od elemenata na indeksu „*i*“. Nad njima će se izvršiti određena operacija i rezultat će biti smešten u određeni vektor unutar vektorske registarske banke.

2.2. Prednosti vektorskog seta instrukcija

Iz razloga što jedna kratka instrukcija može da opiše N operacija i da adresira 3N registarskih operanada, vektorski kod je kompaktan i neophodna propusna moć prilikom prihvata instrukcija iz memorije je mnogo manja. Vektorski set instrukcija umanjuje hardver neophodan prilikom dekodovanja vektorske instrukcije, jer se jedna vektorska instrukcija primenjuje N puta na *N* elemenata. Takođe, *N* operacija koje se izvode su međusobno nezavisne, te iz tog razloga nije neophodan hardver za detekciju zavisnosti.



Slika 2. *Pojednostavljena struktura vektorskog procesora*

Šablon koji vektorski procesor prati prilikom izvođenja operacija nad elementima vektorskog registra je regularan. To omogućava visok stepen paralelizma, jer bi vektorski procesor mogao da se implementira pomoću više paralelnih linija (*eng. Vector lanes*), pri čemu bi svaka linija vršila operacije nad jednim delom vektorskih elemenata. Vektorski set instrukcija može se dodati kao ekstenzija na već postojeći skalarni set.

2.3 RISC-V ISA(eng. Instruction Set Architecture)

Arhitektura skupa instrukcija (eng. ISA, Instruction Set Architecture) opisuje na koji način određeni procesor funkcioniše i koje su njegove mogućnosti. Ona opisuje registre koje će procesor imati kao i sve mašinske instrukcije koje će podržavati [2]. Iz tog razloga prilikom projektovanja procesora neophodno je odabrati određenu arhitekturu, čiji set ili podset instrukcija će biti podržan. U ovom radu odabrana je RISC-V ISA [3]. Ova arhitektura, koja je potekla sa Berkli (*eng. Berkeley*) univerziteta, je novi set instrukcija (ISA) koji je na početku bio zamišljen da podrži naučna istraživanja i edukaciju, ali za koji sada postoji nada da će postati arhitektura koja će biti besplatna i otvorena za sve industrijske implementacije.

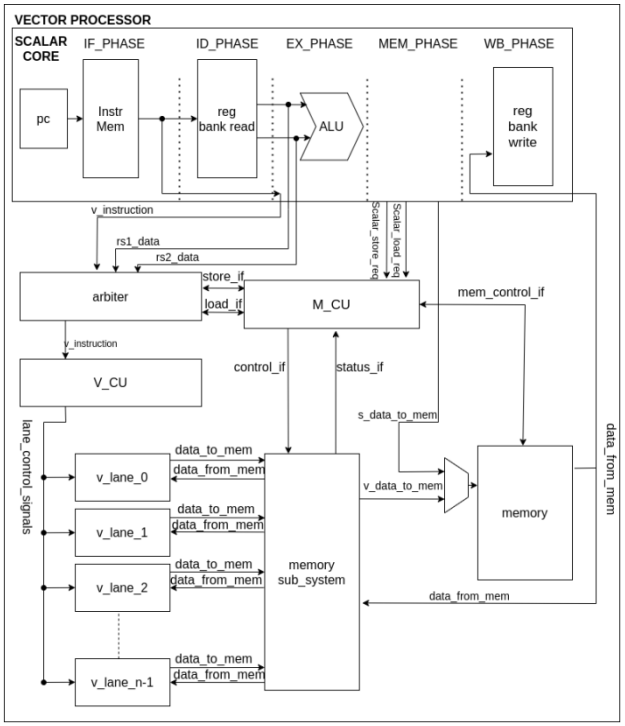
2.4 FPGA platforma i motivacija za njeno korišćenje

Vektorski procesor opisan u ovom radu je realizovan pomoću FPGA (*eng. Field Programmable Gate Arrays*) platforme. To su poluprovodnički uređaji zasnovani na matricama konfigurabilnih logičkih blokova povezanih pomoću programabilnih interkonekcija.

Sistemi koji se realizuju na FPGA platformama su najčešće akceleratori za aplikacije koje iskorišćavaju paralelizam između podataka. To su hardverski blokovi dizajnirani da obavljaju jednu vrstu zadatka sa ne toliko konfigurabilnih opcija. Oblasti u kojima je ovakav pristup zastupljen su: obrada slike i videa, mašinsko učenje (*eng. Machine learning*), emulacija hardvera, itd. Ovaj rad istražuje alternativnu mogućnost korišćenja FPGA platforme za kreiranje vektorskog procesora kao akceleratora opšte namene. Procesor bi posedovao standardan set instrukcija, tako da bi svako, bez iskustva sa dizajnom hardvera, mogao da ga programira. Takođe, zbog mogućnosti reprogramiranja dizajna na FPGA platformama, moćiće da se menjanju neke od karakteristika procesora kako bi se povećale performanse ili kako bi se optimizovala iskorišćenost resursa.

3. ARHITEKTURA VEKTORSKOG PROCESORA

U ovoj sekciji biće opisana RISC-V vektorska ekstenzija kao i mikroarhitektura procesora baziranog na njoj (slika 3). Osnovna ideja je da se pored skalarnog jezgra, koje podržava RISC-V *integer* set instrukcija, implementira i vektorsko jezgro koje bi se ponašalo kao ekstenzija na već postojeći set instrukcija.



Slika 3. *Blok dijagram vektorskog procesora*

3.1 Opis skalarnog jezgra

Skalarno jezgro (*scalar core*, slika 3) je 32-bitni procesor bez dinamičkog izvršavanja instrukcija (*eng. In-order processor*), sa pet faza protočne obrade, koji implementira RISC-V *integer* set instrukcija.

Osnovna uloga skalarnog jezgra jeste prihvat i prosleđivanje vektorskih instrukcija vektorskom jezgru, kao i izvršavanje ne vektorskog koda (skalarnih instrukcija).

3.2 Programski model vektorske ekstenzije

Vektorska ekstenzija dodaje 32 vektorska registra (V0 – V31) na već postojeće registre skalarnog jezgra. Svaki vektorski registar biće predstavljen preko *VLEN* bita, pri čemu *VLEN* mora da bude stepen broja 2. To znači da ukoliko je potrebno da vektorski registar unutar sebe sadrži 32 elementa, pri čemu bi svaki bio 32-bitni, *VLEN* mora da bude 1024.

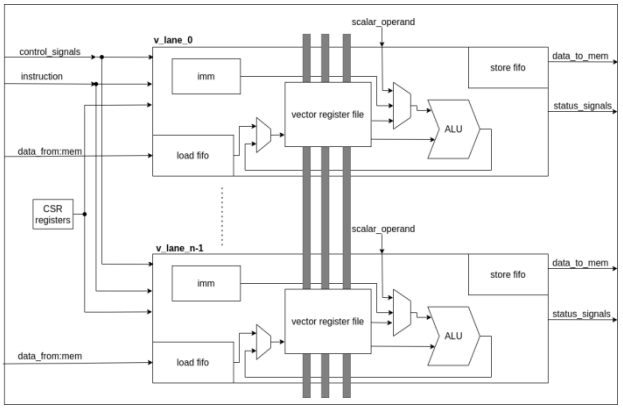
Pored vektorskih registara, ekstenzija dodaje i 2 neprivilegovana CSR (*eng. Control Status registers*) registra (*vtype* i *vl*) koji programeru mogu da daju veću kontrolu.

U 0.8 RISC-V „V“ nacrtu [4] predviđeno je da postoji 7 neprivilegovanih CSR registara, ali zbog skupa instrukcija koje će ovaj procesor da poseduje zaključeno je da su *vtype* i *vl* registri dovoljni.

3.3 Mikroarhitektura vektorskog jezgra

Vektorsko jezgro (slika 3) čine: A*rbiter*, V_CU, M_CU, vektorske linije i *memory_sub_system.*

Vektorske linije (slika 3) su blokovi označeni sa *v_lane_0* do *v_lane_n*-1, što naznačava da je broj linija u vektorskom jezgru parametrizovan (u trenutnoj implementaciji broj linija mora bit stepen broja 2). Svaka vektorska linija sadrži kopiju funkcionalnih jedinica, deo vektorske registarske banke (pola ukoliko je broj linija 2, četvrtinu ukoliko je broj linija 4, itd), *load* i *store FIFO (eng. First In First Out)* memorije i mrežu za rutiranje. Takođe, svaka vektorska linija ima u potpunosti isti interfejs i kontrolisana je od strane istih kontrolnih signala. Na slici 4 prikazan je blok dijagram vektorskih linija.



Slika 4. *Blok dijagram vektorskih linija*

Vektorska registarska banka (*VRF - vector register file*) je distribuirana između vektorskih linija (označeno sivim linijama na slici 4) i na ovaj način izbegnuta je tradicionalna arhitektura jedne centralizovane banke [5] koja je zahtevala previše portova za upis i čitanje.

Distribucija je realizovana tako da svaka banka unutar vektorskih linija ima 32 vektorska registra sa *VECTOR_LENGTH*/*NUM_OF_LANES* elemenata. *VECTOR_LENGTH* je parametar koji određuje koliko elemenata ima u jednom vektoru, dok je *NUM_OF_LANES* parametar koji određuje koliko ima vektorskih linija.

Load i Store FIFO blokovi (slika 4) su posrednici između memorije sa podacima (*memory, slika 3*) i VRF bloka. Prilikom izvršavanja instrukcije prihvata podataka podaci se prvo smeštaju u *load FIFO* memoriju i kada se svi prihvate, onda se, ukoliko vektorska linija nije zauzeta izvršavanjem druge instrukcije, smeštaju u *VRF*. Slično važi i za instrukcije upisa podataka u memoriju sa podacima, stim što se tada, podaci prvo smeštaju iz VRF modula u *store FIFO* memoriju i tek kada se svi smeste, onda se prosleđuju u memoriju sa podacima. Na ovaj način omogućeno je paralelno izvršavanje drugih instrukcija (ukoliko su nezavisne) dok vektorski procesor prenosi podatke iz memorije ili u memoriju.

Uloga Arbiter bloka je da prihvata instrukcije od skalarnog jezgra i da u zavisnosti od tipa prihvaćene instrukcije prosledi vektorsku instrukciju, vrednosti iz skalarnih registara i kontrolne signale *V_CU* modulu,

M_CU modulu i vektorskim linijama. Arbiter je takođe zadužen za rešavanje zavisnosti prilikom paralelnog izvršavanja instrukcija prihvata/prenosa podataka i ostalih instrukcija.

Uloga *V_CU* (slika 3) bloka je da na osnovu vektorske instrukcije koju dobija od *Arbiter* komponente, generiše kontrolne signale koji upravljaju vektorskim linijama.

Memory_sub_system (slika 3) blok je zadužen da u cikličnom maniru prosleđuje podatke iz memorije sa podacima ka vektorskim linijama i obrnuto, od vektorskih linija ka memoriji sa podacima.

M_CU (slika 3) generiše kontrolne signale za memorijskim podsistemom i komunicira sa *Arbiter* komponentom i skalarnim jezgrom kako bi na osnovu informacija koje dobije od njih započeo transakcije prenosa podataka iz memorije ka procesoru, ili obrnuto, od procesora ka memoriji.

4. ANALIZA ISKORIŠĆENOSTI RESURSA I PERFORMANSI

Prethodno opisani sistem implementiran je na *Zybo* razvojnoj ploči koja pripada *Zynq-7000* familiji sistema na čipu (*eng. System on chip*) [6]. Za njeno programiranje korišćen je *Vivado* alat kompanije *Xilinx* [7], koji omogućava sintezu i implementaciju sistema opisanih u HDL (*eng. Hardware Description Language*) jezicima, kao i analizu utrošenih resursa i performansi. U tabelama 1 i 2 je prikazan izveštaj *Vivado* alata o utrošenosti resursa na *Zybo* razvojnoj ploči kada je broj vektorskih linija koje procesor poseduje 8 i 1, respektivno.

Tabela 1. *Iskorišćenost resursa kada je broj vektorskih linija 8*

Resurs	Zauzeto	Dostupno	Zauzeto %
LUT6	15156	17600	86.11
LUTRAM	883	6000	14.72
FF	7897	35200	22.43
BRAM	36.5	60	60.83
DSP	48	80	60

Tabela 2. *Iskorišćenost resursa kada je broj vektorskih linija 1*

Resurs	Zauzeto	Dostupno	Zauzeto %
LUT6	8209	17600	46.64
LUTRAM	787	6000	13.12
FF	7422	35200	21.09
BRAM	24	60	40
DSP	6	80	7.5

Ova dva slučaja su uzeta u obzir jer *Zybo* razvojna ploča nema dovoljno resursa da implementira procesor sa 16 vektorskih linija. Zauzetost BRAM ćelika je velika jer se one koriste za implementaciju VRF modula, *LOAD_FIFO* modula, *STORE_FIFO* modula, memorije sa podacima i memorije sa instrukcijama. Iz tabela 1 i 2 se može videti drastična razlika u zauzetosti DSP ćelija, razlog za to je taj što svaki ALU modul unutar vektorskih linija koristi 6 DSP ćelija. Frekvencija rada procesora sa 8 vektorskih linija iznosi 95 MHz, dok za procesor sa 1 vektorskom linijom ona iznosi 96 MHz. Na maksimalnu frekvenciju utiče količina iskorišćenih resursa na FPGA platformi, jer što više resursa je neophodno, to je alatu teže da vrši povezivanje (*eng. Routing*).

5. ZAKLJUČAK

U ovom radu izvršena je vektorska ekstenzija 32-bitnog procesora, koji implementira RISC-V *integer* set instrukcija, prateći verziju 0.8 RISC-V - „V“ nacrta za vektorsku ekstenziju [4]. Za razliku od standardne implementacije procesora pomoću ASIC (*eng. Application Specific Integrated Circuit*) tehnologije, ovaj rad istražuje alternativnu mogućnost kreiranja vektorskog procesora koristeći FPGA platformu. Specifičnost ovih platformi je mogućnost reprogramiranja, čime bi mogle da se menjanju neke od karakteristika procesora kako bi se povećale performanse ili kako bi se optimizovala iskorišćenost resursa. Implementacija sistema izvršena je na *Zybo* razvojnoj ploči pomoću *Vivado* alata.

6. LITERATURA

[1] J. L. Hennessy and D. A. Patterson, „Computer Architecture - A Quantitative Approach, Sixth Edition“, Morgan Kaufmann, 2017

[2] Đ. Mišeljić i N. Kovačević, Napredni mikroprocesorski sistemi, Upoznavanje sa RISC-V procesorom, 2019.

[3] A. Waterman, K. Asanović, „The RISC-V Instruction Set Manual Volume I: User-Level ISA Document Version 2.2“, CS Division, EECS Department, University of California, Berkeley, 2017.

[4] „Working draft of the proposed RISC – V „V“ vector extension“, https://github.com/riscv/riscv-v-spec (pristupljeno u septembru 2020).

[5] C. Kozyrakis and David Patterson, „Overcoming the Limitations of Conventional Vector Processors“, Proceedings of the International Symposium on Computer Architecture, 2003.

[6] Zybo reference manual, https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual (pristupljeno u septembru 2020.)

[7] *https://www.xilinx.com/*, (pristupljeno u septembru 2020.)

Kratka biografija:

Nikola Kovačević rođen je u Loznici 1995. god. Bečelor rad na Fakultetu Tehničkih Nauka, usmerenje Embeded sistemi i algoritmi, odbranio je 2018. godine, kada je izabran u zvanje saradnika u nastavi. Kontakt: nikolakovacevicftn@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE47Vasiljevic

SPRINGEXTRACTOR APLIKACIJA ZA AUTOMATIZACIJU PROCESA PREGLEDANJA RADOVA

SPRINGEXTRACTOR APPLICATION FOR AUTOMATING THE PROCESS OF REVIEWING STUDENT PAPERS

Aleksandar Vasiljević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je opisano idejno rešenje problema automatizacije procesa pregledanja studentskih radova. U tu svrhu implementirana je aplikacija Spring-Extractor – rešenje koje se oslanja na radni okvir Spring, relacionu bazu podataka MySQL i Angular frejmvork. Uz to, dat je uporedni prikaz prednosti i mana napredne u odnosu na inicijalnu verziju, a izložene su i potencijalne buduće tačke proširenja i unapređenja.*

Ključne reči: *automatizacija, aplikacija, Java, Spring, MySQL, Angular, proces pregledanja radova*

Abstract – *The paper describes the conceptual solution to the problem of automating the process of reviewing student papers. For this purpose, the SpringExtractor application was implemented – a solution that relies on the Spring framework, relational MySQL database and Angular framework. In additional, a comparative overview of the advantages and disadvantages of the advanced version is given, and potential future points of expansion and improvement are presented.*

Keywords: *automatization, application, Java, Spring, MySQL, Angular, the process of reviewing student papers*

1. UVOD

Inicijalno su računari bili prilagođeni rešavanju relativno jednostavnih problema budući da je njihova resursna moć bila ograničena. Kako su se problemi usložnjavali, javila se potreba za nadogradnjom računarske inteligencije, što je posledično označilo podizanje nivoa apstrakcije, a samim tim i mogućnost rešavanja kompleksnijih problema šireg spektra.

Integracija sve moćnijih računara, novorazvijenih programskih jezika i potreba za podizanjem nivoa apstrakcije stvorila je osnovu za kreiranje sofisticiranih softverskih rešenja prilagođenih optimizaciji i automatizaciji problema koji se rešavaju.

Neke od dobrobiti procesa automatizacije softvera su: ušteda vremena, smanjenje napora koji je potrebno da korisnik softvera uloži, nije neophodno domensko poznavanje problema koji se rešava i sl. Automatizacija danas ima važnu ulogu u savremenim modelima nastave jer pozitivno utiče na poboljšanje njenog kvaliteta.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Popov, vanr. prof.

2. IDEJNI PUT RAZVOJA APLIKACIJE SPRINGEXTRACTOR

Ideja o razvoju aplikacije koja bi vršila automatizaciju procesa pregledanja radova nastala je usled problema sa vremenom: prevelik broj studenata, a nedovoljno nastavnog osoblja do sada je značilo dugotrajno čekanje na rezultate održanog testa/ispita. Da bi se taj problem rešio bilo bi dobro imati softverski alat koji bi mogao da pripremi studentske radove za ocenjivanje, kao i da popuni tabelu za unos poena neophodnim podacima (podaci o studentu i skala za ocenjivanje).

O problemu automatizacije diskutovalo se u nekolicini naučnih radova. Jedan od njih jeste „*Artificial intelligence, automation and work*“ [1]. U ovom radu autori su pažnju posvetili veštačkoj inteligenciji koja je iskorišćena za obavljanje poslova koji su automatizovani, a koji su prvobitno bili povereni ljudima.

„*The Benefits and Generic Procedure of Automating and Academic Student System in Primary and Secondary Schooles as an Impetus for Educational Technology*“ [2] primer je naučnog rada koji fokus stavlja na opis softvera razvijenog sa ciljem beleženja razlika između uspeha ostvarenog u osnovnim i srednjim školama, kao i razlike u uspehu učenika privatnih i državnih škola. Automatizacija je izvršena u domenu preuzimanja predatih učeničkih radova i njihovoj pripremi za ocenjivanje (nastavnik se prijavljuje na sistem i za preuzimanje mu postaju dostupni radovi koje treba da pregleda).

Glavna teza koja je potvrđena prilikom implementacije softvera razvijenih u svrhu detaljne analize izložene u prethodno navedenim radovima jeste da je ostavljen prostor za automatizaciju pojedinih kritičnih tački sistema, ali uz izvesnu dozu opreza kako se ne bi došlo u situaciju da prekomerna upotreba automatizacije izazove neželjene efekte.

3. SERVERSKA STRANA: RADNI OKVIR SPRING

Aplikacije pisane u programskom jeziku *Java* mogu se razvijati uz oslonac na radni okvir *Spring* čija je glavna karakteristika koncept primene inverzije kontrole (eng. *dependency injection*). Prednost ovog radnog okvira je to što enkapsulira module pomoću kojih je moguće izgenerisati inicijalni *Spring* projekat sa svim osnovnim podešavanjima neophodnim za dalji razvoj [1].

Tehnologija čijoj upotrebi danas pribegava velik broj programera koji rade na implementaciji veb orijentisanih softverskih rešenja, koju odlikuje jednostavnost u pogledu podešavanja konfiguracije, mogućnost rada sa lokalnim serverom i brzo dobijanje osnovne projektne strukture

buduće veb aplikacije, između ostalog ostavlja i prostor za manipulaciju zavisnostima, upotrebu predefinisanih ali i sopstvenih anotacija, asinhrono izvršavanje zadataka (`@Async`), kao i automatsko generisanje inicijalnih konfiguracionih fajlova.

3.1. Arhitektura SpringExtractor aplikacije za automatizaciju procesa pregledanja radova

Izgradnja ***SpringExtractor***-a oslanjala se na radni okvir *Spring* u kombinaciji sa *Maven* alatom koji pruža mogućnost jednostavnog uvezivanja sa već postojećim bibliotekama klasa.
Konvencija *Spring*-a nalaže podelu sastavnih delova *Spring* projekta po slojevima:

- sloj modela - sačinjen je od model klasa pomoću kojih je apstrahovan deo realnog sveta koji predstavlja predmet interesovanja aplikacije u razvoju, anotacija `@Entity`
- repozitorijum – interfejsi namenjeni pojednostavljenom procesu komunikacije sa skladištem podataka kojima aplikacija manipuliše, anotacija `@Repository`
- servisni sloj - predstavlja vezu između baze podataka i podržanih *backend* funkcionalnosti (interfejsi i klase koje ih implementiraju), anotacija `@Service`
- sloj kontrolera - sadrži aplikativnu (biznis) logiku sistema koji se implementira. Često se upotrebljava u kombinaciji sa *REST* arhitektonskim šablonom zasnovanim na principu postojanja jedinstvene putanje na kojoj se nalazi svaki od resursa, anotacija `@Controller`
- *DTO* – opcioni sloj koji predstavlja posrednika u komunikaciji između klijentske i serverske strane, sa ciljem izbegavanja direktne komunikacije

3.2 Uporedni prikaz strukture aplikacija PHPExtractor i SpringExtractor

Suštinska razlika između ***PHPExtractor*** i ***SpringExtractor*** aplikacija je u tome što je prvobitna verzija u okviru jednog projekta enkapsulirala i prezentacioni i upravljački sloj. Naredna verzija predstavljala je pokušaj relaksacije gradivnih komponenti aplikacije koje su raspoređene u dve nezavisne projektne celine – kreiran je zaseban projekat za prezentacioni sloj, kao i poseban projekat koji sadrži biznis logiku.

Glavni problem na koji se naišlo prilikom razvoja ***PHPExtractor***-a jeste složen postupak kreiranja komponenti *MVC* arhitekture. Naime, nedostatak *user-friendly* okruženja iziskivao je stalnu potrebu za radom sa terminalom. Da bi se napravio svaki od slojeva aplikacije, bilo je potrebno ručno kucati komande kojima se kreiraju klase određene namene. Nasuprot ovakvom načinu rada nalazi se integrisano razvojno okruženje *Eclipse* koje ostavlja prostor da se prilikom implementacije ***SpringExtractor***-a jednostavnim klikom na željenu opciju izgeneriše ono što je potrebno.

Nedostatak radnog okvira *Spring* sadržan je u činjenici da je *CRUD* operacije neophodno ručno napisati, dok je *Laravel* koncipiran tako da programeru olakšava posao tako što od njega zahteva samo da pozove odgovarajuću metodu. Problem bezbednosti *Laravel* uspešno prevazilazi zahvaljujući postojanju ugrađenog posrednika (eng. *middleware*) koji filtrira pristigle *HTTP* zahteve, dok *Spring* zahteva da zaštita veb aplikacije u razvoju bude ručno implementirana upotrebom *Spring security*-a. Najvažnija funkcionalnost – preuzimanje arhive studentskih radova dobijene od administratora i njeno raspakivanje – uspešno je implementirana u obe verzije aplikacije. Prva verzija aplikacije je u tu svrhu koristila eksterne dodatke (eng. *plugin*) koji zahtevaju posebnu instalaciju. *Spring* to izbegava zahvaljujući postojanju *maven* zavisnosti (navode se u konfiguracionom *pom.xml* fajlu). Nedostatak koji se pojavio prilikom rada sa *Spring* radnim okvirom odnosi se na to da se unapred mora znati tip podataka koji se prosleđuju putem mreže. Sa ovim problemom prethodna verzija aplikacije nije imala dodira zato što je postojala univerzalna promenljiva `$request` iz koje su preuzimani potrebni podaci i koji su zatim mapirani na odgovarajuće elemente.

4. BAZA PODATAKA

Pojam baze podataka objašnjava se kao prostor za skladištenje, organizaciju, čuvanje i pristup podacima. Najvažnija karakteristika svake baze podataka jeste očuvanje integriteta, konzistentnosti i neredundantnosti uskladištenih podataka.

4.1. MySQL relaciona baza podataka

MySQL je relaciona baza podataka koja obezbeđuje tabelarni prikaz podataka koji se u njoj nalaze. Prednost *MySQL*-a jeste direktno mapiranje iz objektnog u relacioni model podataka (uz posredstvo *Hibernate ORM*-a) [2]. Objašnjenje zašto *MySQL* ima široku primenu među veb aplikacijama sadržano je u činjenici da je reč o multiplatformskoj bazi podataka, što može da sarađuje sa velikim brojem programskih jezika, što može da radi sa ogromnom količinom podataka, što je besplatna, otvorenog koda i što za izražavanje upita koristi intuitivnost i jasnoću *SQL*-a.

4.2. MySQL baza podataka u kombinaciji sa PHPExtractor-om i SpringExtractor-om

U prvoj verziji aplikacije prilikom kreiranja svake klase modela trebalo je odmah kreirati i korespondirajuću šemu. Da bi šema bila kompletna trebalo je ručno popuniti polja koja će se naći u odgovarajućoj tabeli. Pored zadavanja imena poljima i dodeljivanja njihovih tipova, eksplicitno se morao navesti i naziv same šeme. Na kraju je potrebno bilo odraditi još i migraciju.

Naredna verzija aplikacije je optimalnija u pogledu rada sa bazom podataka jer je većina problema na koje se naišlo u *Laravel*-u prevaziđena upotrebom *Spring*-ovih anotacija. Pomoću `@Entity` anotacije u bazi podataka izgenerisaće se odgovarajuća tabela čiji naziv je naziv klase. Samo atributi klase modela ispred kojih je navedena `@Column` anotacija biće vidljivi u bazi. Sloj repozitorijuma (`@Repository`) predstavlja direktnu vezu između sloja modela i baze podataka. `JpaRepository` predstavlja *Spring*-ovu adaptaciju za rad sa bazom podataka – realizovan je tako što odabranu metodu transformiše u predefinisane / parametrizovane *SQL* upite.

Po pitanju baze podataka ispostavilo se da su oba pristupa ravnopravna u pogledu načina rukovanja sa podacima, transformacije iz objektnog u relacioni model i po pitanju održavanja konzistentnog stanja uskladištenih podataka.

5. KLIJENTSKA STRANA

SpringExtractor razvijen je uz oslonac na *Angular* frejmvork.

Razlog zbog kog je odabran baš taj radni okvir jeste to što sadrži velik broj biblioteka koje omogućavaju automatsko podešavanje i generisanje inicijalnih komponenti koje potom treba nadograditi u skladu sa zahtevima aplikacije.

5.1. Angular frejmvork

Angular je *TypeScript* jezik i predstavlja rešenje kom se pribegava kada se radi na razvoju modernih aplikacija složenih korisničkih zahteva.
Neke od pogodnosti koje nudi ovaj alat su: brzina kompajliranja, uvođenje *CLI*-ja, u osnovi se koristi samo jedna stranica koja se prikazuje korisniku na kojoj se smenjuju komponente od kojih je izgrađena (tzv. arhitektura zasnovana na komponentama), redukcija veličine koda (minimalan kod – maksimalan efekat) i informativnija obaveštenja o greškama [3].
Mane *Angular*-a su: nekompatibilnost pojedinih verzija koja uzrokuje potrebu za ponovnim pisanjem koda, zahtevan je za učenje zbog prevelike kompleksnosti njegovih sastavnih delova, nije pogodan za aplikacije koje zahtevaju pretragu veće količine podataka zato što u osnovi koristi *JavaScript* meta-tagove pomoću kojih se prikazuje sadržaj na stranici, teško je preći sa starije na noviju verziju (obrnuto ne važi) itd. [4].

5.2. Korisnički interfejs SpringExtractor-a

Za potrebe razvoja aplikacije ***SpringExtractor*** implementirane su sledeće komponente:

first-page (omogućava odabir uloge koju će korisnik koji pristupa sistemu imati u daljem radu: profesor/student),

home (registrovani korisnik u ulozi profesora ima mogućnost otpremanja arhive studentskih radova i bodovne skale za realizovani test),

finish (omogućava preuzimanje raspakovanih studentskih radova sa delimično popunjenom tabelom za unos ostvarenih bodova, otpremanje u međuvremenu potpuno popunjene *.xls* tabele i iniciranje njene konverzije u *.pdf* format),

student-home (komponenta koja služi za prikaz rezultata održanih testova u *.pdf* formatu),

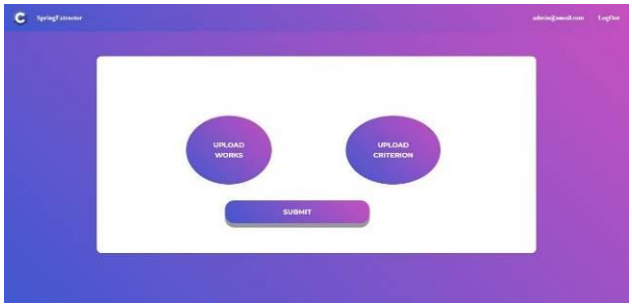
registration (izlistava formu koju je potrebno popuniti podacima neophodnim za registraciju korisnika) i

login (namenjena prethodno registrovanim korisnicima – očekuje unos korisničkih kredencijala za pristup naprednim funkcionalnostima). Na slici 1 predstavljen je korisnički interfejs ***SpringExtractor***-a.

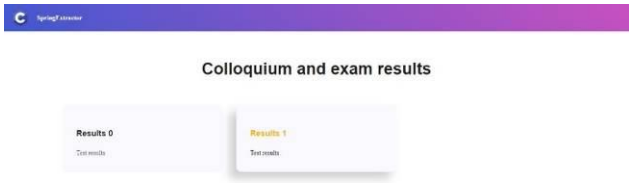
Na slici 2 može se videti izgled stranice za prikaz fajlova sa rezultatima testa.

Sastavni deo ***SpringExtractor*** aplikacije su: `home-service`, `finish-service`, `student-home-service`, `registration-service` i `login-service`.

Svaki od navedenih servisa implementira poslovnu logiku, tj. funkcionalnosti koje treba da se dese kao odgovor na akciju korisnika.



Slika 1. *Korisnički interfejs: home-page*



Slika 2. *Korisnički interfejs: student-home*

5.3. Komparacija prezentacionog sloja PHPExtractor-a i SpringExtractor-a

Konstatacija do koje se došlo nakon što su implementirane aplikacije ***PHPExtractor*** i

SpringExtractor jeste da je tehnologije korišćene pri razvoju ***PHPExtractor***-a optimalnije upotrebiti kada se radi na manjim aplikacijama koje nemaju potrebu da često dobavljaju podatke iz baze. Ako su zahtevi aplikacije takvi da podrazumevaju učestalo manipulisanje podacima uskladištenim u bazi, pametnije je odabrati koncept frejmvorka i razdvajanja klijentske i serverske strane (pristup koji je korišćen u implementaciji ***SpringExtractor***-a).

6. NAPREDNE FUNKCIONALNOSTI SPRINGEXTRACTOR-A

Napredna verzija aplikacije za automatizaciju procesa pregledanja radova enkapsulirala je implementaciju pojedinih mehanizama zaštite (validacija podataka, uvođenje uloga i oporavak u slučaju zaboravljene lozinke) i naprednih funkcionalnosti (registracija korisnika; prijava na sistem i odjava sa sistema; automatsko popunjavanje zaglavlja tabele adaptivnom bodovnom skalom; preuzimanje, parsiranje i ponovno arhiviranje kolekcije studentskih radova; konverzija izvornog *.xls* formata fajla sa rezultatima u *.pdf* format; preuzimanje konvertovanog fajla i mogućnost njegove dalje distribucije).

6.1. Registracija

Kako bi se novi korisnik registrovao na sistem potrebno je da popuni formu traženim podacima, nakon čega se upućuje zahtev ka serverskoj strani (poziva se metoda `registration`). Uspešnoj registraciji prethodi validacija korisničkog unosa i provera da li u bazi već postoji korisnik sa unetom *email* adresom.

6.2. Prijava na sistem i odjava sa sistema

Nakon što je korisnik izvršio registraciju na sistem i aktivirao svoj korisnički nalog posetom aktivacionog linka iz *mail*-a, omogućena mu je prijava za šta je neophodan unos korisničkih kredencijala. Klikom na dugme `Login` poziva se istoimena *backend* metoda.
Odjava sa sistema vrši se odabirom opcije `LogOut`. Na serverskoj strani ova metoda implementirana je tako da ima zadatak da ukloni aktivnog korisnika iz sesije.

6.3. Zaboravljena lozinka

U slučaju da korisnik dođe u situaciju da je zaboravio lozinku za pristup sistemu, može zahtevati da mu se automatski izgeneriše nova. Ova funkcionalnost radi na sledećem principu: u bazi se prvo pronađe korisnik sa unetom *email* adresom (validira se korisnički unos na klijentskoj strani), potom se pozove metoda koja generiše nasumičan string dužine deset karaktera sastavljen od alfanumerika (u tu svrhu iskorišćena je `RandomStringUtils` klasa uz prethodno dodavanje odgovarajuće *maven* zavisnosti u *pom.xml* fajl), da bi se zatim dobijeni string postavio kao vrednost nove lozinke. Kako bi korisnik znao sa kojom se lozinkom nadalje može prijaviti, sistem mu automatski šalje *email* sa tom informacijom.

6.4. Adaptivno bodovanje

Zapakovana kolekcija studentskih radova u sebi sadrži podarhive u kojima su smešteni radovi. Kada se ova arhiva otpremi, potrebno je odabrati još i kriterijum po kom će biti izvršeno bodovanje nakon čega je dozvoljeno kliknuti na dugme `submit`. ***SpringExtractor*** omogućio je dinamičko unošenje kriterijuma bodovanja: zaglavlje tabele popunjava se isparsiranim sadržajem fajla koji predstavlja bodovne stavke.

6.5. Preuzimanje i konverzija popunjene .xls tabele u .pdf format

Kada se korisnik nađe na *finish* stranici može odabrati opciju `download works` čiji je zadatak preuzimanje prethodno programski zapakovanog foldera koji sadrži raspakovane studentske radove i tabelu sa popunjenim zaglavljem i podacima o studentima. Proširena verzija aplikacije podržava mogućnost konverzije formata fajla sa rezultatima. Da bi bila pozvana metoda koja vrši tu funkciju korisnik treba da otpremi željeni fajl u *.xls* formatu i da potom klikne na dugme `convert excel to pdf`.

7. ZAKLJUČAK

U radu je predstavljen pokušaj implementacije softverskog rešenja koje bi automatizovalo proces pregledanja studentskih radova.

SpringExtractor uspešno je automatizovao prvu fazu procesa pregledanja radova – raspakivanje otpremljene arhive sa radovima i pripremu tabele za ocenjivanje. Nedostatak prve verzije aplikacije koji se odnosi na to da kandidati moraju ispoštovati određenu formu prilikom unosa ličnih podataka prevaziđen je tako što je implementirano parsiranje fajla sa podacima o svim studentima dobijenog od administratora.

Rešeno je ograničenje koje se ticalo vrste testa i bodovne skale za isti – dozvoljeno je da korisnik u ulozi profesora otpremi adaptivnu skalu za bodovanje. Uvedena su prava pristupa tako da samo registrovani korisnici mogu obavljati sve operacije kojima aplikacija raspolaže. Neregistrovani korisnici imaju samo *readonly* pravo pristupa nad dokumentom sa rezultatima testa.
Kada se bude radilo na implementaciji naredne verzije rešenja akcenat će biti stavljen na automatizaciju druge faze, tj. na sam proces pregledanja radova. Plan je da se pokuša napraviti skripta koja bi prolazila kroz priloženo rešenje svakog studenta i analizirala programski kod na osnovu kog bi generisala predlog ostvarenih bodova za odgovarajuću bodovnu stavku.
Ako bi se došlo do stadijuma da je polazna problematika u potpunosti uspešno automatizovana, to bi značilo da su stvoreni preduslovi za proširenje oblasti primene softverskih rešenja čiji je akcenat na procesu automatizacije.

8. LITERATURA

[1] Spring Framework, https://spring.io/projects/spring-framework, pristupljeno: 06.09.2020.
[2] Baza podataka – definicija, vrste i modeli, https://kompjuteras.com/baze-podataka/?script=lat, pristupljeno: 06.09.2020.
[3] Angular: Best Use Cases and Reasones To Opt For This Tool, https://yalantis.com/blog/when-to-use-angular/, pristupljeno: 07.09.2020.
[4] Angular tutorials point, https://www.tutorialspoint.com/angular7/angular7_tutorial.pdf, pristupljeno: 07.09.2020.

Kratka biografija:

Aleksandar Vasiljević rođen je 02.02.1996. u Prizrenu. Završio je osnovnu školu „Mirko Jovanović“ u Kragujevcu. Nakon završene osnovne škole upisuje srednju tehničku školu „Prva tehnička škola“ u Kragujevcu. 2015. godine upisuje „Fakultet tehničkih nauka“ u Novom Sadu, smer *Računarstvo i automatika*. Školske 2017/18. godine opredeljuje se za usmerenje *Primenjene računarske nauke i informatika*. Školske 2018/19. godine upisuje modul *Internet i elektronsko poslovanje*. Zvanje *diplomirani inženjer elektrotehnike i računarstva* dobija 30.08.2019. Školske 2019/20. godine upisuje master akademske studije, smer *Primenjene računarske nauke i informatika – Elektronsko poslovanje*. Zvanje *saradnik u nastavi* na Departmanu za računarstvo i automatiku na Fakultetu tehničkih nauka u Novom Sadu dobija 28.10.2019. kontakt: alexva02@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE48Radakovic

PRIMENA BLOKČEJN TEHNOLOGIJA U SISTEMIMA ZA MEĐUNARODNE FINANSIJSKE TRANSAKCIJE

APPLICATION OF BLOCKCHAIN TECHNOLOGIES IN SYSTEMS FOR INTERNATIONAL FINANCIAL TRANSACTIONS

Danijel Radaković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu su opisana tri blokčejn sistema za međunarodne transakcije: Ripple, Stellar i Libra, kao i njihove prednosti u odnosu na tradicionalne sisteme plaćanja. Predstavljeno je njihovo međusobno poređenje u izvršavanju međunarodnih transakcija i analiza svakog sistema pojedinačno.*
Ključne reči: *distribuirana glavna knjiga, blokčejn, Libra, Ripple, Stellar, SWIFT gpi, međunarodne transakcije.*

Abstract – *The paper describes tree blockchain systems for international transactions: Ripple, Stellar, and Libra, as well as their their advantages over traditional payment systems. Their mutual comparison in the execution of international transactions and the analysis of each system individually are presented.*
Keywords: *distributed ledger, blokchain, Libra, Ripple, Stellar, SWIFT gpi, international financial transactions.*

1. UVOD

Teško je u današnje vreme zamisliti život bez novca jer je toliko integrisan u naše društvo i svetsku ekonomiju. Polovinu svake transakcije čini novac, dok drugu polovinu čini usluga ili materijalno dobro za koje se novac razmenjuje.

Skoro sve današnje transakcije uključuju posrednike za njihovo uspešno izvršavanje poput banaka, kompanija za kreditne kartice i kompanija koje pružaju usluge plaćanja na globalnom nivou.

Ovi posrednici vode računa o transakcijama, odnosno kako se novac prebacuje sa jednog računa na drugi i tu uslugu dobro naplaćuju. Usluge su još skuplje ukoliko je u pitanju međunarodna transakcija koja se takođe duže čeka na njeno izvršavanje.

Bitcoin je svojim dizajnom uspeo da reši dva velika problem u sistemu plaćanja: pružanje usluga prenosa novca za deo svetske populacije koji ne pripadaju bankarskom sistemu, kao i mogućnost izvršavanja transkacija sa niskim nadoknadama po celom svetu bez potrebe za centralnim autoritetom [1]. Međutim, države žirom sveta ne mogu da prihvate *Bitcoin* ni kao valutu ni kao platformu za plaćanje jer bito to jer bi to dovelo do narušavanje njihovih monetarnih politika, suvereniteta i kontrolu nad finansijksim tokovima.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dušan Gajić, docent.

Iako *Bitcoin* nudi dosta slobode i pogodnosti svojim korisnicima mreže, postoje određeni problemi. Jedan od problema jeste skaliranje mreže prvenstveno zbog toga što mreža koristi *proof-of-work* [1] konsenzus algoritam na kojem se troši puno energije za rudarenje novog bloka. Drugi problem je velika volatilnost kriptovalute koja otežava da se usvoji kao uobičajeno sredstvo plaćanja.

U ovom radu dat je opis tri blokčejn sistema: *Ripple*, *Stellar* i *Libra*, kao i njihove prednosti u odnosu na tradicionalne sisteme plaćanja. Cilj ovog rada jeste analizirati prethodno navedene sisteme u izvršavanju međunarodnih finansijskih transakcija i uočiti prednosti i mane svakog sistema pojedinačno.

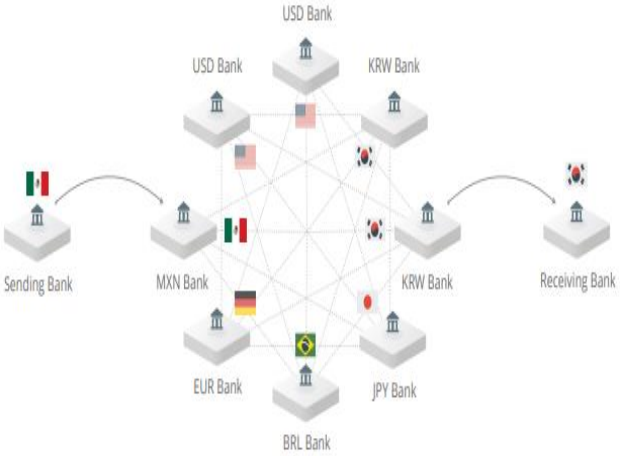
2. PROBLEMI SA TRADICIONALNIM SISTEMIMA ZA PLAĆANJA

Međunarodne finansijske transakcije su uvek predstavljale problem bankama i ostalim finansijskim institucijama. Izvršavanje međunarodne finansijske transakcije je dugotrajan proces, postoji veliki rizik da se transakcija neće uspešno izvršiti i nadoknade za njihovo izvršavanje su velike. Glavni uzrok ovih nedostataka jeste to što banke koriste tradicionalne sisteme za međunarodna plaćanja koji su stari i po nekoliko decenija. Tradicionalna infrastrukura se sastoji od gomilu centralizovanih mreža pri čemu svaka mreža poseduje svoja pravila i procedure. Da situacija bude još gora, nisu sve banke podjenako umrežene i ne pripadaju sve banke istoj mreži. Ovo znači da postoji velika veravatnoća da ne postoji direkta linija između dve banke kako bi se prebacile sredstva sa jednog računa na drugi. U tom slučaju da bi se sredstva prebacila sa jednog računa na drugi, transfer sredstava mora da prođe kroz više posrednih banaka. Zbog toga je izvršavanje međunarodne finansijske transakcije dugotrajan i skup proces jer svaka banka u lancu zahteva određenu količinu vremena za obradu transakcije kao i proviziju za njenu obradu.

Određene međunarodne finansijske transakcije zahtevaju konverziju valuta, pri čemu se proces dodatno komplikuje i postaje skuplji. Ovakve transakcije su skuplje jer zahtevaju višestruku konverziju valuta. Ovo dovodi do toga da se uopšte ne isplati izvršavati međunarodne finansijske transakcije sa malim iznosom s obzirom na to da su troškovi transfera sredstava veliki.

Najveći nedostatak tradicionalnih sistema za globalna plaćanja je u tome što ne postoji globalna mreža sa univerzalnim skupom pravila koja povezuje sve banke i ostale finansijske institucije. Svaki put kada se vrši međunarodna transakcija mora da se pronađe putanja između

računa pošaljioca i primalaca, a te putanje se razlikuje od slučaja do slučaja, Slika 1. Trenutno najbolje rešenje zasnovano na tradicionalnim sistemima za plaćanje je *SWIFT* kod koga izvršavanje transakcija može da potraje danima. Glavni problem predstavlja to što se koristi finansijska inftrastruktura stara nekoliko decenija [3][4]. Upravo je ovo problem koji blokčejn sistemi poput *Ripple*, *Stellar* i *Libra* pokušavaju da reše.



Slika 1. Mreža različitih vrsta banaka [2]

3. RIPPLE

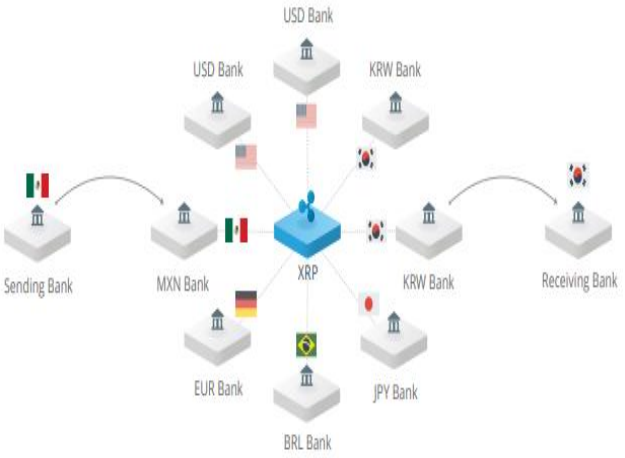
Ripple omogućava transfer skoro bilo koje valute bilo gde u svetu u nekoliko sekundi. Cena izvršavanja tih transakcija je veoma mala jer *Ripple* omogućava direktan transfer sredstava između banaka u realnom vremenu bez potreba za posrednim (korespodentnim) bankama.

Može se reći da *Ripple* nudi rešenje za probleme koji se odnose međunarodne finansijske transakcije. To rešenje predstavlja ***Ripple Transaction Protocol*** (***RTXP***) koji obuhvata skup pravila koji omogućuje brz i efikasan prenos vrednosti na globalnom nivou. *RTXP* se koristi u ***RippleNet*** i ***XRP Ledger*** (***XRPL***) blokčejn mrežama.

XRPL predstavlja javnu blokčejn mrežu bez centralnog autoriteta kreiranu od strane *Ripple*-a. Mreža se sastoji od validatora (računara) koji su geografski distribuirani i održavaju distribuiranu glavnu knjigu u konzistetnom stanju. ***XRP*** predstavlja nativnu kriptovalutu *XRPL*-a koja se koristi kao posrednik razmene između različitih fiat valuta, kriptovaluta i ostalih oblika vrednosti. Razmena vrednosti se vrši pomoću ***Decentralized Exchange*** (***DEX***) mehanizma. Jedna od bitnih karakteristika mreže jeste brza i jeftina razmena sredstava (vrednosti) s obzirom na to da se koristi algoritam (eng. ***pathfinding***) koji pronalazi najkraću i najjeftiniju putanju (uz pomoć *DEX*-a) koja će dovesti do njihove razmene [5]. Problem dvostruke potrošnje se na *XRPL* mreži rešava korišćenjem ***Ripple Protocol Consensus Algorithm*** (***RPCA***).

RippleNet predstavlja privatnu mrežu koja se zasniva na *XRPL*-u i predstavlja vlasništvo *Ripple* kompanije koja je namenjana da poveže banke i ostale finansijske institucije širom sveta kako bi se omogućila brza i jeftina globalna plaćanja, slika 2.

Kompanije koje žele da pristupe *RippleNet* mreži mogu da to učine pomoću posebnih banaka i finasijskih institucija autorizovanih od strane *Ripple* kompanije koji se još nazivaju *gateways*.



Slika 2. *RippleNet* mreža [2]

XRP se pored konverzija valuta koristi i za plaćanje transakcionih troškova. Kreirano je 100 milijardi tokena i prema *RTXP*-u ne mogu se dodatno kreirati novi tokeni. Najmanja jedinica *XRP* tokena je *drop* koji predstavlja milioniti deo *XRP*-a. Svaki račun na *XRPL*-u mora u rezervi da poseduje minimum 20 *XRP* tokena.

Ovo ograničenje je uspostavljeno kako bi se sprečilo spamovanje na mreži.

Transakcioni troškovi trajno uništavaju *XRP* tokene i jednom uništeni *XRP* tokeni se više ne mogu povratiti. Uništavanje *XRP* tokena dovodi do njegove deflacije odnosno do povećanja njegove vrednosti s obzirom na to da ga vremenom ima sve manje u opticaju.

RPCA omogućava uspostavljanje konsenzusa nad manjom mrežom čvorova i propagira novo stanje sveta koje je validirano od strane konsenzus algoritma nad celom mrežom. Ovo pokazuje da je moguće ostvariti konzistetnost mreže uz minimalnu komunikaciju među čvorovima.

Direktna posledica ovog pristupa su mala kašnjenja prilikom uspostavljanje konsenzusa i velika skalabilnost kada je u pitanju broj čvorova u mreži s obzirom da se konsenzus ne uspostavlja nad svim čvorovima u mreži. Svaki validator održava svoju jedinstvenu listu validatora (*Unique Node List* - *UNL*) kojima veruje. *RPCA* omogućava 20% malicioznih učesnika u određenom *UNL*-u, odnosno čim se 80% servera iz određenog *UNL*-a usaglase o novom stanju sveta konsenzus je postignut.

4. STELLAR

Stellar predstavlja javnu blokčejn mrežu osnovanu 2015. godine koja je namenjena za izvršavanje međunarodnih finansijskih transakcija i razmenu različite oblike novca. *Stellar* se po svojim ciljevima, nameni, performansama i dizajnu ima sličnosti sa *Ripple*-om, a ova sličnost je proizašla iz toga što je jedan od osnivača *Stellar*-a, *Jed McCeled*, ujedno i osnivač *Ripple*-a. Razvoj projekta prati ***Stellar Development Foundation*** (***SDF***) čiji su svi projekti otvorenog koda uključujući *Stellar* blokčejn mrežu i alate koji omogućavaju efikasno korišćenje funkcionalnosti mreže.

Stellar, kao i *Ripple*, može da vodi evidenciju i razmenu bilo kog oblika imovine u vidu tokena.

Tokeni su digitalni oblik imovine iz realnog sveta, a proces kreiranja digitalne imovine se naziva tokenizacija. Tokeni na *Stellar* mreži uglavnom predstavljaju različite fiat valute i kriptovalute, ali se tehnički mogu predstaviti hartije od vrednosti, akcije i stvari iz realnog sveta poput zlata, srebra itd. Tokeni se izdaju od strane posebnih korisnika u mreži koji se nazivaju ***anchors*** (*Ripple*-ov ekvivalent je *gateway*). Bilo ko može da postane *anchor* ali uglavnom *anchors* predstavljaju kompanije i finansijske institucije. Prilikom izdavanja tokena određene fiat valute, *anchors* su u obavezi da konverziju tokena i fiat valute rade u odnosu 1:1 i da poseduje rezerve fiat valute koja je ekvivaletna količini izdatih tokena. Ovaj tip tokena (imovine) se naziva ***stablecoin***.

Stellar koristi svoju nativnu kriptovalutu koja se zove ***lumen*** (***XLM***) koja se takođe kao i *XRP* koristi kao sredstvo za konverziju različitih fiat valuta i kriptovaluta. Takođe, poseduje algoritam (*path payment*) koji pronalazi nakrajću i najjeftiniju putanju do razmene sredstava koristeći *DEX* mehanizam. *XLM* tokeni se koristi i za plaćanje troškova transakcija koji se prenose na poseban račun (*fee pool*) koji ne pripada nikome i *XLM* tokeni sa tog računa se ne mogu više koristiti. Svaki račun na mreži mora da poseduje minimum 1 *XLM* kako bi se sprečilo spamovanje mreže, a najmanja jedinica *XLM*-a je *stroop* koji predstavlja deseto milioniti deo *XLM*-a. Kao što je slučaj sa *XRP*-om, i *XLM* tokeni su unapred kreirani i ne mogu se dodatno kreirati. Kreirano je 50 milijardi.

Problem dvostruke potrošnje se na *Stellar* mreži rešava pomoću ***Stellar Consensus Protocol***-a (***SCP***) koji se zasniva na ***federated Byzantine agreement*** (***FBA***) algoritmu [10]. *FBA* algoritam je inicijalno zamišljen da reši probleme koji poseduju tradicionalni konsenzus algoritmi zasnovani na rešavanju problema vizantijskih generala. Glavni problem tih algoritama jeste što zahtevaju zatvoren sistem validatora u kojem su validatori unapred poznati i zahteva da dve trećine validatora budu validni. Među tim algoritmima se ubrajaju *SBFT* i *HotStuff* [11].

5. LIBRA

Libra je kriptovaluta kreirana od strane *Facebook*-a i ima za cilj da omogući ljudima širom sveta globalnu valutu i sredstvo plaćanja. *Libra* se zasniva na bezbednom, skalabilnom i pouzdanom blokčejnu. Korisnici mogu da razmenjuju određene fiat valute za *Libra* novčiće (u daljem tekstu *≈LBR*) i pomoću tih novčića mogu da kupuju materijalna dobara i usluge na *Facebook*-ovoj platformi.

Libra se sastoji od sledećih komponenti:

- Zasniva se na bezbednom, skalabilnom i pouzdanom blokčejnu - ***Libra Blockchain***.
- *Stablecoin* koji je baziran na rezervi valuta i imovine niske volatilnosti - ***Libra Reserve***.
- Rukovodi je nezavisna zajednica - ***Libra Association***.
- Koristi varijantu *Byzantine Fault Tolerant Consensus* algoritma- ***LibraBFT (LBFT)*** *consensus mechanism*.
- Koristi pametne ugovore (*smart contracts*) za izvršavanje transakcija pomoću ***Move*** programskog jezika.

Libra Zajednicu (***Libra Association***) predstavlja neprofitnu zajednicu koja se bavi nadgledanjem razvoja kriptovalute, upravaljanjem rezervama nad kojima se zasniva vrednost kriptovalute, kao i donošenjem pravila upravljanja blokčejn mrežom. Sedište *Libra* Zajednice nalaziće se u Švajcarskoj, u gradu Ženeva. Zemlja je pažljivo odabrana jer je poznata po svom neutralnom statusu i jakoj podršci za finansijske inovacije uključujući i blokčejn tehnologiju.

Libra Rezerve (***Libra Reserve***) će se skladištiti rezerve dominatnih fiat valuta na osnovu kojih će se zasnivati razlitčiti *stablecoin*-i. Neki primeri *stablecoin*-a zasnovani na jednoj fiat valuti su: *LibraUSD* (*≈USD*), *LibraEUR* (*≈EUR*), *LibraGBP* (*≈GBP*), *LibraSGD* (*≈SGD*). Svaki *stablecoin* biće pokriven rezervama u gotovini i kratkoročnim hartijama od vrednosti. *≈LBR* će biti digitalna kompozicija *stablecoin*-a zasnovani na jedno fiat valuti koji su dostupni na *Libra* mreži. Na ovaj način mogu se dodavati novi *stablecoin*-i koji se zasnivaju na jednoj fiat valutu ako se za to ukaže potreba. Takođe, još veći benefit predstavlja to što se na ovaj način mogu integrisati kriptovalute koje su kreirane od strane Narodnih banaka (*central bank digital currencies – CBDC*).

LibraBFT pripada klasi klasisčnih *Byzantine Fault Tolerance* (*BFT*) konsenzus algoritama. Zasniva se na *HotStuff* konsenzus algoritmu sa određenim izmenama i proširenjima. *HotStuff* koristi određene osobine *Practical Byzantine Fault Tolerance* (*PBFT*) algoritma [12]. *LibraBFT* može da toleriše jednu trećinu malicioznih validatora.

Facebook je razvio novi programski jezik nazvan ***Move*** koji je dizajniran tako da omogući lagan način za pisanje bezbednih i fleksibilnih pamethin ugovora koji će se izvršavati na *Libra* mreži [13]. Ključna osobina *Move* programksog jezika jeste mogućnost definisanja proizvoljnih resursa koji predstavalju imovinu u digitalnom obliku koje poseduju iste osobine kao i fizička imovina: resurs ima samo jednog vlasnika, može da se potroši samo jednom i kreiranje novih resursa je ograničena.

6. ANALIZA I POREĐENJE SISTEMA

Ripple je kreator *RippleNet* i *XRPL* mreža. Prednosti koje ove mreže nude u odnosu na tradicionanle sisteme za globalna plaćanja su brzo, jeftino i efikasno izvršavanje međunarodnih finansijskih transakcija. Obe mreže nude *DEX* mehanizam za razmenu sredstava i algoritam koji pronalazi najkraću putanju razmene sredstava. Takođe, obe mreže koriste *ILP* koji im omogućava da se lako integrišu u ostale sisteme [6]. Pored ovih prednosti, postoje određeni nedostaci sa *RPCA*-om i *XRP* tokenima.

RPCA ima nedostatak što se zahteva mnogo veći procenat preklapanja validatora između čvorova (oko 90%) kako bi se sprečilo grananje u mreži nego što je inicijalno zamišljeno (20%) [7][8][9]. Međutim, nedostatak svakako ubija inicijalnu ideju protokola koji je zamišljen da se konsezus može uspostaviti nad jednim skupom validatora koji nisu u jakoj vezi sa ostalim validatorima u mreži. Upravo iz ovog razloga *Ripple* radi na novom konsezus protokolu *Cobalt* koji treba da smanji poklapanje na validatora na 60%.

Najveći nedostatak *XRP* tokena je u tome što se velika većina tokena nalazi u vlasništvu *Ripple* kompanije pri

čemu se sva kontrola prepuša u ruke *Ripple* kompanije. Zbog ove činjenice, kao i zatvorenost *RippleNet* mreže čine sistem centralizovanim iako je mreža decentralizovana jer *Ripple* kompanija ima veliki uticaj u mreži. Još jedan nedostatak koji se odnosi na *XRP* tokene jeste da nisu obavezni da se koriste u *RippleNet* mreži. Iako je *XRP* namenjen da ubrza i pojeftini izrvšavanje transakcija, njegova volatilnost sprečava da se usvoji od strane banaka. Vrednost *XRP*-a zavisi prvenstveno od toga koliko banaka je spremno da ga usvoji.

Sve prednosti koje poseduje *Ripple* poseduje i *Stellar*, a pored toga eliminiše *Ripple*-ove nedostatke. Eliminiše nedostatke koje se odnose na *XRP* jer *Stellar* mreža ne može da funkcioniše bez *XLM*-a, ali je i dalje volatilan kao *XRP*. Takođe, eliminiše nedostatke koje se odnose ne zatvorenost i centralizovanost *Ripple* sistema jer *Stellar* predstavlja javnu i decentralizovanu mrežu, a i sam *SCP* deluje dosta obećavajuće. Nedostatak koji je prisutan u *Stellar* mreži jeste činjenica da su fiat valute predstavljene u vidu *stablecoin*-a.

Problem sa *stablecoin*-om koji se zasniva na rezervama fiat valute jeste u tome što se nikad ne može dokazati da li *anchor* poseduje ekvivalentu sumu fiat valuta u realnom svetu npr. na računu određene banke jer su te informacije nedostupne na mreži. Sa kriptovalutama to ne predstavlja problem jer su računi javni i može se bez problema utvrditi kolike su rezerve kriptovalute.

Takođe, potencijalni problem predstavlja ukoliko računi gde se čuvaju rezerve fiat valute ili kriptovalute budu kompromitovani (npr. budu uhakovani, zamrznuti ili ugašeni), gubi se vrednost njihovih tokena na *Stellar* mreži.

Na sve to se postavlja pitanje da li će se rezerve uspešno očuvati na duži vremenski period, jer istorija je pokazala da to nije moguće. Jedan od najpoznatijih slučajeva jeste kada je američki dolar izgubio zlatnu podlogu [14]. Ovi nedostaci se takođe odnose i na *Libra* sistem s obzirom na to da i *Libra* koristi *stablecoin*-e.

Ripple i *Stellar* mogu da tokenizuju bilo koju stvar iz realnog sveta bez potrebe za pametnim ugovorima jer im je tokenizacija ugrađena u sam protokol. Ovaj pristup kreiranja imovine na mreži je bezbedniji i jednostavniji u odnosu na upotrebu pametnih ugovora. Pametni ugovori su manje bezbedni jer snose rizike od postojanje bagova. Međutim pametni ugovori iako snose rizike od bagova nude mnogo veći spektar manipulacije imovine na mreži. *Libra* sa druge strane, tokenizaciju sprovodi kreiranjem *stablecoin*-a i korišćenjem pametnih ugovora pisanih u *Move* programskom jeziku.

Libra mreža asocira na hibrid između *Etherium* i *RippleNet* mreže. Od *Etherium*-a su preuzeti pametni ugovori, dok od *RippleNet*-a preuzeti zatvorenost mreže i brzo izvršavanje transakcija. Jedna od bitnih prednost *Libra* sistema je *Move* progreamski jezik koji je inicijalno dizajniran da bude bezbedan i omogući strogu manipulaciju resursa u mreži. *Libra* pored platforme za globalna plaćanja može da postane i dobra platforma za *Decentralized Finance* (*DeFi*) jer podržava pisanje pametnih ugovora i brzo izvršavanje transakcija. *DeFi* se trenutno uglavnom zasniva na *Etherium* platformi [15]. Kao što je već napomenuto, nedostatak pametnih ugovora se snosešnje rizika postajanja bagova, a rešavanje ovog problem nije adresirano u dizajnu *Libra* sistema.

Libra predstavlja dobar primer na kome se ispoljavaju negativne osobine centralizovnih sistema. S obzirom na to da je mreža privatna i u vlasništvu *Libra* Zajednice sve kazne za kršenje propisa i zakona izvršene na *Libra* mreži će se upućivati *Libra* Zajednici. Među bitnijim propisima koje moraju da se ispoštuju na mreži su sprečavanje pranje novca, finansiranje terorizma i zaštita korisnika na mreži.

Mehanizmi za sprečavanje pranje novca i finansiranje terorizma već postoje na *XRPL*, *RippleNet* i *Stellar* mrežama. Međutim, *XRPL* i *Stellar* su javne mreže, pa iako se desi neko kršenje zakona, kazne nemaju kome da se upućuju jer niko nije vlasnik mreže. *RippleNet* sa druge strane iako je privatna mreža, korisnici su banke i finansijske institucije koje su u obavezi da ispoštuje spomenute zakone. Implementacija ovih mehanizama možda i ne predstavlja toliki problem s obzirom da su već implementirani u spomenutim mrežama.

Veći problem predstavlja to što se na *Libra* mreži moraju ispoštovati zakoni o zaštiti potrošača. Na primer, ukoliko neki korisnik izgubi određenu sumu sredstava u prevari ili bude na neki drugi način oštećen, očekuje se da mu se ta sredstva nadohnade i *Libra* Zajednica je u zakonskoj obavezi da to sprovede.

Problem je u tome što se *Libra* zasniva na blokčejn tehnologiji koja sprečava menjanje i poništavanje već izvršenih transakcija.

7. ZAKLJUČAK

Ripple predstavlja prvu platformu za globalna plaćanja namenjena bankama koja treba da zameni tradicionalne sisteme plaćanja. Upravo iz ovog razloga je *RippleNet* mreža implementirana nad već postojećom bankarskom infrastrukturom. *Ripple* kao svoje korisnike cilja banke i finansijske institucije, prosečna osoba nema značajnu korist. *Stellar*, za razliku od *Ripple*-a je prevenstveno namenjen prosečnim osobama, malim i srednjim predužećima.

Oba blokčejn sistema su izmenili tradicionalni sistem plaćanja i nude bolje uslove za poslovanje od *SWIFT*-a. Prednosti koje nude ovi sistemi su uvideli i ostale kompanije poput *JP Morgan*-a koji planira da plasira svoju kriptovalutu koji će im definitivno biti konkurencija u budućnosti [16].

Libra sa druge strane se takmiči sa popularnijim blokčejn sistemima *Bitcoin*-om i *Etherium*-om, ali i definitifno i sa *Ripple*-om i *Stellar*-om s obzirom da cilja određenu grupu njihovih korisnika. *Libra* želi da postane platforma za globalna plaćanja poput *Bitcoin*-a, a takođe i platforma za pravljenje *DeFi* sistema poput *Etherium*-a. Takođe, *Facebook* otvara svoju platformu za elektronsku kupovinu robe *Facebook Shoops* i time se direknto takmiči sa *Amazon*-om i *eBay*-om [17].

Realno je očekivati da će se u nekom momentu plaćanja na *Facebook Shoops* platformi vršiti preko *Libra* sistema kao i da će ostali *Facebook*-ovi proizvodi (*WhatsApp*, *Instagram*, *Messenger*) biti integrisani sa *Libra* sistemom. Ukoliko bi se *Libra* mreža uspešno plasirala, definitivno će imati veliki uticaj na globalni finansijski sektor zbog velikog broja *Facebook*-ovih korisnika.

LITERATURA

[1] Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2009. https://bitcoin.org/bitcoin.pdf

[2] Ripple, *Building Network Effects on Ripple*, poslednji pristup 22.09.2020. https://ripple.com/files/ripple_vision.pdf

[3] Tianyi Qiu, Ruidong Zhang, Yuan Gao, *Ripple vs. SWIFT: Transforming Cross Border Remittance Using Blockchain Technology*, 2018.

[4] Ravishankar Achanta, *Cross-border money transfer using blockchain - enabled by Big Data*, 2018.

[5] Ripple, *XRP Ledger*, poslednji pristup 22.09.2020. https://xrpl.org/index.html

[6] Stefan Thomas, Evan Schwartz, *A Protocol for Interledger Payments* https://interledger.org/interledger.pdf

[7] David Schwartz, Noah Youngs, Arthur Britto, *The Ripple Protocol Consensus Algorithm*, 2015. https://ripple.com/files/ripple_consensus_whitepaper.pdf

[8] Brad Chase Ethan, MacBrough, *Analysis of the XRP Ledger Consensus Protocol*, februar 2018. https://arxiv.org/pdf/1802.07242.pdf

[9] David Schwartz, Noah Youngs, Arthur Britto, *The Ripple Protocol Consensus Algorithm*, 2015. https://ripple.com/files/ripple_consensus_whitepaper.pdf

[10] David Mazieres, *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus*, februar 2016. https://www.stellar.org/papers/stellar-consensus-protocol

[11] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb, *Fast and secure global payments with Stellar*, oktobar 2019. https://www.stellar.org/papers/fast-and-secure-global-payments-with-stellar

[12] Libra, *State Machine Replication in the Libra Blockchain*, maj 2020. https://developers.libra.org/docs/assets/papers/libra-consensus-state-machine-replication-in-the-libra-blockchain/2020-05-26.pdf

[13] Libra, *Move: A Language With Programmable Resources*, maj 2020. https://developers.libra.org/docs/assets/papers/libra-move-a-language-with-programmable-resources/2020-05-26.pdf

[14] Investopedia, *What is Gold Standard*, poslednji pristup 22.09.2020. https://www.investopedia.com/ask/answers/09/gold-standard.asp

[15] *DeFi Pulse*, poslednji pristup 22.09.2020. https://defipulse.com/

[16] *JP Morgan Digital Coin*, poslednji pristup 22.09.2020. https://www.jpmorgan.com/global/news/digital-coin-payments

[17] Arstechica, *Facebook Takes on Amayzon with Online Shopping Venture*, poslednji pristup 22.09.2020. https://arstechnica.com/tech-policy/2020/05/facebook-takes-on-amazon-with-online-shopping-venture/

Kratka biografija:

Danijel Radaković rođen je 21.04. 1996. godine u Vršcu. Osnovnu školu „Mladost“ u Vršcu završio je 2011. godine. Srednju tehničku školu „Školski centar Nikola Tesla“ u Vršcu, smer ekonomskog tehničara završio je 2015. godine. Iste godine upisao se na Fakultet tehničkih nauka, odsek Softversko inženjerstvo i informacione tehnologije.

Diplomirao je 2019. godine na Fakultetu tehničkih nauka odbranivši rad na temu „Implementacija infrastrukture javnih ključeva“.

Master studije je upisao 2019. godine na Fakultetu tehničkih nauka, odsek Softversko inženjerstvo i informacione tehnologije, smer Računarstvo visokih performansi.

U realizaciji Zbornika radova Fakulteta tehničkih nauka u toku 2019. godine učestvovali su sledeći recenzenti:

Aco Antić
Aleksandar Erdeljan
Aleksandar Ristić
Bato Kamberović
Biljana Njegovan
Bogdan Kuzmanović
Bojan Batinić
Bojan Lalić
Bojan Tepavčević
Bojana Beronja
Branislav Atlagić
Branislav Nerandžić
Branislava Novaković
Branka Nakomčić
Branko Milosavljević
Branko Škorić
Damir Đaković
Danijela Lalić
Darko Čapko
Darko Marčetić
Darko Reba
Dejan Ubavin
Dejana Nedučin
Dragan Ivanović
Dragan Ivetić
Dragan Jovanović
Dragan Kukolj
Dragan Mrkšić
Dragan Pejić
Dragan Šešlija
Dragana Bajić
Dragana Konstantinović
Dragana Šarac
Dragana Štrbac
Dragoljub Šević
Dubravka Bojanić
Dušan Dobromirov
Dušan Gvozdenac
Dušan Kovačević
Dušan Uzelac
Duško Bekut
Đorđe Ćosić
Đorđe Lađinović
Đorđe Obradović
Đorđe Vukelić
Đula Fabian
Đura Oros
Đurđica Stojanović
Filip Kulić
Goran Sladić
Goran Švenda
Gordana Milosavljević
Gordana Ostojić
Igor Budak
Igor Dejanović
Igor Karlović
Ivan Beker
Ivana Katić
Ivana Kovačić
Ivana Miškeljin
Jasmina Dražić
Jelena Atanacković Jeličić
Jelena Borocki
Jelena Kiurski
Jelena Radonić
Jovan Petrović
Jovanka Pantović
Ksenija Hiel
Laslo Nađ
Lazar Kovačević
Leposava Grubić Nešić
Livija Cvetičanin
Ljiljana Vukajlov
Ljiljana Cvetković
Ljubica Duđak
Maja Turk Sekulić
Marko Todorov
Marko Vekić
Maša Bukurov
Matija Stipić
Milan Rapajić
Milan Simeunović
Milan Trifković
Milan Trivunić
Milan Vidaković
Milena Krklješ
Milica Kostreš
Milica Miličić
Mijodrag Milošević
Milovan Lazarević
Miodrag Hadžistević
Miodrag Zuković
Mirjana Damnjanović
Mirjana Malešev
Mirjana Radeka
Mirko Borisov
Miro Govedarica
Miroslav Hajduković
Miroslav Popović
Mitar Jocanović
Mladen Kovačević
Mladen Radišić
Nemanja Stanisavljević
Nemanja Sremčev
Nikola Đurić
Nikola Jorgovanović
Nikola Radaković
Ninoslav Zuber
Ognjen Lužanin
Pavel Kovač
Peđa Atanasković
Petar Malešev
Predrag Šiđanin
Radivoje Dinulović
Radovan Štulić
Relja Strezoski
Slavica Mitrović
Slavko Đurić
Slobodan Dudić
Slobodan Krnjetin
Slobodan Morača
Sonja Ristić
Srđan Kolaković
Srđan Popov
Srđan Vukmirović
Staniša Dautović
Stevan Gostojić
Stevan Milisavljević
Stevan Stankovski
Strahil Gušavac
Svetlana Nikoličić
Tanja Kočetov
Tatjana Lončar - Turukalo
Uroš Nedeljković
Valentina Basarić
Velimir Čongradec
Veran Vasić
Veselin Perović
Vladimir Katić
Vladimir Strezoski
Vlado Delić
Vlastimir Radonjanin
Vuk Bogdanović
Zdravko Tešić
Zoran Anišić
Zoran Brujić
Zoran Jeličić
Zoran Mitrović
Zoran Papić
Željen Trpovski
Željko Jakšić