UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

Nenad N. Petrović

RAZVOJ INTELIGENTNIH SISTEMA VOĐEN DOMENSKIM ONTOLOGIJAMA

DOKTORSKA DISERTACIJA

Niš, 2024.

UNIVERSITY OF NIŠ
FACULTY OF ELECTRONIC ENGINEERING

Nenad N. Petrović

DOMAIN-ONTOLOGY DRIVEN DESIGN OF INTELLIGENT SYSTEMS

DOCTORAL DISSERTATION

Niš, 2024.

Podaci o doktorskoj disertaciji

Mentor:	Prof. dr Milorad B. Tošić, redovni profesor, Univerzitet u Nišu, Elektronski fakultet
Naslov:	Razvoj inteligentnih sistema vođen domenskim ontologijama
Rezime:	Od zabave, bankarstva, poslovanja preko industrije i proizvodnje, pa sve do zdravstva, procesi iz raznovrsnih domena se oslanjaju na sisteme u čijoj su osnovi najrazličitiji digitalni uređaji i inteligentne računarske tehonologije. Međutim, usled raznovrsnosti i velikog broja uređaja, upravljanje, kontrola i koordinacija u okviru ovih sistema postaju prilično kompleksni, zahtevaju dosta truda i vremena, pogotovu ako se vrše manuelno. Pored toga, neophodna su i ekspertska znanja o uređajima i procesima, što povećava kognitivno opterećenje i dodatno otežava rukovanje. Međutim, automatsko generisanje koda je jedan od pravaca koji se mogu kiskoristiti za prevazilaženje navedenih problema. U ovoj disertaciji, predstavljen je radni okvir zasnovan na ontologijama i semantičkim tehnologijama za automatsko generisanje koda, čiji je cilj da olakša razvoj i rukovanje kompleksnim sistemima u različitim oblastima primene. Ontologije se u ovom slučaju koriste da predstave znanje o različitim aspektima inteligentnih sistema, kao što su domenske konceptualizacije, aspekti strukture, ali, sa druge strane – ponašanje i koordinacija sistema. Na osnovu njih, u sinergiji sa mehanizmimia automatskog generisanja koda, razvijeni su i odgovarajući alati, koji omogućavaju korišćenje intuitivnih domenski-specifičnih notacija sa ciljem upravljanja i razvoja inteligentnih sistema. Efektivnost razvijenog rešenja je ilustrovana na osnovu eksperimenata u okviru studija slučaja iz različitih oblasti: računarstvo u magli, robotika, blokčejn sistemi i proširena stvarnost. Što se evaluacije tiče, razmatrani su različiti aspekti, poput vremena izvršenja, poboljšanja performansi i postignutog ubrzanja u odnosu na tradicionalni pristup. Prema postignutim rezultatima, ovaj pristup značajno ubrzava rukovanje, ali i sam razvoj domenski-specifičnih inteligentnih sistema i aplikacija. Osim toga, na osnovu znanja predstavljenog ontologijama, ovakav pristup omogućava dodatne pogodnosti, poput adaptivnog ponašanja i verifikacije relevantnih aspekata, što je ilustrovano prikazanim studijama slučaja.
Naučna oblast:	Elektrotehničko i računarsko inženjerstvo

Naučna disciplina:	Predstavljanje znanja
Ključne reči:	Domenski-specifični jezici, Inteligentni sistemi, Ontologije, Semantičke tehnologije
UDK:	111:004.774.2:004.89
CERIF klasifikacija:	T120, Sistemski inženjering, računarska tehnologija
Tip licence Kreativne zajednice:	**CC BY-NC-ND**

Data on Doctoral Dissertation

Doctoral Supervisor:	Dr Milorad B. Tošić, full professor, University of Nis, Faculty of Electronic Engineering
Title:	Domain ontology-driven design of intelligent systems
Abstract:	Systems based on digital devices and intelligent computer technology are of utmost importance across various domains starting from entertainment, through business and banking to industry, manufacturing and healthcare. However, due to their increasing complexity, heterogeneity and rising number, the operations related to management, control and coordination within these systems become significantly more demanding, requiring much time and effort when done entirely manually. Apart from that, it is often the case that detailed domain expertise about the devices as well as underlying processes is necessary, which increases the overall cognitive load while handling such systems. Automated code generation is one of the most promising directions to overcome the mentioned challgenges. In this dissertation, an ontology-based framework is introduced for automated code generation, with aim to make the development and management of complex intelligent systems more convenient for adoption within heterogeneous domains. Ontologies are leveraged for representation of knowledge covering crucial aspects of intelligent systems, including domain conceptualization, topology and structure together with system behavior. Starting from semantic knowledge representation with respect to ontologies in synergy with automated code generation, auxiliary tools which enable the adoption of intuitive domain-specific notation are created as outcome with goal of automated management and development of intelligent systems. Effectivenss of the proposed approach is illustrated based on experiments within realistic case studies from various domains of usage: fog computing, experimental robotics, blockchain systems and augmented reality. When it comes to evaluation, many relevant points of view are considered, such as execution time, performance improvement or speed-up compared to more traditional approaches. According to the achieved results, the proposed approach significantly speeds up handling, development and maintenance of intelligent systems and applications. Apart from that, based on semantic knowledge representation, this approach exhibits many additional benefits, such as context-aware adaptive system behavior, coordination and verification of relevant elements, as illustrated by presented case studies.
Scientific Field:	Electrical and computer engineering

Scientific Discipline:	Knowledge representation
Key Words:	Domain-specific languages, Intelligent systems, Ontology, Semantic technology
UDC:	111:004.774.2:004.89
CERIF Classification:	T120 Systems engineering, computer technology

ZAHVALNICA

Neizmerna zahvalnost prvenstveno članovima moje laboratorije, ali i svim mojim saradnicima i koautorima načunih publikacija na prenešenom iskustvu i iscrpnim diskusijama. Prvenstveno se zahvaljujem mom mentoru dr Miloradu Tošiću, redovnom profesoru, koji mi je dao priliku na samom početku akademske karijere da učestvujem na RAWFIE-SCOR naučnom projektu, tokom čijih aktivnosti sam udario čvrste temelje daljim istraživanjima iz oblasti ontologija i semantičkih sistema. Njegova dalja podrška mojih istraživačkih interesovanja tokom dosadašnje karijere na Elektronskom fakultetu rezultovala je nagrađivanim radovima, ali i uspešno realizovanim projektima. Posebno se zahvaljujem dr Valentini Nejković (vanredni profesor), koja je takođe uticala na moje usmeravanje istraživanja iz ove oblasti i sarađivala na publikacijama. Takođe, zahvaljujem se i svim ostalim članovima kolektiva Elektronskog fakulteta koji su podržali moje učešće na mnogobrojnim nacionalnim i internacionalnim konferencijama i doprineli da moji radovi budu zapaženi.

Zahvaljujem se i profesorki Elisabetti di Nitto, koja mi je pružila jedinstvenu priliku da tokom master studija na univerzitetu Politecnico di Milano, kao privremeni član njene laboratorije u saradnji sa svetskim istraživačima iz oblasti modelima-vođenog softverskog inženjerstva realizujem prototip automatskog generatora koda za upravljanje računarskim infrastrukturama u oblaku polazeći od modela raspoređivanja, što je bila jedna od inspiracija za ovu temu.

Osim toga, dugujem zahvalnost i izvanrednim istraživačima na međunarodnom nivou, koje sam upoznao online za vreme COVID-19 pandemije. Oni su mnogo puta potvrdili validnost mojih ideja kroz desetine zajedničkih naučnih radova, usmeravali me, a na taj način indirektno uticali da te ideje dosegnu viši nivo zrelosti i da ih što bolje uobličim u disertaciju, pri čemu se među njima ističu prof. dr Issam Al-Azzoni (Ujedinjeni Arapski Emirati) i Vasja Roblek (Slovenija).

SADRŽAJ

1 UVOD

Informacioni sistemi su postali neizostavano prisutni kao podrška različitim procesima u maltene svim delatnostima i domenima primene – od sporta i industrije zabave, preko saobraćaja i poslovanja, pa do zdravstva. Sa druge strane, inteligencija kao pojam se često definiše kao mogućnost adaptacije na promene u okruženju, kako u psihologiji i kognitivnim naukama, tako i u prirodno-tehničkim naukama [1]. Adaptivna inteligencija je identifikovana kao jedan od ključnih faktora opstanka u neizvesnim okolnostima [2]. Na važnost i ulogu adaptacije organizma u uslovima promenljivog spoljašnjeg okruženja tokom njegovog životnog ciklusa ukazuju i pojedine studije iz oblasti biologije i evolucije [3], što postaje inspiracija za inovativne arhitekture informacionih sistema [4].

U poslednje dve decenije sve se više teži ka razvoju takozvanih *inteligentih informacionih sistema* koji su sposobni da na osnovu prikupljenih podataka o korisnicima, spoljašnjim akterima, događajima i drugim objektima od značaja, ali i njihovim međusobnim vezama, dođu do određenih zaključaka, sa ciljem da se prilagode trenutnom kontekstu [5]. Što se odgovora na promene tiče, to može biti prilagođeni prikaz određenih informacija, generisanje komandi za aktivaciju aktuatora ili nekog uređaja u kućnom ili industrijskom okruženju, ali i kompletnih programa ili aplikacija, sa ciljem rešavanja problema specifičnog za neki domen. Između ostalog, lakoća proširljivosti, nadogradnje i prilagođavanja inteligentnih sistema novim zahtevima i potrebama predstavlja jedan od ključnih faktora njegove upotrebljivosti na duže staze [6]. Trud, vreme, pa samim tim i cena razvoja novih funkcionalnosti ili modifikacija postojećih u kasnijim fazama životnog ciklusa softverskog proizvoda mogu biti vrlo visoki.

Sa druge strane, pojam generalne kolektivne inteligencije [7] figuriše kao skup harmonijskih, koordinisanih akcija jedinki u sistemu, sa ciljem prilagođavanja na novonastale okolnosti ili postizanja zadatog cilja [7][8]. Mnogi rezultati iz različitih oblasti, pogotovu u računarstvu, ukazuju da se kompleksni problemi mogu efikasnije rešiti organizovanim delovanjem skupa jedinki nego kada to obavlja pojedinac. Međutim, ovaj aspekat nije dovoljno zastupljen, niti iskorišćen u većini postojećih inteligentnih informacionih sistema i predstavlja otvoreno istraživačko pitanje [9]. Pored toga, smatra se da je uzrok nepostojanja rešenja za pojedine probleme upravo nedostatak neophodnih mehanizama koordinacije relevantnih entiteta [7][10]. Što se konkretno informacionih sistema tiče, koordinacija velikog broja uređaja ili softverskih jedinica predstavlja veliki izazov, s obzirom na sve veću brojnost i raznolikost [10][11]. Ne samo da se ovi uređaji razlikuju po skupu mogućnosti koje pružaju, već često i za

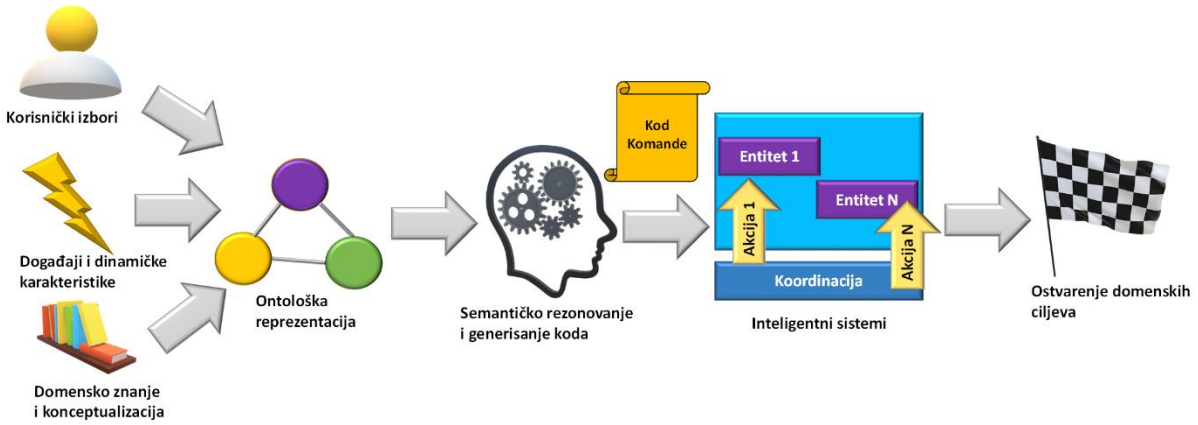
podatke iste vrste koriste različite formate, što dovodi do problema interoperabilnosti [11]. Konačno, aktivnosti upravljanja, održavanja i prilagođavanja ovakvih sistema postaju sve kompleksnije, tako da je kriva učenja postaje sve strmija. Zbog toga, upotreba ovakvih sistema od strane domenskih eksperata postaje previše zahtevna usled same kompleksnosti infrastrukture na koju se oslanja, što dovodi do negativnog efekta da glavni fokus nije na samom rešavanju domenskog zadatka, već na prevazilaženju drugih problema [9][11].

Ova disertacija se bavi istraživanjem primene ontologija i semantičkih tehnologija u sinergiji sa mehanizmima automatskog generisanja koda, sa svrhom da se prevaziđu prethodno navedeni problemi i izazovi. Iako tehnike i metode mašinskog učenja postaju sve prisutnije kao oslonac inteligentnih sistema, semantičke tehnologije se mogu primeniti komplementarno i to sa različitim ciljevima [12][13]. Jedan od slučajeva jeste postizanje objašnjivosti prediktivnih modela, što može pomoći prilikom donošenja odluka u kritičnim primenama, a to predstavlja korak ka takozvanoj objašnjivoj veštačkoj inteligenciji, Explainable Artifiical Intelligence (XAI) [12][13][14]. Sa druge strane, direktna ugradnja eksperstskog iskustva u vidu semantičkih pravila zaključivanja nad podacima anotiranih upotrebom ontologija štedi vreme, potreban trud i ne zahteva ogromne količine podataka za obuku inteligentnih mehanizma, kao što je to slučaj kod modela mašinskoog učenja [8]. Konačno, semantička reprezentacije pruža mogućnost jednostavne verifikacije instanci sistema izvršenjem upita, što olakšava razvoj i potencijalno povećava kvalitet generisanog koda.

1.1 Ciljevi naučnog istraživanja doktorske disertacije

U ovoj disertaciji, istražena je primena domenskog znanja, koje obuhvata i statičke i dinamičke aspekte, sa ciljem olakšavanja razvoja, upotrebe i prilagođavanja kompeksnih inteligentnih informacionih sistema. Kao sredstva za ostvarivanje ovog cilja se koriste ontologije u sinergiji sa mehanizmima semantičkog zaključivanja i automatskog generisanja koda. Dok se ontologije koriste za reprezentaciju domenskog znanja, pravilima semantičkog zaključivanja se detektuje odgovarajući kontekst, a odgovor na nastale okolnosti generiše kao skup komandi ili izvršni kod oslanjajući se na generatorski program. Osim toga, fokus ovog pristupa jeste na olakšavanju rukovanjem kompleksnim sistemima od strane domenskih eksperata, pri čemu je cilj rešavanje domenskog problema. Dok se većina rešenja u ovoj oblasti fokusira uglavnom na topološke karakteristike sistema, ova disertacija stavlja poseban akcenat na ponašanje u toku izvršenja, ali i koordinaciju većeg broja jedinica unutar sistema radi

efikasnijeg ostvarivanja cilja u nekom domenu primene. Ilustracija cilja disertacije na visokom, konceptualnom nivou je ilustrovana na **Slici 1.1**.



Slika 1.1 Cilj istraživanja na konceptualnom nivou

1.2 Doprinosi naučnog istraživanja

Kao ishod istraživanja, ova doktorska disertacija daje sledeće doprinose:

- Radni okvir visokog nivoa za razvoj kompleksnih inteligentnih sistema na osnovu ontologija namenjen domesnkim ekspertima;
- Skup ontologija visokog nivoa koje opisuju statičke, dinamičke i aspekte koordinacije inteligentnih sistema, a služe kao osnova za dobijanje domenski-specifičnih ontologija;
- Implementacija metodologije za izvođenje domenskih ontologija upotrebom principa analogija na osnovu ontologija visokog nivoa;
- Algoritmi za automatsko generisanje izvršivog koda inteligentnih sistema na osnovu predloženih ontologija visokog nivoa;
- Vizuelni alat kao podrška domenskim ekspertima prilikom razvoja sistema;
- Ontologije za predstavljanje domenski-specifičnih problema iz prikazanih studija slučaja;
- Predlog inovativnih domenski-specifičnih notacija za predstavljenih studije slučaja;
- Semantičke skupovi podataka nastali u toku eksperimenata za date studije slučaja;
- Mehanizmi semantičke analize i verifikacije domenski-specifičnog koda koji se oslanjaju na primenu ontologške reprezentacije i SPARQL upita;
- Pomoćna sredstva korišćena od strane algoritama za automatsko genersianje koda, poput modela linearne optimizacije za optimalno raspoređivanje kontejnerskih aplikacija i prediktivnih modula mašinskog učenja;

- Alati i okruženja za modelovanje nastali automatskim generisanjem koda neophodni za prezentaciju studija slučaja;
- Aplikacije namenjene krajnjim korisnicima automatski generisane u kontekstu studija slučaja (kao što su napredni korisnički interfejsi proširene stvarnosti);
- Ostali rezultati generisanja koda na osnovu ontologija koji se mogu dalje nezavisno upotrebiti (alat za raspoređivanje kontejenera kod računarstva u magli);
- Pregled kvalitativnih i kvantitativnih rezultata ostvarenih za izabrane studije slučaja.

1.3 Primenjene naučne metode

Pri izradi doktorske disertacije je korišćen skup istaživačkih metoda [11], koje treba da omoguće realizaciju planiranih ciljeva – kreiranje radnog ovkira za ontološko predstavljanje domenski-specifičnih aspekata sa svrhom automatskog generisanja koda, koja pomaže u realizaciji inteligentnih sistema.

Osnovni metod istraživanja se zasniva na studijama slučaja iz raznolikih domena. Za svaku od predstavljenih studija slučaja, izvedeni su eksperimenti, u kojima se primenjuje predloženi pristup za automatsko generisanje koda zasnovan na ontologijama, a ostvareni benefiti razmatraju iz dve perspektive: 1) kvalitativni rezultati – šta predloženi radni okvir omogućava u posmatranim studijama slučaja, koje nove mogućnosti donosi, i 2) kvantitativni rezultati – kakve performanse ima predloženi pristup. Za kvantitativne rezultate biće primenjena komprativna metoda i to poređenjem dobijenih vrednosti (konkretnije, izmerenih vremena izvršenja za generisanje koda po koracima) u odnosu na ekvivalentne manuelne procedure, ali i naspram drugih sličnih rešenja, ukoliko postoje.

Metode modelovanja, analize i deskripcije će biti korišćenje za razvoj i implementaciju ontologškog radnog okvira i odgovarajućih algoritama za generisanje koda na osnovu njih. Osim toga, metod prikupljanja podataka će biti korišćen za izvođenje eksperimenata u okviru pojedinih studija slučaja. Prikupljeni podaci biće označeni u skladu sa predloženim domenskim ontologijama.

1.4 Pregled strukture disertacije

Ova doktorska disertacija je organizovana u šest poglavlja.

Prvo poglavlje prvenstveno uvodi u problematiku i predmet istraživanja, sa fokusom na motivaciju i praznine koje postoje kod aktuelnih rešenja u ovoj oblasti. Osim toga, razmotreni

su istraživački ciljevi, sredsta za realizaciju cilja i naučne metode, zajedno sa istraživačkim doprinosima, koji su nastali kao ishod.

Drugo poglavlje daje zajedničku teoretsku, ali i tehnološku osnovu za prikazane studije slučaja i njihovu implementaciju, zajedno sa pregledom relevantnih rešenja i metoda, koje se primenjuju u ovim oblastima. Dalje, ovo poglavlje se deli na sedam sekcija. Prva od njih se bavi domenski-specifičnim jezicima i notacijama, kao i njihovim vezama sa ontologijama. Druga sekcija se fokusira na ključne koncepte i terminologiju vezanu za ontologije. Treća sekcija se u ovom poglavlju bavi odgovarajućim semantičkim tehnologijama koje su korišćene u ovoj disertaciji, sa akcentom na RDF i SPARQL jezik za upite nad semantičkim bazama znanja. Četvrta sekcija uvodi koncept viših ontologija i razmatra njihovu primenu u kreiranju domenskih ontologija. Nakon toga, peta sekcija pruža pregled relevantnih operacija i procesa nad ontologijama, koje omogućavaju njihovo bolje iskorišćavanje. Tema šeste sekcije jeste metodologija za automatizovanje razvoja domenskih ontologija korišćenjem analogija i kompetentnih pitanja. Konačno, sedma sekcija drugog poglavlja se fokusira na metode i tehnike za automatsko generisanje koda polazeći od semantičke reprezentacije znanja oslanjajući se na ontologije.

Treće poglavlje prikazuje zajedničku implementacionu osnovu za realizaciju predstavljenih studija slučaja. Prva sekcija se bavi formalnom definicijom problema. Druga sekcija prikazuje arhitekturu implementiranog radnog okvira na visokom, konceptualnom nivou. Treća sekcija opisuje ontološki radni okvir, koji se sastoji od ontologije višeg nivoa, na osnovu koje se izvode domenski-specifične. Četvrta sekcija se bavi implementacijom Java API-ja , koji se koristi kao zajednička osnova za semantičko rezonovanje i pomoćno sredstvo prilikom automatsko generisanje koda. Peta sekcija opisuje korišćene algoritme za automatsko generisanje koda. Šesta sekcija prikazuje primenu generisanog okruženja i grafičkih alata.

Četvrto poglavlje razmatra studije slučaj koje ilustruju primenu razvijenog ontološkog radnog okvira i generisanih alata u različitim domenima. Prva studija slučaja se bavi problemom inteligentnog, automatizovanog upravljanja infrastrukturama koje se oslanjaju na mehanizme računarstva u magli. Druga studija slučaja se bavi problemom koordinisanog upravljanja pokretnih robota zasnovanom na ROS-u, sa ciljem nadzora u industrijskom okruženju. Treća studija slučaja predstavlja primenu predloženog pristupa u domenu proširene stvarnosti, sa ciljem automatskog generisanja naprednih korisničkih interfejsa raznih namena, kao što su daljinsko upravljanje pokretnih robota, ali i muzičko izvođenje. Četvrta studija slučaja razmatra primernu prethodno predloženog radnoh okvira u domenu prialgodljivih pametnih elektirčnih mreža iskorišćavanjem IoT uređaja. Osim toga, ova studija slučaja obuhvata i mehanizme

optimalne trgovine električnom energijom između prosumera (čvorovi u elekričnoj mreži koji su istovremeno potrošači, ali i prooizvođači) oslanjajući se na blokčejn tehnologije i pametne ugovore za obavljanje transakcija. Konačno, peta sekcija ovog poglavlja se bavi pomoćnim alatima i metodama, koje figurišu u prikazanim stduijama slučaja kao pomoćno sredstvo prilikom generisanja koda. Među ovim tehnikama, tu je linearna optimizacija, upotrebljena za rešavanje probema alokacije softverskih komponenti i oblikovanje SDN saobraćaja kod računarstva u magli, ali i kod optimalne distribucije električne energije u pametnoj mreži. Dalje, obuhvaćeno je i nadgledano mašinsko učenje za predikcije posmatranjem označenih podataka, na osnovu kojih se vrši odlučivanje. Od primenjenih tehnika mašinskog učenja, u ovom radu se koriste: 1) klasifikacija - za detekciju anomalija kod računarstva u magli i nestabilnosti u pametnim mrežama, i 2) regresija – za predikciju potrošnje električne energije. Konačno, u ovoj sekciji se razmatraju i osnovni pojmovi vezani za blokčejn i pametne ugovore – obzbeđuju mehanizme transakcija za razmenu električne energije.

Peto poglavlje se bavi rezimiranjem rezultata dobijenim u studijama slučaja, polazeći od jednostvanijih eksperimenata ka složenijem koji integriše sve njih. Pored toga što su različiti aspekti kvantitativno upoređivani sa tradicionalnim pristupom gde se operacije obavljaju manuelno, razmatra se i pozicioniranje predstavljenog radnog okvira u poređenju sa sličnim ili relevantnim rešenjima. Sa druge strane, izdvajaju se i kvalitativno, benefiti i pogodnosti koje predloženi pristup ima u odnosu na postojeća rešenja. Osim toga, razmotreni su nedostaci i ograničenja predstavljenog rešenja.

Na kraju, šesto poglavje daje zaključke o realizovanom radnom okviru za generisanje koda sa ciljem implementacije inteligentnih sistema u razlilitim domenima, uzimajući u obzir predstavljene studije slučaja i ostvarene rezultate. Konačno, predloženi su i potencijalni pravci u budućem istraživanjima na ovu temu, zajedno sa daljim planovima.

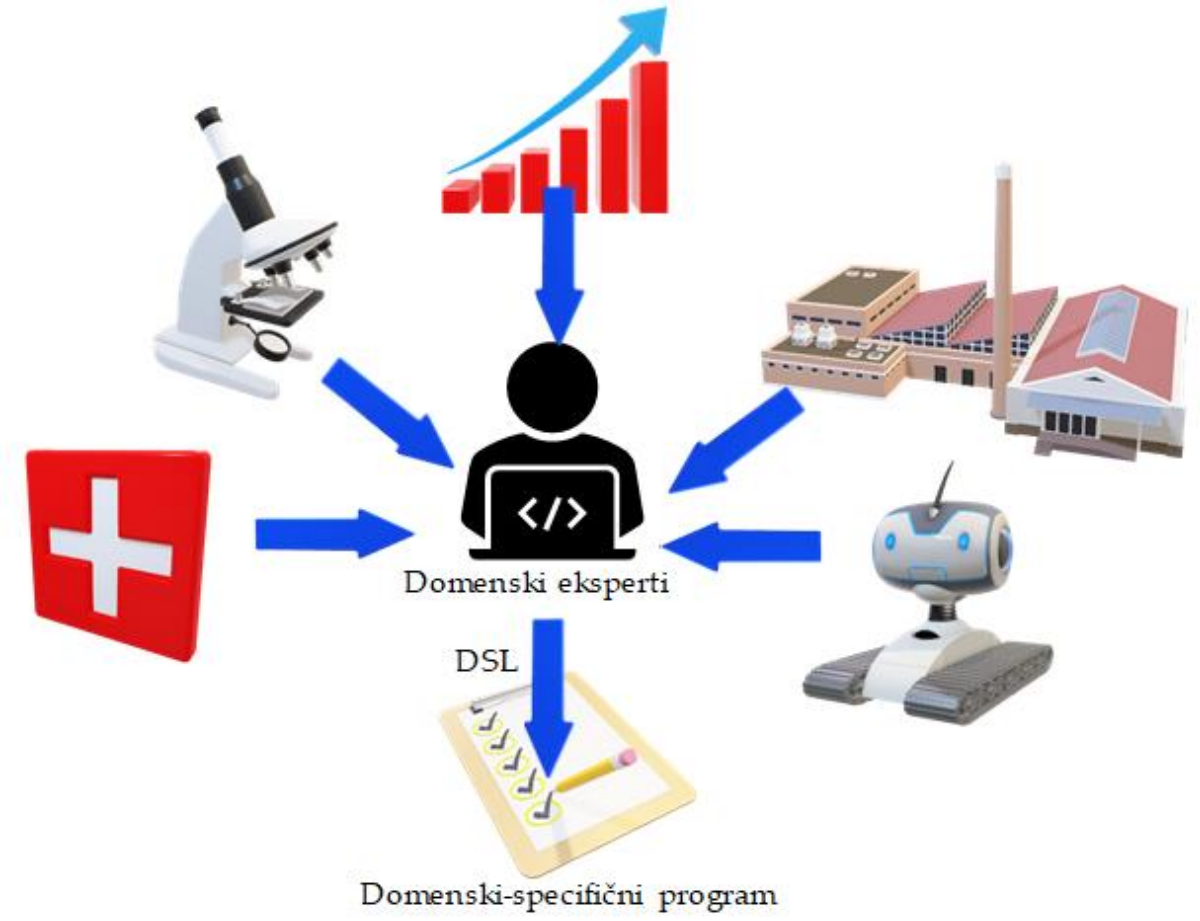
2 TEORETSKA OSNOVA I SRODNA ISTRAŽIVANJA

U ovom poglavlju, predstavljena je konceptualno-teoretska podloga predloženog pristupa za razvoj inteligentnih sistema koji se oslanja na domenske ontologije u sinergiji sa automatskim generisanjem koda. Osim toga, dat je i prikaz tehnologija na koje se oslanja implementacija predstavljenih eksperimenata u okviru studija slučaja. Dalje, u okviru svake od sekcija, dat je pregled karakterističnih rešenja, metoda ili ostalih relevantnih tehnologija za razumevanje pristupa prikazanog u ovoj disertaciji. Prva sekcija se bavi domenski-specifičnim notacijama i njihovim primenama, sa osvrtom na sinergiju sa ontologijama. Druga sekcija daje uvod u osnovne termine vezane za ontologije, zajedno sa pregledom karakterističnih ontologija u različitim domenima, ali i u softverskom inženjerstvu, između ostalog. Osim toga, prikazane su različite primene i uloge ontologija u ovoj oblasti. Dalje, treća sekcija ovog poglavlja predstavlja tehnologije i servise korišćene za njihovu implementaciju – RDF za skladištenje semantičkih podataka, SPARQL za upite, a TaaSOR kao pomoćnu biblioteku za programatski interfejs u Java programskom jeziku. Četvrta sekcija uvodi termin viših ontologija, koje su od značaja za prikazanu metodologiju, jer definišu generalne, apstraktne koncepte, čijim se konkretizovanjem dobijaju domenske ontologije. Posle toga, peta sekcija se bavi pomoćnim operacijama nad ontologijama, koje omogućavaju njihovo bolje iskorišćavanje. Šesta sekcija prezentuje pristup za automatizovan razvoj domenskih ontologija oslanjajući se na analogije i kompetentna pitanja. Konačno, sedma sekcija ovog poglavlja razmatra metode, tehnike i radne okvire za automatsko generisanje koda na osnovu ontologija.

2.1 Domenski-specifični jezici i notacije

Domenski-specifični programski jezici (*Domain-Specific Language* – DSL) se koriste da reše probleme iz oblasti pojedinih domena, pri čemu je fokus na lakšem ostvarivanju željenih ciljeva nego u slučaju kada se oslanjamo na programske jezike opšte namene za te iste zadatke [15][16]. DSL su svoju primenu pronašli u najrazličitijim oblastima, počev od poslovanja i ekonomije, preko robotike pa do upravljanja računarskim infrastrukturama [9]. Međutim, skup problema čije rešavanje omogućavaju je značajno manji od skupa problema koji se rešavaju programskim jezicima opšte namene, kao što su C++, Java i Python. Njihova efektivnost je ograničena isključivo na ciljeve relevantne za domen od interesa i generalnost im je dosta manja. Između ostalog, domenski-specifični jezici često obuhvataju, pored tekstualnih, i

intuitivne vizuelne notacije, koje značajno mogu olakšati njihovu primenu u kompleksnim scenarijima, čak i u slučaju kada korisnici nemaju inženjersko i tehničko predznanje vezano za programiranje [9][15][16]][17]. Na ovaj način, omogućeno je lakše rešavanje složenih problema iz specifičnih ekspertskih domena (ilustracija data na **Slici 2.1**), a sa druge strane i drastično je redukovano potrebno vreme, s obzirom na manje strmu krivu učenja, ali i oslanjanje na mehanizme automatskog generisanja koda. Prema tome, zbog navedenih prednosti, domenski-specifični jezici postaju široko zastupljeni i istraživani, pogotovu tokom poslednje decenije.



Slika 2.1 Primena domenski-specifičnih jezika u različitim oblastima

Generalno, razvoj domenski-specifičnih jezika ima sledeće faze [17][18]: 1) analiza, 2) projektovanje, 3) implementacija i 4) dostavljanje rešenja domenskim ekspertima. Razvoj domenski-specifičnih notacija zahteva detaljno poznavanje domena od interesa i definiciju ključnih koncepata, što je deo domenske analize [18][19]. Rezultat domenske analize je domenski model, za koji se očekuje da sadrži elemente koji se mogu upotrebiti više puta sa ciljem rešavanja domenskih problema iz iste kategorije. Tokom domenske analize, od izuzetnog

je značaja utvrditi vokabular domena, izdvojiti ključne koncepte koji figurišu, a sa druge strane, uočiti koji elementi su zajednički, a koji varijabilini prilikom rešavanja različitih instanci problema. Sve ovo utiče na izgled dobijene notacije i dostupnost podesivih svojstava samih elemenata. Domenska analiza je kritična faza, jer predstavlja osnovu za sve ostale, a sa druge strane, najmanje je dostupne literature na temu smernica i metodologije. Iako se koriste i formalne i neformalne metode, ove druge se često izbegavaju zbog kompleksnosti, a pored toga ni one uglavnom ne pružaju jasne instrukcije kako se proizvodi ove faze dalje koriste za projektovanje programskog jezika. Pokazano je da upotreba ontologija može da pomogne u prevazilaženju većine problema, prisutnih i kod jedne i kod druge grupe metoda [17][18]. U ovakvom scenariju, ontologije se koriste kao sredstvo za definisanje vokabulara nekog domena – objekata od interesa, njihovih veza i svojstava, ali i produkata nastalih kao ishod procesa u tom domenu, sa druge strane [20]. Ontologije se smatraju dobrim izborom za ovu primenu, s obzirom da omogućavaju nedvosmislenu definiciju ključnih termina, ponovnu upotrebljivost elemenata, a takođe su i lako proširljive [20]. U ovom radu, pored upotrebe u okviru domenske analize, razvijeni ontološki radni okvir se dalje iskorišćava kao osnova za automatsko generisanje koda, tako da na taj način pokriva i ostale faze razvoja domenski-specifičnih jezika. Za ovu svrhu se uglavnom koriste takozvane ontologije najvišeg nivoa [17].

Ipak, česta je dilema da li se vremenski i finansijski isplati prvo razviti domenski-specifični jezik, pa onda rešavati problem [18]. Sa druge strane, čak i u slučaju kada postoji dostupan domenski-specifični jezik za oblast razmatranog problema, postavlja se pitanje da li ga upotrebiti ili se ipak osloniti na široko prihvaćene, programske jezike opšte namene. Naravno, razvoj domenski-specifičnih jezika svakako zahteva dodatno vreme, radnu snagu i troškove, pa je zato podjednako bitno razmotriti realnu potrebu za njim. Najčešće, ukoliko je procena da će se u budućnosti često javljati potreba za rešavanje problema iz slične kategorije u tom domenu, pogotovu ako se planira masovna primena od strane većeg broje eksperata, domenski-specifične notacije i jeziki predstavljaju isplativ izbor [17][18][19].

U **Tabeli 2.1** je dat pregled pojedinih postojećih domenski-specifičnih jezika i notacija iz različitih oblasti, relevantnih za ovaj rad. Od značajnih karakteristika, razmatraju se domen od interesa, svrha, obuhvaćeni aspekti i tip notacije (tekstualna ili vizuelna). Kao što se može videti, većinu ovakvih notacija se fokusira na statičke aspekte domena i topologiju, dok su ređe obuhvaćeni aspekti adaptacije sistema, njegovo ponašanje kao odogvor na promene u sredini i koordinacije više elemenata [9]. U ovom radu, pored statičkih, cilj je što bolje iskoristiti dinamičke aspekte i na taj način omogućiti scenarije prilagođavanja i usklađenog ponašanja u okviru studija slučaja.

Tabela 2.1 Pregled domenski-specifičnih jezika u različitim oblastima

Naziv	Domen	Cilj	Aspekti	Notacija
Experiment Description Language (EDL) [21]	Eksperimentalna robotika	Definisanje misija mobilnih robota	Koordinate putanje, aktivacija senzora u određenom trenutku	tesktualna
CloudML [22]	Računarstvo u oblaku	Upravljanje aplikacijama u oblaku	Alokacija resursa, raspoređivanje i adaptacija	tekstualna
DSBPMS [23]	Sistemi za upravljanje poslovnim procesima	Standardizacija u definiciji poslovnih procesa za BPMS sisteme	Akcije i uloge u okviru poslovnih procesa, parametri i formulari za njihov unos	vizuelna
PrintTalk [24]	3D štampa	Defiinisanje štampanog objekta	Toplogija 3D modela i njihova kompozicija	tekstualna
WebML [25]	Razvoj web sajtova	Specifikacija kompleksnih web sajtova na konceptualnom nivou oslanjajući se na grafičku notaciju	Topologija stranica, grafički elementi, linkovi i povezivanje stranica	vizuelna
W3P [26]	Web i multimdija	Automatsko generisanje 3D web prezentacija	Topologija 3D sveta, odgovor na korisničke interakcije sa objektima i njihova animacija	vizuelna
VizDSL [27]	Sistemsko inženjerstvo	Interaktivni prikaz relevantnih informacija o sistemima i procesima	Oblici, topologija pogleda, događaji i akcije	vizuelna
Das Contract [28]	Poslovanje upotrebom blokčejna	Predstavljanje pametnih ugovora nezavisno od konkretne tehnologije, upotrebom vizuelne notacije nalik na BPMN	Struktura ugovora, model podataka, akcije i procesi vezani za transakciju koju ugovor definiše	vizuelna

Dalje, što se tiče implementacije, zastupljene su i tekstualne i vizuelne domenski-specifične notacije. Xtext [16] je često korišćen dodatak u Javi, koji daje mogućnost da se na osnovu formalnog opisa gramatike generišu klase za parsiranje koda i obilazak sintaksnog stabla. Sa druge strane, Ecore u okviru Eclipse Modelling Framework-a (EMF) [29] pruža mogućnost upotrebe grafičkog alata za kreiranje instanci modela, čija struktura podleže definiciji metamodela. Metamodeli se u ovom slučaju zadaju kao UML klasni dijagrami, a na osnovu njih se automatski generišu pomoćne Java klase za obilazak modela. Pri tome, ostvarena je tesna integracija sa Eclipse razvojnim okruženjem, tako da je moguće kreiranje instanci modela, oslanjajući se na bogat grafički korisnički interfejs.

Sa druge strane, Node-RED [30] okruženje, namenjeno low-code realizaciji IoT scenarija oslanjajući se na Node.js[1] tehnologiju, pruža model programiranja u vidu tokova podataka koji se sastoje od čvorova za obradu. Upravo se ovi čvorovi mogu specijalizovati, tako da predstavljaju elemente vizuelne domenski-specifične notacije [8] [9]][31], što je zapravo i iskorišćeno u ovom radu. Grafičko razvojno okruženje koje se izvršava u web pretraživaču pruža jednostavan, intuitivan korisnički inerfejs, što olakšava razvoj domenskim ekspertima bez programerske osnove i smanjuje krivu učenja.

2.2 Ontologije

Ontologija je pojam koji potiče iz oblasti filozofije, a odnosi se na disciplinu koja se bavi izučavanjem opštih, fundamentalnih i konstitutivnih određenja bića. Naziv potiče od grčkih reči: *ontos* – što se tumači kao biće, bitak, postojanje i *logos* – izučavanje, nauka. Pojam ontologija u oblasti računarstva ustaljen je ranih devedestih godina 20. veka, počevši od radova T. Grubera i definisan kao formalna i eksplicitna reprezentacija zajedničke konceptualizacije [32]. Ova reprezentacija obuhvata definiciju ključnih koncepata (vokabular), njihovih svojstava i veza između koncepata koji figurišu unutar jednog ili više domena i to na način razumljiv, kako ljudima, tako i mašinama [32][33]. Kao takve, često se koriste kao model za formalnu reprezentaciju znanja u informacionim sistemima. Dalje, činjenice o određenom domenu, koje su reprezentovane u skladu sa prethodno definisanom ontologijom, predstavljaju semantičke baze znanja. Na ovaj način, mašinama je omogućeno da razumeju koncepte i procesiraju informacije slično kao i ljudi. Osim toga, pojedine definicije ontologije obuhvataju i skup pravila koji se može primeniti nad konceptima i njihovim vezama sa ciljem proširenja baze znanja. Ovaj postupak se naziva i zaključivanje ili rezonovanje (engl. *reasoning*). Prema tome, koristeći ovako definisanu reprezentaciju znanja, dalje je moguće izvesti zaključke i otkriti nove veze na način blizak i razumljiv čoveku, dok je forma istovremeno pogodna za obradu od strane računara [34].

Osnovni elementi koji se tiču ontologija u ovom kontekstu su [32][33]: 1) *koncepti* – apstraktni skupovi ili kolekcije entiteta relevantnih za domen, koji su obično taksonomski organizovani, pri čemu se za ovu svrhu koristi mehanizam nasleđivanja; 2) *relacije* – asocijacije između koncepata, tako da je prvi argument domen (*domain*) relacije, dok drugi argument predstavlja njen opseg (*range*), u opštem slučaju mogu biti unarne, binarne ili *n*-arne; 3) *atributi*

[1] https://nodejs.org/en

ili *svojstva* – predstavljaju posebnu vrstu unarnih relacija, kod kojih je domen relacije koncept, a opseg predstavlja neki tip podataka, poput stringa ili broja; 4) tvrdnje - semantičke rečenice koje se sastoje od jedne ili više povezanih (logičkim ∧) konstrukcija oblika *koncept1 relacija koncept2*; 5) *aksiomi* – tvdrnje o konceptima i njihove relacije za koje pretpostavljamo da su inicijalno tačne, bez dokazivanja, a koirste se kao polazna tačka pri zaključivanju.

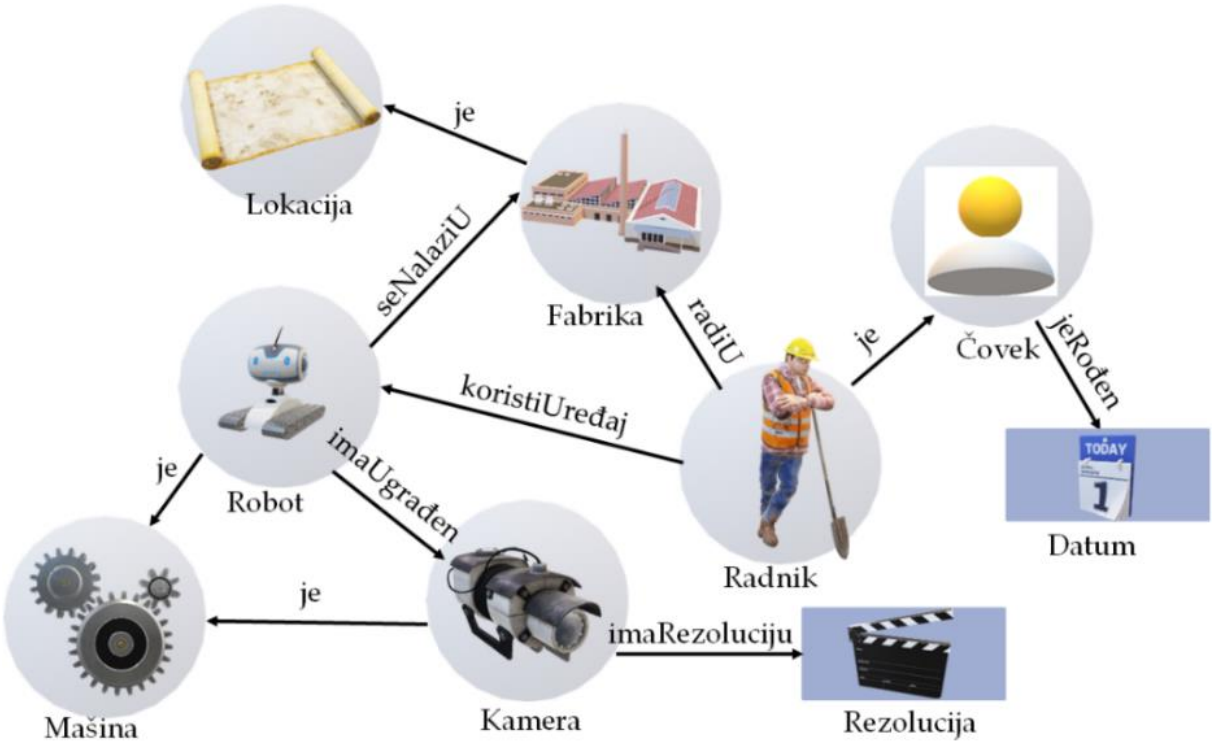
U ovom kontekstu, semantičko zaključivanje predstavlja logički proces izvođenja novih tvrdnji nad formalnm reprezentacijom znanja na osnovu postojećih aksioma i zadatih veza između koncepata, primenom pravila zaključivanja. Jedan od ciljeva ovog procesa može biti bilo kombinovanje različitih znanja sa svrhom davanja odgovora na zadata pitanja ili dolaženja do novih saznanja. Česta forma implementacije semantičkog zaključivanja je zasnovana na skupu pravila u *if-then* formatu. Korišćena notacija[2] za reprezentaciju primera zaključivanja je opisana u **Tabeli 2.2**.

Tabela 2.2 Pregled elemenata notacije za prikaz procesa semantičkog zaključivanja

Aspekat	Konstrukcija	Opis
Relacija između koncepata	*:koncept1 :veza1 :koncept2*	Koncept koji se naziva *koncept1* je u relaciji *veza1* sa konceptom imenovanim *koncept2*.
Tvrdnja	*[?instanca1:koncept1 :veza1 ?instanca2:koncept2]*	*Instanca1* koja je tipa *koncept1* je u relaciji veza1 sa *instancom2* koja po svojoj prirodi *koncept2*.
If-then pravilo	*tvrdnja1, tvrdnjan :- zaključak*	Ako su istinite sve tvrdnje koje su povezane logičkom konjunkcijom u uslovu pravila (if-deo), onda se tvrdnja iz then-dela smatra tačnom. U ovoj notaciji, koristi se oznaka :- za razdvajanje if (levo od nje) i then dela (desno od nje). Što se tiče tvrdnji navedenih u if-delu i razdvojenih zarezom, smatra se da su povezane logičkom konjunkcijom - ∧.

Slika 2.2 prikazuje primer manje ontologije u domenu industrijske proizvodnje. Koncepti su predstavljeni čvorovima, a relacije potezima (početak potega predstavlja domen, a strelica opseg). Kao što se može videti, ovde su ključni koncepti čovek, mašina i lokacija. Radnik je podklasa čoveka, pri čemu radi u fabrici (koja predstavlja lokaciju), a koristi robote (koji su mašine). Dalje, roboti imaju ugrađene kamere. Što se svojstava tiče, datum rođena je karakterističan za čoveka, dok je recimo, za kameru rezolucija sninmka.

[2] https://legislate.ai/blog/an-introduction-to-semantic-technology-and-semantic-reasoning



Slika 2.2 Ilustrativni primer ontologije u industrijskom domenu

U nastavku je dat primer za pravila zaključivanja kod ontologije sa **Slike 2.2**.

Recimo, ako imamo da *radnik1* koristi uređaj *robot1*:

$$radnik1: Radnik : koristiUređaj\ robot1: Robot \quad (2.1)$$

Sa druge strane, neka se *robot1* nalazi u *fabrici1*:

$$robot1: Robot : seNalaziU\ fabrika1: Fabrika \quad (2.2)$$

Na osnovu ovih činjenica, možemo formulisati pravilo da ako se robot nalazi u nekoj fabrici, to podrazumeva da je tamo i radnik koji ga koristi tokom porizvodnje, što bi bilo zadato kao:

[?radnik :koristiUređaj ?robot] , [?robot: seNalaziU ?fabrika]

:- [?radnik :seNalaziU ?fabrika] (2.3)

U ovom slučaju, simbol *:-* označava da ako su tvrdnje sa leve strane tačne, onda se primenjuje pravilo i dodaje nova činjenica kao zaključak u bazi znanja, koja stoji sa desne strane ovog simbola. Iako je mehanizam prilično jednostavan, ovakav pristup ima zaista dosta potencijala da smanji neophodno vreme i trud od strane ljudske radne snage u mnogim sistemima. Postojanje ovakvih pravila pojednostavljuje proces analize podataka, izvođenje novih činjenica, ali i donošenje odluka od važnosti u relevantnim procesima za mnoge domene. Umesto da imamo angažovanog eksperta koji neprestano razmatra nove podatke i donosi zaključke, oslanjajući se na semantičke tehnologije i mehanizme automatskog zaključivanja,

možemo značajno povećati efikasnost. Osim toga, semantičko zaključivanje se može iskoristiti u sinergiji sa mašinskim učenjem sa ciljem unapređenja celokupnog procesa upravljanja podacima. Ova dva pristupa komplementiraju jedan drugog. Dok se mašinsko učenje oslanja na prediktivne modele, koji su obučeni na ogromnoj količini podataka, semantičko zaključivanje iskorišćava ontologije i pravila zaključivanja. Sa jedne strane, primena mašinskog učenja može olakšati proces semantičke anotacije velike količine podataka, što je zadatak koji, ako se obavlja ručno, zahteva dosta vremena. Sa druge strane, ugradnja ekspertskog znanja i domenske intuicije kroz pravila zaključivanja može znatno uštedeti vreme neophodno za treniranje prediktivnog modela, ali i ogroman trud za konstruisanje skupa podataka na osnovu kojeg bi se obučio taj algoritam. Konačno, slepo prihvatanje ishoda prediktivnih modela mašinskog učenja može biti vrlo opasno u pojedinim domenima i slučajevima korišćenja (kao što su zdravstvo i energetika), tako da su objašnjenja zasnovana na semantičkim anotacijama od važnosti za konačno donošenje odluke od strane eksperata [35].

U **Tabeli 2.3** je dat pregled široko primenjenih, ali i pojedinih manje zastupljenih karakterističnih ontologija iz raznorodnih oblasti.

Tabela 2.3 Pregled ontologija primenjivanih u različitim domenima

Naziv	Oblast	Cilj
Human-Computer Interaction Ontology (HCIO) [36]	Interakcija čovek-računar	Definisanje mogućih akcija prilikom interakcije sa različitim interaktivnim sistemima (mobilne aplikacije, pametni gradovi, pametne kuće)
Semantic Sensor Network Ontology (SSN) [37]	Senzorske mreže i prikupljanje podataka	Opis senzora u pogledu njihovih mogućnosti, odgovarajućih procesa merenja, zabeleženih vrednosti, njihove topologije i raspoređivanja
Gene Ontology (GO) [38]	Biologija	Uspostavljanje vokabulara gena, proteina i njihovih uloga.
Systematized Nomenclature of Medicine-Clinical Terms (SNOMED CT) [39]	Zdravstvo	Leksikon medicinskih termina koji ima za cilj standardizaciju, skladištenje, pribavljanje i razmenu zdravstvenih podataka u elektronskoj formi
Coronavirus Infectious Disease Ontology (CIDO) [40]	Epidemiologija	Standardizacija, integracija, deljenje i analiza znanja o koronavirusu
Multi-agent recommendation system ontology in pandemic contenxt [41]	Edukacija	Preporuka edukativnog sadržaja u sistemu za elektronski-podržano učenje na osnovu semantičkih pravila koja obuhvataju zdravstveno, emotivno i mentalno stanje učenika
Avionics Analytics Ontology (AAO) [42]	Avio transport	Koordinisano upravljanje skupom raznovrsnih letelica
Computer-Aided Design (CAD) Ontology [43]	Inženjersko crtanje	Interoperabilnost CAD sistema
Food Recipe Ingredient Substitution	Ishrana	Zamena sastojaka recepata ekvivalentima ili

Ontology [44]		alternativama na osnovu zadatih zahteva i ograničenja
Ontology-based food transformation process [45]	Ishrana	Lanac procesiranja hrane na primeru vina
Urban Tourism Ontology [46]	Turizam	Pretraživanje kulturnih znamenitosti i muzeja
CityGML Cultural Heritage Application Domain Extension (CHADE) [47]	Arhitektura i kulturologija	Proširenje CityGML modela sa elementima za modelovanje kulturoloških aspekata arheoloških nalazišta u urbanom okruženju (osim prostornih)
ROMAIN [48]	Industrija	Održavanje u industriji – rukovanje procesima i servisima u slučaju otkaza nekog dela sistema
Blockchain Ontology with Dynamic Extensibility (BLONDiE) [49]	Blokčejn tehnologije	Uniforman i standardizovan način predstavljanja informacija o blokčejn transakcijama nezavisno od konkretne tehnologije
Ontology-driven approach for Security Orchestration Platform (OnSOAP) [50]	Bezbednost računarskih sistema	Interoperabilnost i orkestracija bezbednosnih sistema za odbranu od hakerskih napada
Ontology of Software: Requirements Engineering Perspective [51]	Inženjerstvo zahteva	Predsatvljanje softverskih artefakata i njihovih veza u procesu razvoja kao osnove inženjerstva zahteva i kasnijih faza evolucije softverskog proizvoda

Između ostalog, kao što se može videti u **Tabeli 2.3**, ontologije su našle najrazličitije primene i u računarstvu – od upravljanja i pretraživanja informacija do ostvarivanja interoperabilnosti [34][35]. U softverskom inženjerstvu, od specifikacije zahteva, preko implementacije do održavanja, veliki broj naučnih rezultata ukazuje da primena ontologija može dovesti do različitih benefita, ali i ubrzati i učiniti celokupan razvoj efektivnijim, oslanjajući se pri tome na generatore koda , što je iskorišćeno kao jedna od osnovnih ideja pristupa predloženog u ovom radu. U literaturi, ontologije korišćene da potpomognu razvoj softvera, najčešće se mogu podeliti u sledeće kategorije [52][53][54][55]: 1) domenske ontologije, 2) metodološke ontologije, 3) ontologije statusa i 4) ontologije procesa. U ovom kontekstu, uloga domenskih ontologija je da pruži kontrolisani, strukutuirani i hijerarhijski organizovanu reprezentaciju entiteta koji figurišu u domenu primene softvera. Najčešće se uz pomoć njih anotiraju podaci o entitetima, sa ciljem da olakšaju pribavljanje relevantnih informacija i obradu podataka od strane aplikacija [53]. Dalje, metodološke ontologije pokrivaju znanje o tome koji su mogući načini obavljanja pojedinih zadataka ili zaduženja u okviru procesa razvoja softvera. Sa druge strane, uloga ontologija statusa je opis statičkih, ali i dinamičkih aspekata softverskog sistema. Dok statički deo opisuje samu strukturu softvera,

dinamički se bavi njegovim ponašanjem. Ontologije procesa povezuju kontekst, sadržaj i strukturu različitih nivoa činjenica i znanja o softverskom sistemu u funkcionalnu celinu [54].

2.3 Semantičke tehnologije

Za implementaciju semantičkih baza znanja i operacija nad njima, koriste se semantičke tehnologije. One predstavljaju skup tehnologija, čiji je cilj da podatke učine razumljivim mašinama, ali i omoguće izvođenje novih zaključaka na osnovu njih [34]. U ovom kontekstu, često se pominje i pojam *semantički web*, koji predstavlja nadogradnju tradicionalnog web-a sa ciljem obezbeđivanja da infromacije imaju potpuno definisano značenje, što bi trebalo da omogući kooperativne aktivnosti ljudi i mašina. Osim toga, semantičke tehnologije se često koriste u sinergiji sa grafovskim bazama podataka, koje su fleksibilnije, jer pribavljanje podataka čine lakšim i efikasnijim, pogotovu što se tiče navigacije na osnovu veza između koncepata [34].

Što se ovog rada tiče, oslanjamo se na Resource Description Framework (RDF)[3]. To je standardni model za opisivanje web resursa, inicijalno zasnovan na XML sintaksi, a koristi se za predstavljanje ontologija i činjenica u semantičkim bazama znanja. U današnje vreme, podržane su i noviji sintaksni standardi za RDF, kao što su Turtle i JSON-LD. RDF šema (RDFS) se koristi za reprezentaciju rečnika uz pomoć kog se dalje opisuju konkretni resursi.

U tom kontekstu, semantička baza znanja ima formu usmerenog grafa, predstavljenog u vidu trojki ili tripleta *(subjekat, predikat, objekat)*. Komponente tripleta imaju sledeće uloge: 1) subjekat – predstavlja resurs koji je predmet opisivanja 2) predikat – odnos između subjekta i objekta 3) objekat – resurs koji u vezi sa subjektom koji opisujemo. U vizuelnim notacijama semantičkih grafova, subjekat i objekat se obično predstavljaju kao čvorovi, dok je predikat poteg između njih. Jedan isti čvor može biti i subjekat i objekat istovremeno, ako figuriše u većem broju tripleta.

Da bismo formirali činjenice o domenu na ovaj način, prethodno je neophodno obezbediti jednistvenu identifikaciju resursa od značaja koji figurišu u tripletima sa prethodno opisanom strukturom. Resursi mogu predstavljati bilo objekte iz okruženja ili apstraktne koncepte koji imaju svoj identitet. Za ovu svrhu, RDF koristi Uniform Resource Identifier (URI), koji predstavlja standardizovani format koji služi za identifikaciju apstraktnih ili fizičkih resursa. Za vrednost URI-ja se često koristi URL koji upućuje na neki konkretan web resurs (poput web

[3] https://www.w3.org/RDF/

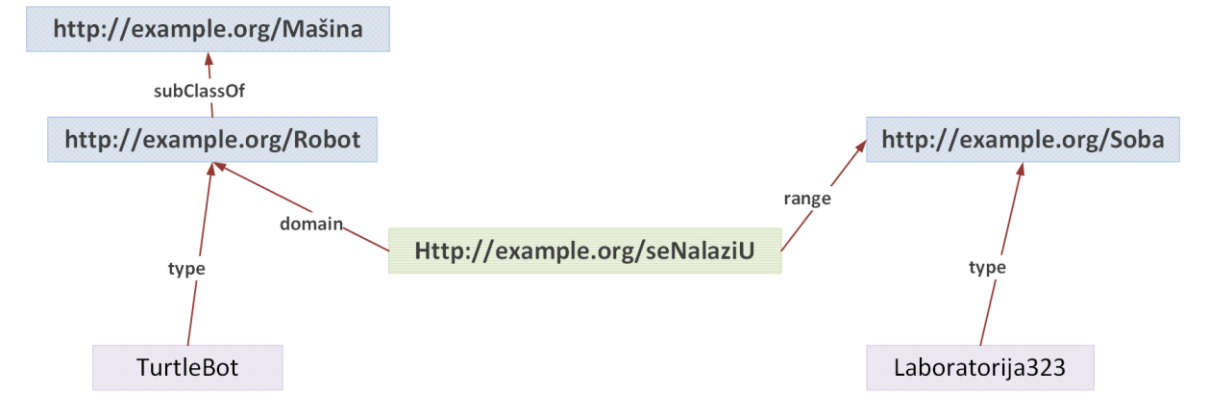
stranice ili dokumenta), ali, generalno, može da sadrži proizvoljne podatke. Kasnije je standardizovan i Inernationalized Resource Identifier (IRI) kao proširenje koje omogućava upotrebu šireg skupa međunarodnih karaktera za identifikaciju resursa.

RDF ima sledeća tri osnovna elementa: 1) *resource* – stvari koje opisujemo 2) *class* – kategorija za grupaciju opisanih stvari 3) *property* – veza između opisanih stvari. Dalje, u **Tabeli 2.4** je dat pregled bitnih svojstava u RDF-u.

Tabela 2.4 Pregled viših ontologija od značaja

Svojstvo	Opis
`rdf:type`	Služi da se označi instanca koje klase je neki od resursa *R rdf:type C* – resurs R je instanca klase C
`rdf:subClassOf`	Tranzitivna relacija, ima za cilj da označi činjenicu da su sve instance klase pripadaju ujedno i drugoj klasi *C1 rdfs:subClassOf C2* – klasa C1 je podklasa klase C2.
`rdfs:subPropertyOf`	Služi da definišu tvrdnju kojom su svi resursi povezani za jednu vrstu svojstva, takođe u vezi i sa još jednom drugom, koja je podsvojstvo inicijalnog – *P1 rdfs:subPropertyOf P2*.
`rdfs:range`	Označava da u tripletima gde je P predikat, objekat predstavljaju instance klase C - *P rdfs:range C*
`rdfs:domain`	Označava da u tripletima gde je P predikat, subjkat predstavljaju instance klase C - *P rdfs:domain C*
`rdfs:label`	Služi da pruži ljudima-čitljive vrednosti za imena RDF resursa

U ovom kontekstu, **Slika 2.3** ilustruje ključne RDF elemente i njihove međusobne odnose.




```
<http://example.org/Robot> <http://example.org/seNalaziU> <http://example.org/Soba>
<http://example.org/Robot> <rdfs:subClassOf> <http://example.org/Mašina>
<http://example.org/seNalaziU> <rdfs:domain> <http://example.org/Robot>
<http://example.org/seNalaziU> <rdfs:range> <http://example.org/Soba>
<<http://example.org/Turtlebot> <rdf:type> <http://example.org/Robot>
<http://example.org/Laboratorija323> <rdf:type> <http://example.org/Soba>
<http://example.org/Turtlebot> <http://example.org/seNalaziU> <http://example.org/Laboratorija323>
```

Slika 2.3 Ilustracija primene ključnih RDF elemenata

U poređenju sa ostalim metodama skaldištenja podataka, RDF je pogodniji za semantičke baze znanja u kojim su podaci povezani na prethodno definisan način: 1) veća fleksibilnost –

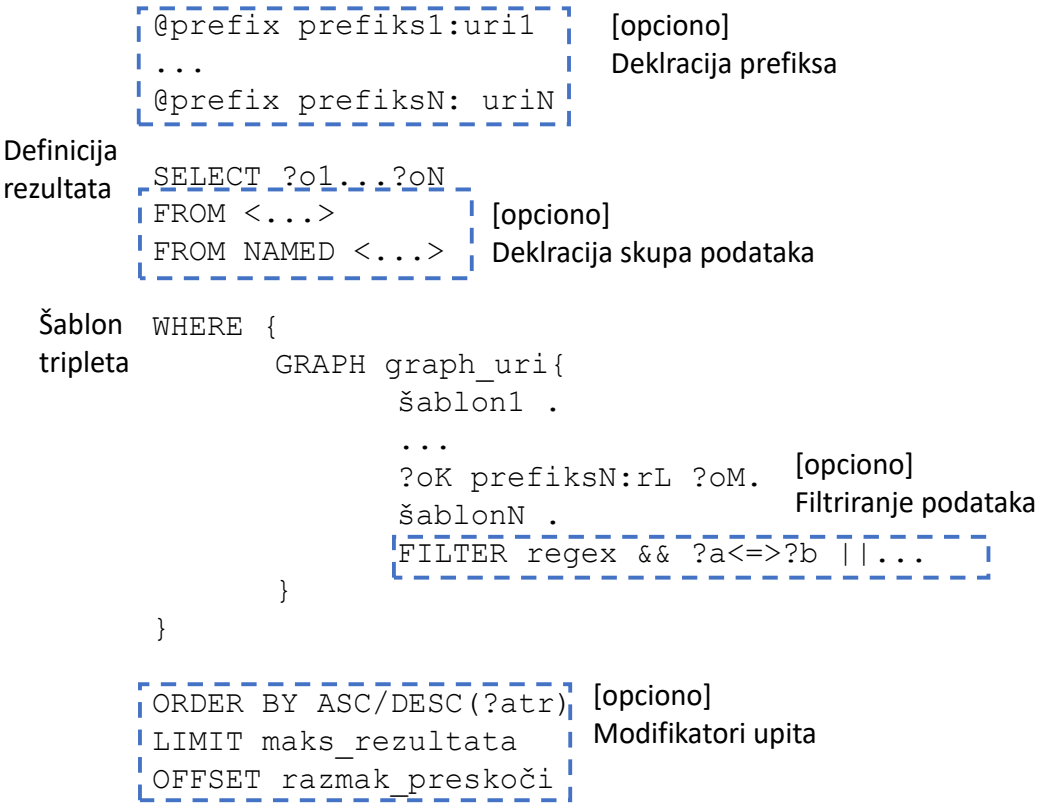
podaci mogu biti pribavljeni iz bilo koje tačke gledišta veze 2) veća efikasnost u bazama ogromnih razmera – nisu linearne kao relacione baze, niti strogo hijerarhijske kao XML.

Zbog ovih svojstava, RDF se koristi kao standard pri realizaciju semantičkih grafova, koji služe za organizaciju, predstavljanje i pretragu informacija na web-u. U ovom kontekstu, semantički grafovi predstavljaju mrežu povezanih čvorova (resursa) i ivica (veza) koje opisuju odnose između tih resursa, pri čemu su resursi identifikovanih uz pomoć URI-ja. U praksi, semantički grafovi mogu doprineti na stvaranje bogatijih, povezanih i smislenijih podataka i to pokrivajući različite aspekte [33]: 1) pretraga informacija - pomažu u poboljšanju rezultata pretrage kroz bolje razumevanje konteksta i značenja pojmova; 2) integracija podataka - omogućavaju integraciju heterogenih podataka iz različitih izvora u jedinstvenu i koherentnu celinu; 3) analiza podataka - koriste se za naprednu analizu podataka, otkrivanje skrivenih veza i obrazaca; 4) obogaćivanje sadržaja - pomažu u automatizovanom povezivanju i obogaćivanju sadržaja na web-u, kao što je dodavanje relevantnih linkova i informacija.

Sa druge strane, SPARQL[4] je jezik, ali i protokol za formiranje upita nad semantičkim bazama znanja koje skladište RDF trojke. On pruža funkcionalnosti za pribavljanje podataka, zatim dalje ispitivanje pribavljenih rezultata, ali i složene spojeve informacija iz više različitih RDF repozitorijuma u okviru jednog upita. Prema tome, oslanjajući se na SPARQL, olakšan je razvoj semantičkih aplikacija. Struktura SPARQL upita nad RDF bazama tripleta je data na **Slici 2.4**. Kao što se može videti, prvi deo je opcion i predstavlja deklaraciju prefiksa koji se u nastavku mogu iskoristiti da zamene odgovarajuće identifikatore resursa (URI). Nakon toga, ide SELECT kojim se definiše kako će izgledati rezultat – koje to elemente izdvajamo iz tripleta u formi *(subjekat, predikat, objekat)*. Kasnije, u uslovima WHERE klauzule se koriste isti ovi nazivi za selekciju u okviru šablona tripleta, ali i izdvajanje uz pomoć filtera. Opciono, moguće je navesti konkretan skup podataka koji se koristi – imenovani ili ne. Nakon toga ide WHERE klauzula u kojoj prvo uz pomoć GRAPH specificiramo URI grafa iz kog želimo da izdvojimo podatke. Zatim, navodimo skup šablona tripleta koji se nadovezuju tačkom. Svaki od ovih šablona je u formi *?subjekat predikat ?objekat*, pri čemu se mogu koristiti prethodno definisani prefiksi. Dalje, po želji je moguće primeniti FILTER klauzulu za izdvajanje samo onih tripleta koji zadovoljavaju date uslove – uklapaju se u šablon regularnog izraza; imaju vrednost jednaku zadatoj, zadovoljavaju relacioni operatori za numeričke vrednosti i slično. Osim toga, pojedinačni uslovi mogu biti povezani logičkim operatorima konjunkcije (`&&`) ili disjunkcije (||). Konačno, na kraju se može primeniti i skup modifikatora upita, po želji: 1) `ORDER BY` –

[4] https://www.w3.org/TR/sparql11-query/

sortiranje vraćenih tripleta po zadatoj vrednost, u rastućem (ASC) ili opadajućem redosledu (DESC); 2) LIMIT – maksimalan broj rezultata koji se vraća iz semantičke baze RDF podataka 3) OFFSET – koliko vraćenih rezultata da se preskoči.



Slika 2.4 Generalizovana struktura SPARQL upita

U nastavku je, na **Slici 2.5** prikazan primer SPARQL upita čiji je cilj da pribavi sve robote koji se nalaze u laboratoriji 323.

```
PREFIX industry: <http://www.example.org/industry>

SELECT ?robot
WHERE {
  ?robot rdf:type industry:Robot ;
         industry:seNalaziU industry:Laboratory323 .
}
```

Slika 2.5 Primer SPARQL upita

Pored toga, nad semantičkim bazama znanja se često primenjuju pravila zaključivanja, uz pomoć kojih se može doći do novih činjenica, polazeći od već postojećih. Prilikom zaključivanja, potrebno je pribaviti određene informacije ili proveriti da li su ispunjeni pojedini uslovi u semantičkoj bazi zananja, a upravo za to se, između ostalog, mogu eksploatisati SPARQL upiti. Jedan od načina realizacije ovog mehanizma jeste upotreba ASK upita koji služe za utvrđivanje da li su pojedini uslovi ispunjeni ili identifikovani šabloni date strukture u

semantičkom grafu. Kao rezultat, ASK upiti vraćaju *true* (ispunjen uslov) ili *false* (nije ispunjen uslov).

U ovom kontekstu, u eksositemu semantičkih tehnologija je relevantan i SHACL (Shapes Constraint Language)[5], koji predstavlja standard za validaciju RDF podataka. SHACL omogućava opisivanje očekivane strukture RDF podataka, kao i pravila i ograničenja koja ti podaci moraju zadovoljiti, što je ključno za obezbeđivanje njihove ispravnosti i doslednosti. Praktično, ovi uslovi se izražavaju u formi RDF grafa, a grafovi sa ovom ulogom se nazivaju *grafovima oblika*. SHACL koristi nekoliko ključnih komponenti za opisivanje i validaciju RDF podataka: 1) oblici (engl. *shapes*) - predstavljaju kolekciju ograničenja koja se primenjuju na čvorove u RDF grafu, a postoje dve vrste oblika: a) oblici čvorova (engl. node shapes) - primenjuju se na čvorove b) oblici svojstava (engl. *property shapes*) - primenjuju se na vrednosti svojstava čvorova; 2) ciljevi (engl. *targets*) - definišu skup čvorova na koje se primenjuju oblici, a oni mogu biti definisani eksplicitno, korišćenjem RDF tipa, ili implicitno, korišćenjem SPARQL upita; 3) ograničenja (engl. *constraints*) - opisuju pravila koja RDF podaci moraju zadovoljiti, a tipični primeri ograničenja uključuju obavezna svojstva, minimalne i maksimalne kardinalnosti.

Što se tiče pomoćnih alata za skladištenje semantičkih podataka u okviru RDF repozitorijuma i programatskog interfejsa za njihovo pribavljanje, oslanjamo se na Open Testbed-as-a-Service with Ontology Repository (Open TaaSOR) platformu [56]. Ona je inicijalno namenjena semantičkoj anotaciji servisa i aspekata vezanih za eksperimente u testbedovima, prvenstveno u oblasti mreža poslednje generacije. TaaSOR obuhvata javno dostupan repozitorijum za objavljivanje ontologija, ali mehanizme za njegovo upravljanje – kreiranje novih ontologija, uređivanje, dodavanje tripleta, izvršenje upita, prikaz rezultata. Navedene funkcionalnosti su dostupne kroz web grafički korisnički interfejs, ali i kroz Java programski kod, oslanjajući se na skup biblioteka koji u pozadini koristi open source rešenja, poput Apache Jena. U **Tabeli 2.5** je dat kratak pregled najbitnijih funkcija TaaSOR API-ja u Java programskom jeziku za operacije nad ontologijama i izvršavanje upita.

Tabela 2.5 Pregled najbitnijih TaaSOR API funkcija u Javi

Funkcija	Efekat
`SPARQLstoreClient(serviceEP): SPARQLstore`	Konstruktor SPARQL klijenta za izvršenje semantičkih upita nad bazom tripleta ili šemom ontologije koju pruža servis zadat URL-om pod nazivom *serviceEP*. Kao povratnu vrednsost vraća *SPARQLstore* objekat nad kojim se mogu izvršavati upiti koji se primenjuju na semantičku bazu datu preko URL-a.

[5] https://www.w3.org/TR/shacl/

`TripletUpdateRequestBuffer (int size, String updateDS, SPARQLstore store, UpdateAction action)`	Konstruktor bafera zahteva za operacije nad izvorm podataka veličine *size*, zadatog URI-jem kao patametar *updateDS*, koristeći store klijenta za SPARQL, pri čemu je odgovarajuća akcija: *UpdateAction.INSERT* – za ubacivanje novih tripleta; *UpdateAction.DELETE* – za brisanje tripleta.
`TripletUpdateRequestBuffer. addDataPropTriplet( String subjectURI, String predicateURI, String objectValue, String objectType):void`	Dodati triplet u baferu zahteva za ažuriranje semantičke baze tripleta, pri čemu su kao argumenti zadati URI subjekta i predikata, vrednost i tip objekta.

`store.executeDatasetSPARQL(String SPARQLquery, OutputStream baos, String resultType, String arg3): void`	Izvršava SPARQL upit zadat stringom nad semantičkom bazom tripleta *store*, pri čemu se formatiranje rezultata specificira argumentom *resultType* (u ovoj disertaciji se koristi "json"), dok se rezultat baferuje u izlaznom toku *baos*.
	Dalje, pozivom ove metode se rezultatima datog upita popunjava bafer. Nakon toga se ovaj bafer pretvara u string, a zatim kreira JSONObject na osnovu njega.
	`JSONObject aux_obj = new JSONObject(baos.toString());`
	Konačno, iz ovog JSON objekta preuzimamo rezultate (niz koncepata, veza ili vrednosti) kroz petlju čiji je indeks *i*:
	`String result= aux_obj.getJSONObject("results"). getJSONArray("bindings").get(i).toString();`

2.4 Sistematski pristup razvoju ontologija

Što se tiče sistematskog pristupa u razvoju ontologija, u literaturi se tokom godina sreću različite metodologije, a jedna od skorijih metodogija koja prilično uspešno generalizuje opšterohvaćene faze i korake pri razvoju ontologija iz ranijih radova na ovu temu, data je u [55]. **Tabela 2.6** prikazuje rezime ključnih faza, sa njihovim koracima.

U fazi identifikacije zahteva, potrebno je utvrditi zahteve koje ontologija treba da ispuni i to po pitanju cilja (ili svrhe, zbog čega pravimo ontologiju) i opsega realnog sveta koji ona obuhvata. Kao pomoćno sredstvo u ovoj fazi se obično sastavljaju i takozvana kompetentna pitanja na koja ontologija treba da bude u mogućnosti da pruži odgovor. Dalje, sledeća faza predstavlja dizajn ontologije. Prvi korak dizajna jeste konceptualizacija, čiji je ishod skup ključnih termina koji su relevanti za ispunjenje zahteva ontologije u definisanom opsegu. Nakon toga, obično se razmatraju postojeće, opšteprihvaćene ontologije sa ciljem ponovne upotrebe, ukoliko postoje preklapajući elementi sa novom ontologijom koja se projektuje. Ukoliko postoje elementi u već postojećim ontologijama, preuzimaju se, po potrebi dopunjuju i povezuju na adekvatan način sa ostalim konceptima ontologije u izradi. Posle ovog koraka,

prelazi se na formalizaciju, čiji je cilj eksplicitna specifikacija novonastale ontologije, bilo u tekstualnoj ili vizuelnoj notaciji (semantički graf). Nakon dizajna, prelazi se na implementaciju prethodno formalizovanog opisa ontologije, i to korišćenjem neke od postojećih notacija (kao što je RDF u ovoj disertaciji), uz upotrebu pomoćnih alata (Open Tasor u ovom radu). Obično je ishod upotrebe ovih alata skup tripleta kojim se definišu ključni elementi novonastale ontologije, a zatim se ubacuju u odgovorajući triple store. Dalje, nakon implementacije se (opicono) dodaju probni tripleti instanci, koji su u skladu sa datom ontologijom radi inicijalne provere upotrebljivosti novonastale ontologije. Poslednji korak u konstrukciji ontologije jeste njena integracija sa odgovarajućim semantički-omogućenim sistemom, koji se na nju oslanja sa određenom svrhom. Kao primer, u ovom radu, ontologije se koriste od strane algortima za automatsko generisanje koda. Konačno, nakon implementacije ontologije i njene integracije sa odgovarajućim sistemom, pristupa se evaluaciji, sa ciljem utvrđivanja kvaliteta dobijenog rezultata. Prvi nivo evaluacije jeste provera da li se uz pomoć novodobijene ontologije može odgovoriti na inicijalno identifikovana kompetentna pitanja iz prve faze. Obično se za ovu svrhu definiše skup SPARQL upita uz pomoću kojih bi trebalo pribaviti rezultate koji predstavljaju odgovore na ova pitanja. Osim toga, mogu se izvršavati i kompleksniji upiti nad instancama, sa ciljem provere da li važe definisana ograničenja i pravila u semantičkoj bazi znanja nastaloj oslanjajući se na novonastalu ontologiju.

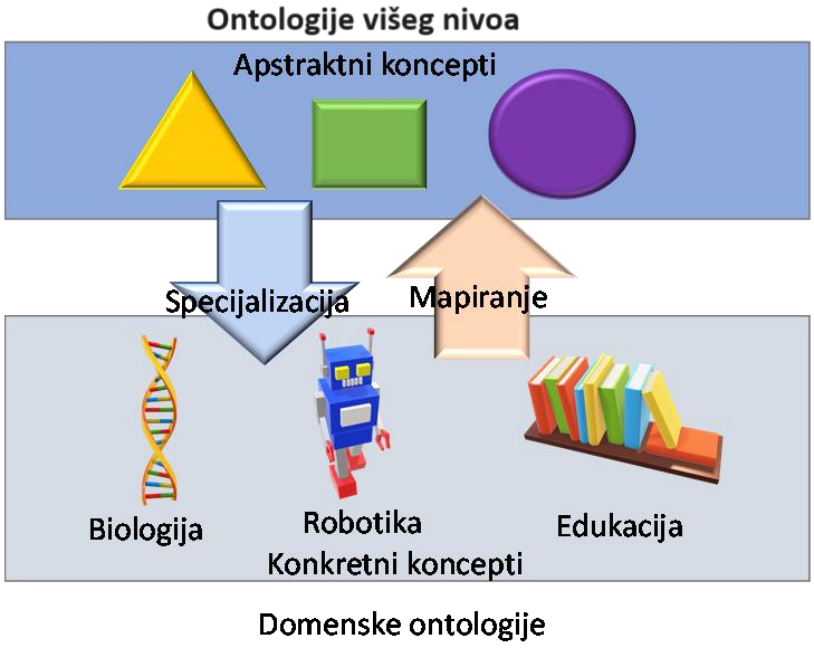
Tabela 2.6 Faze i koraci u razvoju ontologija

Faza	Koraci
Identifikacija zahteva	Prepoznavanje svrhe
	Utvrđivanje opsega
	Identifikacija kompetentnih pitanja
Dizajn ontologije	Utvrđivanje ključnih termina (konceptualizacija)
	Modularizacija ontologije i integracija sa postojećim ontologijama
	Formalizacija
Konstrukcija ontologije	Implementacija modula ontologije - kodiranje
	Dodavanje individua (populacija ontologije)
	Integracija sa semantičkim sistemom
Evaluacija ontologije	Evaluacija kompetentnosti
	Evaluacija kvaliteta

2.5 Ontologije višeg nivoa

Fundamentalne ontologije se definišu kao domenski-nezavisne ontologije najvišeg nivoa, čiji je cilj da pruže generalni opis koncepata [58]. Osim toga, u literaturi se često pominju i kao gornje ontologije. One obuhvataju reprezentacije najgeneralnijih apstraktnih entiteta (tipova) i njihovih relacija, na koje se oslanjamo da bismo konstruisali konkretnije, specijalizovane

aspekte. Sa druge strane, domenske ontologije specificiraju koncepte, njihova svojstva i veze sa tačke gledišta specifičnog domena. Prema tome, na osnovu fundamentalnih možemo konstruisati domenski-specifične ontologije. Fundamentalne onologije su uglavnom ograničene na generičke, apstraktne i filozofske koncepte, koji su osnovni za razumevanje sveta od strane ljudi. Između ostalog, koriste se i nazivi *univerzalne* ili *osnovne* ontologije. Osim toga, domenske ontologije izvedene iz zajedničke osnove se lako mogu prevoditi jedna na drugu, što olakšava ostvarivanje interoperabilnosti sistema iz različitih domena. Prema tome, postoje i referentne ontologije, kao još jedan tip ontologija višeg nivoa uz pomoć kojih se ostvaruje veza između domenskih i fundamentalnih. Iz ovog razloga, s obzirom da se po nivou apstrakcije nalaze između, referentne ontologije se često nazivaju i ontologijama srednjeg nivoa. Ilustracija procesa primene ontologija višeg nivoa za implementaciju domenski-specifičnih je prikazana u okviru **Slike 2.6** .



Slika 2.6 Primena ontologija višeg nivoa

Suštinski, postoje dva pristupa u upotrebi ovih ontologija [58]: odozgo naniže (*top-down*) i odozdo naviše (*bottom-up*). Što se prvog pristupa tiče, ovde ontologiju višeg nivoa primenjujemo kao osnovu za dalje izvođenje koncepata u domenskoj ontologiji. Na ovaj način, kreator domenske ontologije ugrađuje znanje i iskustvo implicitno ugrađeno u višu ontologiju. U slučaju drugog pristupa, nova ili već postojeća domenska ontologija se koristi za mapiranje na generlizovanu, opštiju – višu ontologiju. U ovom radu se koristi prvi pristup, tako da se polazeći od ontologija višeg nivoa izvode domenske ontologije. U **Tabeli 2.7** je dat pregled istaknutih ontologija višeg nivoa, zajedno sa kratkim opisom i ključnim karakteristikama.

Tabela 2.7 Pregled značajnih ontologija višeg nivoa

Naziv	Opis
Basic Formal Ontology (BFO) [60]	Ima za cilj da olakša pribavljanja, analizu i integraciju u okviru naučnih ili drugih domena. Jedna od glavnih ideja je podela entiteta u dve disjunktne kategorije: kontinuirani i javljajući. Dok se prvi sastoje od objekata i prostornih regiona, druga grupa se odnosi na porcese koji se prostiru u vremenu. Prema tome, BFO teži da pokrije aspekte prostora i vremena u okviru jedinstvenog radnog okvira.
Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [61][62]	Orijentisana je prikupljanju osnovnih ontoloških kategorija ljudskog govora i zdravorazumskog razmišljanja. Kategorije koje uvodi su zapravo kognitivni artefakti, koji zavise od ljudske percepcije, kultroloških karakteristika i socijalnih konvencija. Trenutno broji 76 klasa, 112 svojstava i 581 aksiom. Osnovni koncepti ove ontologije su: 1) održnik ili endurantni entitet (engl. *endurant*) - entiteti koji postoje kroz vreme bez promene u svom identitetu – objekti i supstance 2) protrajni ili perdurantni entiteti (engl. *perdurant*) - entiteti koji se odvijaju u vremenu, što uključuje događaje, procese i aktivnosti 3) kvaliteti - predstavljaju inherentne karakteristike ili osobine entiteta 4) relatori -predstavljaju odnose između entiteta, specificiraju na koji način su entiteti povezani jedni sa drugima. 5) uloge - funkcija ili pozicija koju entitet igra u određenom kontekstu ili situaciji 6) vreme - predstavlja temporalnu dimenziju i pruža okvir za modeliranje temporalnih aspekata entiteta i događaja 6) prostor - predstavlja prostornu dimenziju i pruža okvir za modeliranje prostornih odnosa između entiteta 7) kolekcije - skupovi ili grupacije entiteta koji dele neke zajedničke karakteristike ili osobine.
Unified Foundational Ontology (UFO) [63]	Istraživački poduhvat dug maltene dve decenije sa različitim proširenjima u toku svog veka postojanja. Obuhvata sledeće aspekte: taksonomije i tipove; atribute i svojstva; odnose delova i celine; događaji; uloge.
Suggested Upper Merged Ontology (SUMO) [64]	Definiše hijerarhiju klasa, njihove veze, ali i pravila povezivanja (sintaksa nalik na LISP programski jezik). Namenjen je za ostvarivanje interoperabilnosti mehanizama za zaključivanje. Incijalno, fokus ove ontologije je bio na definisanju meta-koncepata – generalnih entiteta, koji ne pripadaju niti jednom specifičnom domenu. Osnovni koncepti obuhvataju: fizičke i apstraktne objekte, događaje, procese, stanja, atribute i realcije.
Common Semantic Model (COSMO) [65]	Kolaborativni, otvoreni projekat čiji je krajnji cilj olakšavanje semantičke interoperabilnosti. Članovi mogu proširivati dodavanjem elemenata koje smatraju neophodnim, a specijalizacijom osnovnih koncepata se dobijaju domenske ontologije.
Super Ontology (Sup_Ont) [58]	Opisuje strukutru univerzuma i definiše koncepte stvarnosti. Vodi se pretpostavkom da su svi entiteti univerzuma trajni, ali su neprestano podložni promenama. Ove promene se dalje dele na prirodne i veštačke. Dalje, u ontologiji postoje još neke podele entiteta, poput: konkretni ili apstraktni, živi ili neživi, prirodni ili stvoreni od strane čoveka, pokretni ili statični i slično.
General Social Ontology (GSO) [66]	Definiše generalizovanu strukturu socijalnih mreža u vidu proširenja FOAF ontologije, pri čemu obuhvata aspekte društvenog konteksta u vidu društvenih uloga i institucionih pravila.

U ovoj disertaciji, viša ontologija se koristi za formalni apstraktni opis prilagodljivog inteligentnog sistema, koji se dalje može iskoristiti za izvođenje domenskih ontologija specijalizacijom. Što se konkretne implementacije više ontologije korišćene u ovom radu tiče, inspirisana je SUMO [64] ontologijom, s obzirom da u predloženom radnom okviru figurišu fizički i apstraktni objekti, događaji, procesi (koordinacija i adaptacija), atributi i relacije. Osim toga, po pitanju promenljivosti entiteta, u predloženoj višoj ontologiji se oni dalje dele na statičke i dinamičke, što je preuzeto iz DOLCE [61][62]. Na osnovu predložene više ontologije dobijamo domenske ontolgoije, koje se dalje koriste kao osnova za automatsko generisanje koda i pomoćnih alata, poput grafičkih okruženja za modelovanje konkretnih instanci sistema i slično.

2.6 Operacije nad ontologijama

Vrlo često se već postojeće ontologije upotrebljavaju sa ciljem kreiranja novih ontologija, da bi se uštedelo vreme i trud ali i da bi se obezbedila konzistentnost u kompleksim ontološkim sistemima. Što se ponovne upotrebe ontologija sa svrhom stvaranja novih tiče, ključni procesi koji se koriste jesu mapiranje, poklapanje, stapanje, poravnanje, anotacija podataka i integracija ontologija [67]. Što se tiče tehnika i pomoćnih alata za implementaciju ovih operacija, koriste se metode iz različitih oblasti – metrike sličnosti u vektorskim prostorima, tehnike obrade prirodnog govora, mašinsko učenje, otkrivanje informacija i grafovski algoritmi, ali i rečnici termina kao što je WordNet [67][68].

Pregled ovih operacija, njihovih ulaza i izlaza, dat je u **Tabeli 2.8**.

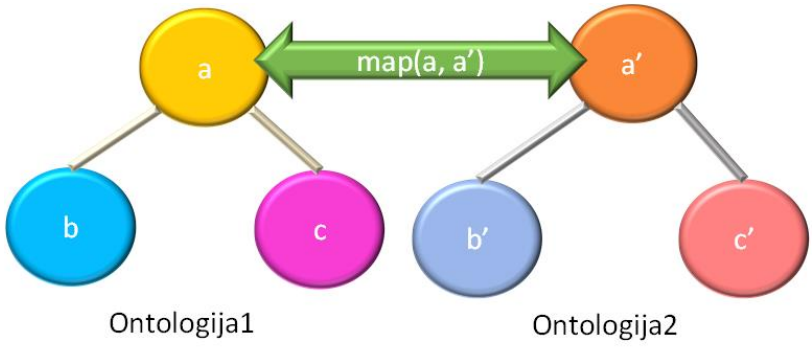
Tabela 2.8 Rešenja za automatsko generisanje koda upotrebom ontologija

Operacija	Opis	Ulazi	Izlaz
Mapiranje	Uspostavljanje semantičke veze koncepata iz različitih ontologija.	Dve ontologije – O1 i O2	Veze između koncepata ontolgija O1 i O2
Poklapanje	Pronalaženja korespondencije između lingvistički povezanih entiteta iz različitih ontologija.	Dve ontologije – O1 I O2; rečnici (opciono)	Skup entiteta koji se poklapaju
Stapanje	Proces generisanja jedne koherentene ontologije, polazeći od dve ili više ontologija koje su iz istog ili preklapajućih domena	Dve ontologije – O1 i O2	Nova ili objedinjujuća ontologija – O’
Poravnanje	Poravnanje ontologija je zadatak uspostavljanja veza između dve ontologije, u slučaju kada je potrebno da ostanu konzistentne jedna sa drugom, ali ostaju zasebne		
Integracija	Integracija ontologija je proces		

	generisanja nove ontologije na osnovu dve ili više koje su iz različitih domena, ili koje se bave različitom tematikom		
Učenje ontolgoija	Automatsko otkrivanje i konstruisanje ontologija, polazeći od korpusa tekstova	Tekst	Ontologija
Semantička anotacija	Pproces kreiranja metapodataka, oslanjajući se na ontologiju kao rečnik termina.	Tekst Ontologija	Tripleti

2.6.1 Mapiranje i poklapanje ontologija

Mapiranje ontologija je formalni izraz, kojim se uspostavlja semantička veza koncepata iz različitih ontologija. Ova operacija kao ulaz ima dve ontologije O_1 i O_2, a definiše se kao poklapanje entiteta jedne od ontologija sa najviše jednim entitetom iz druge. Generalno, postoje dva tipa mapiranja: 1) na apstraktnom nivou – obavlja se na nivou samih ontologija, i 2) na konkretnom nivou – nad konkretnim instancama ontologija. Prema tome, mapiranja su obično apstraktne ili konkretne, funkcije ili relacije između artefakata različitih ontologija (obično koncepata), a suštinski odgovaraju preciznoj ili približnoj sličnosti, identičnosti ili supsumciji (podređivanje posebnog pod opšte). U najprostijoj formi, mapiranje možemo predstaviti kao relaciju *map(a,a')*, pri čemu su $a \in O_1$ i $a' \in O_2$ artefakti iz različitih ontologija [69]. Ova realcija postaje konkretna, tako što *map* možemo zameniti operatorom iz skupa $(\approx, \equiv, \subseteq, \supseteq)$. Osim toga, mapiranja se mogu odnositi ne samo na koncepte, već i na svojstva artefakata. Proces mapiranja ontologija ilustrovan je na **Slici 2.7** .



Slika 2.7 Mapiranje ontologija

Poklapanje ontologija se odnosi na proces pronalaženja korespondencije između na neki način (najčešće lingvistički) povezanih entiteta iz različitih ontologija. Kao rezultat, generiše se skup mapiranja *M u obliku* $map_k(a_i, a_j')$ između koncepata dve ontologije, pri čemu $a_i \in O_1$ i $a_j' \in O_2$, pri čemu i može imati vrednosti od 1 do *N*, a j od 1 do *M*, pri čemu su *N* i *M*, redom, ukupan broj koncepata svake od ulaznih ontologija. Poklapanja se uspostavljaju u obliku veza

ekvivalencije (*is-a*), specijalizacije/generalizacije (*subclass-of*) i kompozicije (*part-of*) između koncepata polaznih ontologija. Ovaj proces se obično odvija oslanjajući se na tehnike obrade govornog jezika, klasifikacije reči, upotrebom unapred definisanih rečnika ili ručno, od strane domenskih eksperata.

Mapiranje i poklapanje ontologija često se koriste kao osnovne operacije za složenije procese, kao što je rezonovanje na osnovu analogija i izvođenje novih ontologija na osnovu postojećih [70][71], o čemu je dato više detalja kasnije u tekstu.

2.6.2 Stapanje, integracija i poravnanje ontologija

Stapanje ontologija je proces generisanja jedne koherentene ontologije, polazeći od dve ili više ontologija koje su iz istog ili preklapajućih domena. Novodobijena ontologija uključuje informacije iz svih izvornih ontologija na jednom mestu. U ovom slučaju, iako polazne ontologije imaju istu tematiku, one su inicijalno zasebne i ne predstavljaju reviziju neke zajedničke osnove. Nakon pronalaženja korespondencije između dve ontologije, mapirani koncepti se spajaju u jednu ontologiju.

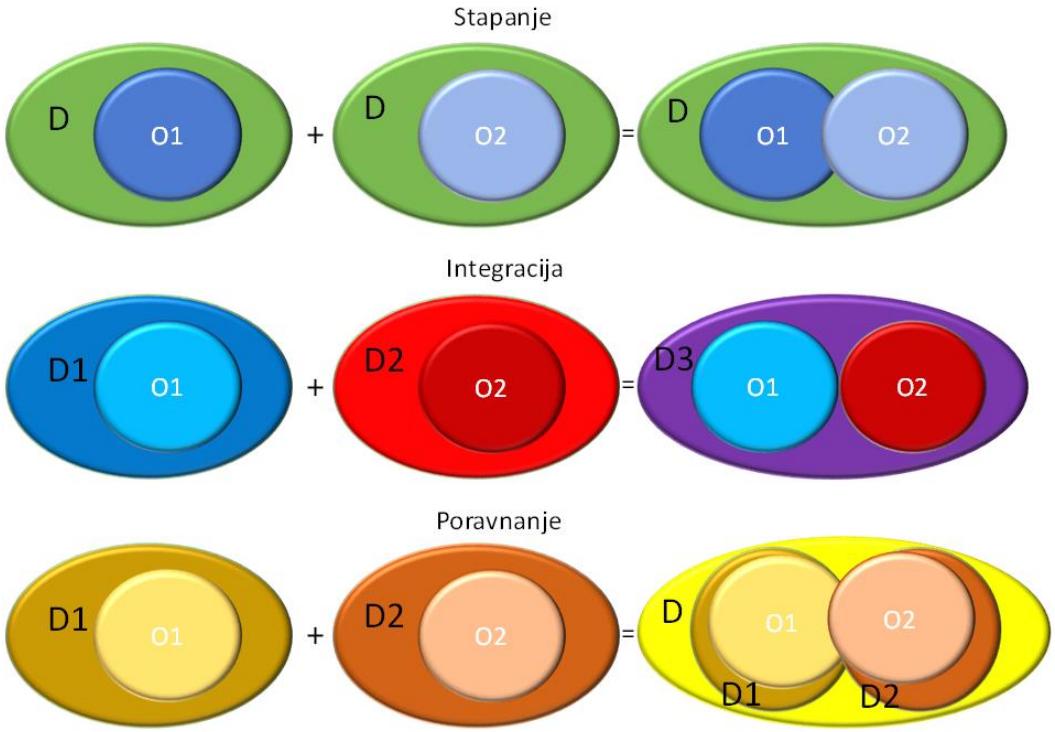
Kao ulaz u ovaj proces imamo dve ontologije O_1 i O_2 koje opisuju isti domen D. Pretpostavimo da je svaka ontologija O predstavljena trojkom (C, R, D), pri čemu je: C – skup koncepata, R – skup odnosa koncepata (relacija), a D – domen koji ontologija opisuje. Prema tome, što se tiče ontologija koje su ulaz procesa stapanaja, možemo ih označiti kao O_1:(C_1, R_1, D), i O_2:(C_2, R_2, D), respektivno. Ishod procesa stapanja ontologija jeste rezultujuća ontologija O' koja opisuje isti domen D, ali za čije koncepte i relacije važi da su unija elemenata iz polaznih ontologija – O': $(C_1 \cup C_2 , R_1 \cup R_2, D)$.

Sa druge strane, integracija ontologija je proces generisanja nove ontologije na osnovu dve ili više koje su iz različitih domena, ili koje se bave različitom tematikom. Iako se ostvaruje njihovo povezivanje, ove dve ontologije nakon integracije nastavljaju životni vek kao zasebne celine. Za razliku od stapanja, ovde se ulazne ontologije mogu predstaviti kao O_1:(C_1, R_1, D_1), i O_2:(C_2, R_2, D_2), pri čemu su D_1 i D_2 dva različita, nepreklapajuća domena D_1 i D_2, za koje važi da nemaju zajedničkij koncepata i relacija: $C_1 \cap C_2 = \emptyset$ i $R_1 \cap R_2 = \emptyset$. U ovom slučaju, rezultujuća ontologija se može označiti kao O': $(C_1 \cup C_2 , R_1 \cup R_2, D_3)$.

Poravnanje ontologija je zadatak uspostavljanja veza između dve ontologije, u slučaju kada je potrebno da ostanu konzistentne jedna sa drugom, ali ostaju zasebne. Ovo se uglavnom radi kada imamo ontologije koje su iz komplementarnih domena. U ovom slučaju su ulazne

ontologije O_1:(C_1, R_1, D_1) i O_2:(C_2, R_2, D_2), pri čemu za domen rezultata D *važi* $D_1 \subseteq D$ i $D_2 \subseteq D$, pri tome postoje zajednički koncepti $(\exists c: c \in C_1 \wedge c \in C_2)$ i relacije $(\exists r: r \in R_1 \wedge r \in R_2)$.

Ilustracija prethodno opisanih procesa stapanja, integracije i poravnanja ontologija je data u notaciji Venovih dijagrama na **Slici 2.8**.

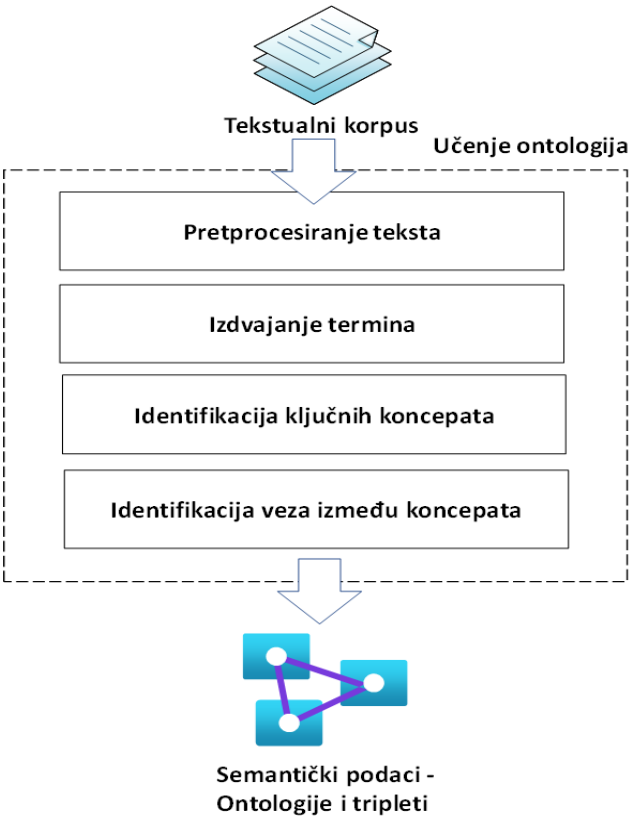


Slika 2.8 Stapanje, poravnanje i integracija ontologija u notaciji Venovih dijagrama

2.6.3 Učenje ontologija

Još jedan kompleksniji proces koji se odnosi na formiranje novih ontologija jeste učenje ontologija. On predstavlja automatsko otkrivanje i konstruisanje ontologija, polazeći najčešće od korpusa tekstova, ali i drugih tipova podataka kao što su video, zvuk, vremenske serije koje dolaze sa senzora i slično [72][73][74]. Problem učenja ontologija bi se mogao definisati na sledeći način: na osnovu korpusa tekstova T, potrebno je identifikati skup koncepata $c_i \in C$ i skup relacija između njih $r(c_k, c_m) \in R$. Tipični koraci procesa učenja ontologija [72][73][74]: 1) priprema teksta – pretprocesiranje sa svrhom da se obezbedi odgovarajući format 2) identifikacija termina iz teksta 3) izdvajanje ključnih termina – koriste se metrike poput tf-idf i rečnici termina 4) identifikacija veza između koncepata – generalno, obuhvata dva tipa: taksonomske ili hijerarhijske (hipernimi i hiponimi), ali i netaksonomske ili nehijerarhijske veze (posesija – has-a, meronimija - part-of, prepoznavanje svosjtava koncepata - property i uzročnost). Jezički posmatrano, imenice predstavljaju koncepte, dok su glagoli kandidati za različite vrste relacija. Često korišćena biblioteka, kao pomoćno sredstvo za ovu svrhu,

zasnovana na tehnikama obrade govornog jezika jeste Stanford CoreNLP – Natural Language Processing Toolkit [75]. Omogućava da se, uz pomoć rečnika za konkretan jezik, na osnovu isečka teksta dobije skup tripleta (subjekat, predikat, objekat), koji se dalje procesiraju sa ciljem određivanja važnosti termina i otkrivanja direktnih, ali i indirektnih relacija između njih. U radu [76] je predstavljena primena ove biblioteke za formiranje ontologija iz biološkog domena, sa ciljem semantičke predikcije funkcija proteina na osnovu skrivenih veza iz tekstova. Proces učenja ontologija polazeći od tekstualnog korpusa je ilustrovan na **Slici 2.9**.

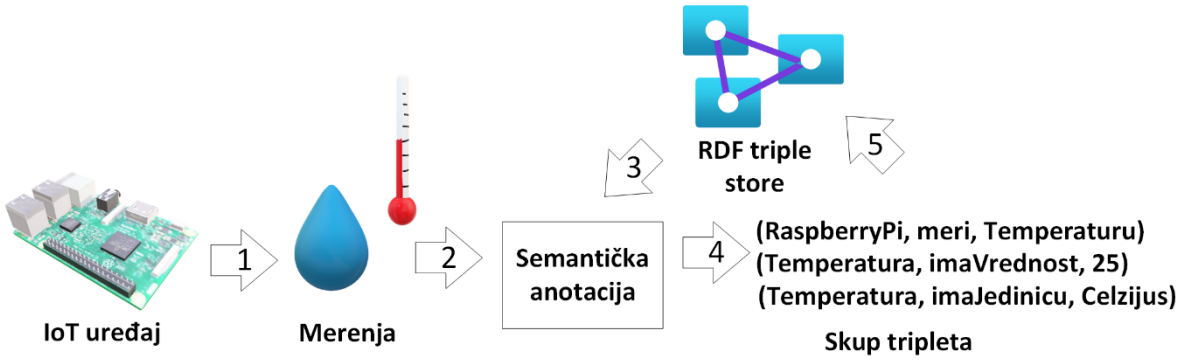


Slika 2.9 Proces učenja ontologija na osnovu tekstualnog korpusa

2.6.4 Semantička anotacija

Sa druge strane, semantička anotacija podataka se odnosi na proces kreiranja metapodataka, oslanjajući se na ontologiju kao rečnik termina. U ovom slučaju se nekoj vrednosti (nizu karaktera) ili resursu – *v*, dodeljuje jedan ili više tripleta u skladu jednom ili više odabranih ontologija $(O_1...O_N)$. U ovim tripletima, anotirana vrednost se nalazi u ulozi subjekta ili objekta. Prema tome, ishod ovog procesa je skup tripleta T_v oblika (v, p_i, o_i) ili (s_k, p_k, v). Pri tome su $(p_i, o_i) \in O_i$, a $(s_k, p_k) \in O_k$, pri čemu je $O_i \in (O_1...O_N)$ i $O_k \in (O_1...O_N)$. Na ovaj način se nekoj vrednosti dodeljuje semantički opis pomoću kojeg se definiše šta zapravo ta vrednost predstavlja. Ilustracija procesa semantičke anotacije senzorskih podataka koji

dolaze sa IoT uređaja je data na **Slici 2.10**. Često se primenjuje korak parsiranja za izdvajanja podataka od interesa iz sirovih (merenja i slično) pre nego što se obavi semantička anotacija. Zatim, bira se odgovarajuća ontologija koja će biti korišćena za semantičku anotaciju ovih podataka. Po prijemu vrednosti, konstruiše se skup tripleta u skladu sa odabranom ontologijom i, konačno, ubacuje u triple store.



Slika 2.10 Ilustracija procesa semantičke anotacije na primeru IoT senzorskih podataka: 1-Čitanje vrednosti sa senzora 2-Parsiranje sirovih vrednosti senzorskih merenja 3-Ontologija za semantičku anotaciju senzorskih podataka 4-Konstrukcija tripeta 5-Ubacivanje kreiranih tripleta

2.7 Automatizovani razvoj ontologija upotrebom analogije i kompetentnih pitanja

Iako su ontologije široko primenjene za reprezentaciju znanja u različitim domenima i aplikacijama, učešće ljudskog faktora i skup ručnih koraka predstavljaju usko grlo u njihovom efikasnom razvoju [71] . Jedan od mogućih načina da se ovaj problem prevaziđe jeste ponovna upotreba već postojećeg znanja, odnosno ontologije koje su već razvijene se mogu primeniti u više različitih aplikacija. Međutim, ako nam je potrebna ontologija koja treba da pokrije neki kompletno drugačiji domen, onda moramo pristupiti problemu drugačije. Da bi se prevazišao ovaj problem, jedan od pristupa jeste algoritam zasnovan na sličnosti, odnosno analogiji koja postoji između različitih domena. Na ovaj način, u sinergiji sa mapiranjem ontologija, dobijamo mogućnost da povežemo koncepte koji imaju slične uloge u svojim domenima. Cilj ovakvog pristupa jeste da se razvoj ontologija učini što efikasnijim kontinualno sa povećanjem količine akumuliranog znanja.

U sistematskom pristupu razvoju ontologija [55], postoje dve koraka kod kojih maltene uvek postoji dilema da li je dovoljno ono što već imamo u postojećim ontologijama ili je neophodno uvesti nove koncepte i veze. Ta dva koraka su konceptualizacija i kodiranje. Upravo ovaj problem jeste jedan od većih izazova što se tiče metodologija razvoja ontologija. Po pitanju

cene razvoja novih ontologije, trud potreban za pojedine korake se može izraziti, recimo kao osoba-mesec ili kroz trajanje samih aktivnosti u satima ili danima. Ponovnom upotrebom ontologija za razvoj novih može redukovati ukupno vreme koje je potrebno za proces razvoja [71]. Međutim, čak ukoliko se ustanovi da slični koncepti zaista već postoje, nije uvek očigledno kako se ovi koncepti mogu prilagoditi i ponovo upotrebiti. U ovom radu, ideja je prevazići prethodno navedeni problem oslanjajući se na analogije [70][77].

Analogija se u ovom kontekstu definiše kao strukturalna sličnost svojstava dva objekta. Dalje, metoda spoznaje po analogiji je kognitivni proces koji se odnosi na određivanje analogija između sličnih domena znanja, pri čemu se dva domena smatraju sličnim ako predstavljaju interpretacije iste formalne teorije. U oblasti kognitivnih nauka, ovaj proces ima tri koraka [69]: pribavljanje analogne situacije iz memorije; mapiranje - projektovanje elemenata i relacija iz jedne situacije u drugu; evaluacija - filtriranje pogrešnih zaključaka. U [70] i [71] su prikazani pristup kojim je omogućeno da se na osnovu polazeće ontologije i skupom odgovora na pitanja kompetentnosti izvede nova ontologija korišćenjem spoznaje po analogiji, dok je primena ove metode ilustrovana na oblast eksperimentisanja sa bežičnim mrežama unutar testbedova. Pitanja kompetencije su namenjena domenskim ekspertima i njihov cilj je utvrđivanje da li koncepti ili predikati već postoje u nekoj ontologiji.

Da bismo formirali analogiju, potrebno je mapirati elemente - koncepte i relacije izvornog domena na elemente ciljanog domena. Analogija se može uspostaviti bilo unutar jednog domena ili između različitih. Na ovaj način, definišemo mapiranje ontolgija, koje opisuje kako se elementi izvorne (postojeće) i ciljane (nove) ontologije povezuju. Dalje, koncepti mogu imati osobine koje se takođe nazivaju i svojstvima ili atributima. Što se tehnika mapiranja u ovom kontekstu tiče, postoje dve grupe tehnika – atributsko i relaciono mapiranje [70]. Kod atributskog mapiranja, analogija se fokusira na sličnost koja proizilazi iz samih svojstava koncepata, dok se relaciono mapiranje odnosi na sličnosti koje se tiču veza koncepata sa drugim elementima.

Sa druge strane, takozvana kompetentna pitanja [78][79] su efektivan pristup koji se tiče realizacije procesa razvoja ontologija, ali i prikupljanja zahteva koje ontologija treba da ispuni i pokrije sa ciljem rešavanja problema u nekom domenu. Ova pitanja su zadata u govornom jeziku, a njihova struktura prati određene šablone, tako da se odgovorima dobijaju informacije koje eksperti očekuju da dobiju na osnovu ontologije.

U literaturi postoje metode koje koriste analogije [70][80][81], ali i kompetentna pitanja [79]. Međutim, u postojećoj literaturi, još uvek nema ustanovljenih pristupa koji koriste njihovu

sinergiju. U ovoj disertaciji, implementiran je algoritam za razvoj ontologija na osnovu analogija i kompetentnih pitanja, oslanjajući se na prethodne radove [71][82].

U ovoj disertaciji, sličan pristup zasnovan na analogijama se koristi za izvođenje novih domenskih ontologija na osnovu ontologija srednjeg nivoa za apstraktnu reprezentaciju statičkih i dinamičkih aspekata inteligentnih sistema. Detaljan opis korišćenog mehanizma dat je u nastavku rada (podsekcija 3.4 - Mehanizam kreiranja domenskih ontolgoija uz pomoć analogija i kompetentnih pitanja).

2.8 Automatsko generisanje koda upotrebom ontologija

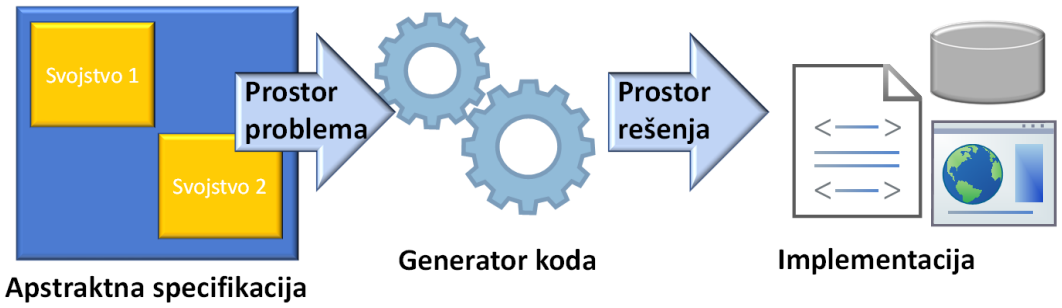
U današnje vreme, s obzirom na konkurentnost i ogromne potrebe za softverom u različitim oblastima, tržište zahteva brz, efikasan razvoj artefakata visokog kvaliteta, pri čemu bi njihovo održavanje trebalo biti što jednostavnije. To je podsticalo istraživače u ovoj oblasti da neprestano traže metode i alate koji omogućavaju da se na osnovu što višeg nivoa apstrakcije automatski generiše kod, uz minimalne intervencije, što predstavlja novu paradigmu razvoja softvera – automatsko programiranje. Automatsko programiranje je način računarskog programiranja koji se oslanja na mehanizme čiji je cilj da omoguće efikasnije programiranje - brže, uz manju količinu manuelno pisanog koda i sa manje strmom krivom učenja, pri čemu je nivo apstrakcije viši nego kod tradicionalnog pristupa.

Ovakav pristup teži generisanju kvalitetnog, održivog softverskog koda na osnovu intuitivnih apstrakcija vrlo visokog nivoa, sa konačnim ciljem skraćenja neophodnog vremena za implementaciju softverskog proizvoda i njegovo dostavljanje krajnjim korisnicima. Bez obzira na to što ove ideje datiraju više od pola veka unazad, univerzalno rešenje za automatsko programiranje još uvek nije pronađeno i predstavlja otvoren istraživački problem [83].

Jedna od disciplina automatskog programiranja jeste generativno programiranje, koje teži da iskoristi generatore koda za potpomaganje procesa razvoja aplikacija. Generativno programiranje se definiše kao projektovanje i implementacija programskih modula, koji se mogu kombinovati u generatorski program i iskoristiti sa ciljem generisanja visoko specijalizovanih i optimizovanih sistema namenjenih rešavanju specifičnih zadataka.

U ovom slučaju, ključnu ulogu ima generatorski program, koji kao ulaz prihvata model aplikacije visokog nivoa specificiran od strane korisnika, dok je rezultat generisanja programski kod njene implementacije. Nekoliko različitih tehnika se koristi u ovoj oblasti pored tradicionalne UML specifikacije, pri čemu je široko prihvaćen pristup modelovanja funkcionalnosti aplikacije kojim se beleže zajedničke, ali i prilagodljive karakterisike, zajedno

sa njihovim uzajamnim zavisnostima. Jedna funkcionalnost aplikacije je svojstvo relevantno sa tačke gledišta neke zainteresovane strane. U generativnom programiranju, domenski model se fokusira na mapiranja između prostora problema i rešenja. Dok se prostor problema bavi skupom funkcionalnosti koje treba da pruži grupa softverskih proizvoda, prostor rešenja se odnosi na implementaciju apstrakcija iz specifikacije. Generator koda u ovom kontekstu vrši preslikavanje specifikacije na odgovarajuću implementaciju. Proces upotrebe generatorskog programiranja je ilustrovan na **Slici 2.11**.



Slika 2.11 Proces generativnog programiranja

Upotreba ontologija u softverskom inženjerstvu ima sledeća četiri moguća pristupa [83][84][85][86]:

1) *ontologijama-vođen razvoj* – korišćenje tehnologija u toku faza razvoja softvera sa ciljem da se opiše domenski problem;

2) *ontologijama-omogućen razvoj* – upotreba ontologija u razvoju softvera sa ciljem pružanja podrške programerima u njihovim zadacima;

3) *ontološki-zasnovane arhitekture* – iskorišćavanje ontologija kao primarnih izvršivih artefakata; i

4) *ontologijama-omogućene arhitekture* – oslanjanje na ontologije sa ciljem pružanja podrške softveru tokom izvršenja.

Dalje, zavisno od faze kada se primenjuju, ontologije delimo na one koje se koriste tokom razvoja, dok su sa druge strane one na koje se softver oslanja u toku izvršenja.

U ovoj disertaciji, specifikacija sistema se zadaje primenom domenskih ontologija, koje se oslanjaju na zajedničku ontologiju višeg nivoa. Dalje, na osnovu njih se vrši generisanje koda, koji obuhvata više faktora relevantnih za životni ciklus inteligentnog sistema [84]: 1) razvoj – sam programski kod sistema 2) održavanje – komande za adaptaciju i prilagođavanje 3) pomoćni alati – poput grafičkih okruženja za modelovanje različitih aspekata i slično. **Tabela 2.9** daje pregled nekih od postojećih radnih okvira i pristupa za automatizovano generisanje

softverskog koda oslanjajući se na ontologije. Osim naziva i sažetog opisa rešenja, identifikovani su i korišćeni pristupi razvoja softvera upotrebom ontologija.

Tabela 2.9 Pregled rešenja za automatsko generisanje koda upotrebom ontologija

Naziv	Opis	Pristup
OnToCode [87]	Python paket koji omogućava programerima automatsko generisanje koda na osnovu skupa ontologija i šablona. Pruža API za učitavanje ontologija, ali i izvršavanje upita nad njima.	ontologijama-vođen razvoj
FIWARE [88]	Omogućava automatsko generisanje Java koda namenjenog IoT uređajima na osnovu onologija, sa ciljem olakšavanja integracija različitih uređaja. Semantički opis IoT uređaja i njihovih svojstava se koristi za generisanje middleware koda koji se tiče njihove integracije, dok pojedinosti vezane za načine pristupa podacima, karakteristične za individualne vrste IoT uređaja nisu pokrivene automatskim generatorom.	ontologijama-vođen razvoj ontologijama-omogućene arhitekture
Process Interaction Discrrete Event Simulation Ontology (PIDESO) [89]	Pristup za automatsko generisanje koda na osnovu ontologije, sa ciljem modelovanja i simulacije sistema. Instance su predstavljene ontologijama koje se iz XML formata upotrebom XSLT pravila pretvaraju u ciljani kod simulatora za Japrosim (rešenje otvorenog koda).	ontološki-zasnovane arhitekture
User Interface Adaptation (UIA) [90]	Ontologije se koriste za generisanje koda sa ciljem adaptacije korisničkog interfejsa na osnovu unapred zadatih pravila adaptacije.	ontologijama-omogućen razvoj
Algorithm Ontology (AO) [91]	Semantički radni okvir od nekoliko ontologija različitog nivoa apstrakcije za generisanje Java i PHP koda na osnovu ontološke reprezentacije logike koju taj algoritam implementira. Korišćen je Saxon XSLT za generisanje koda.	ontologijama-vođen razvoj

2.9 Dodaci

Ova podsekcija prikazuje dodatne mehanizme, koji se pored ontologija, upita i algoritama za generisanje koda koriste kao pomoćni prilikom realizacije eksperimenata vezanih za neke od studija slučaja. Što se primenjenih metoda tiče, u studijama slučaja se pojavljuju linearna optimizacija i nadgledano mašinsko učenje. Osim osnovne terminologije i koncepata koji se vezuju za ove metode, pomenute su i odgovarajuće tehnologije korišćene za implementaciju.

2.9.1 Linearna optimizacija

Linarna optimizacija (takođe poznata i kao linearno programiranje) predstavlja metodu čiji je cilj da pronađe najpogodniji mogući ishod (poput maksimalnog profita ili minimalne cene, zavisno od konteksta upotrebe), oslanjajući se na matematički model čiji su zahtevi izraženi u

obliku linearnih jednačina [92]. Praktično, ovo je tehnika optimizacije linearne funkcije cilja, pri čemu moraju biti ispunjena ograničenja u vidu linearnih jednačina ili nejednačina. Problem izražen u kanoničkom obliku se naziva i linearnim programom, kao što je prikazano u jednačini 4.1. U ovom slučaju, imamo x koji je zapravo vektor promenljivih koje treba odrediti (promeljiva odluke), c i b vektori poznatih koeficijenata, dok A predstavlja matricu poznatih koeficijenata.

$$\textit{Minimizirati } c^T x$$
$$\textit{Pod uslovom } Ax \leq b,\ x \geq 0 \tag{4.1}$$

Linearno programiranje se primenjuje u različitim oblastima – od poslovanja i ekonomije do inženjerstva. Što se inženjerskih primena tiče, posebno se ističu primene u oblasti planiranja, transporta, logistike, energetike i rutiranja. U ovom radu, linearno programiranje se primenjuje sa svrhom da realizuje globalne ciljeve koordinacije u studijama slučaja računarstva u magli, ali i pametnih energetskih mreža. Što se računarstva u magli tiče, koristimo ovu metodu sa ciljem postizanja optimalnog rasporeda Docker kontejnera koji odgovaraju zadacima koje dodeljujemo serverima.

Što se same implementacije tiče, koristimo programski jezik AMPL (A Mathematical Programming Language) [93]. Njegova svrha je definicija modela linearne optimizacije korišćenjem notacije, koja nalikuje na tradicionalnu algebru, sa proširenjima koji se omogućavaju intuitivnu implementaciju. Opredeljujemo se za AMPL zbog toga što kombinuje, ali i proširuje ekspresivnost sličnih algebarskih jezika za modelovanje (poput AIMMS, GAMS, LINGO i MPL), što ga čini generalnijim od njih, ali je ujedno i lak za upotrebu. Međutim, AMPL sam po sebi nema mogućnost rešavanja problema linearne optimizacija na osnovu zadatog modela, pa je neophodno da interaguje sa spoljašnjim alatima u ovu svrhu. Jedna od glavnih prednosti upotrebe AMPL-a jeste to što pruža uniforman interfejs ka alatima za rešavanje, što značajno olakšava rad, jer korisnik ne mora da vodi računa o svakom od njih zasebno. Za tu svrhu, u ovom radu se oslanjamo na pomoćni alat CPLEX[6], koji se koristi za rešavanje linearnih modela optimizacije primenom *simplex* metoda [94].

Dodatna objašnjenja o upotrebi i implementaciji linearne optimizacije za alokaciju softverskih komponenti, uzimajući u obzir i softverski definisane mrežne funkcije, ali i proaktivnu adaptaciju u sinergiji sa predikcijama mašinskog učenja, tema su prethodnih radova autora [95][96].

[6] https://www.ibm.com/products/ilog-cplex-optimization-studio

Osim linearne optimizacije, postoje i drugi pristupi rešavanju probelma alokacije resursa, kao što su razne heurističke i meta-heurističke metode, a među njima genetski algoritmi [95]. Međutim, u poređenju sa drugim pristupima, sa tačke gledišta ovog rada, linearna optimizacija ima prednosti u odnosu na druga rešenja [96]. Generalno, glavna prednost linearne optimizacije jeste u jednostavnijem procesu prilagođavanja na promene, što je od ključnog značaja, jer se ovaj rad bavi mehanizmima adaptacije i koordinacijom. Sa druge strane, u osnovi linearne optimizacije su modeli čija notacija se lako može generisati automatski na osnovu konceptualizacije domena kao što je ontologija, što dodatno olakšava proširivost pristupa, ali i smanjuje potrebno vreme i trud.

2.9.2 Nadgledano mašinsko učenje uz pomoć Weka biblioteke u Javi

Nadgledano mašinsko učenje obuhvata skup tehnika, metoda i algoritama čija je svrha predviđanje ciljanih vrednosti od interesa, posmatranjem velikog broja uzoraka kod kojih je tačan ishod već poznat. Generalno, parametre metoda mašinskog učenja možemo podeliti u dve velike kategorije: 1) *hiperparametri* – vrednosti koje imaju ulogu da utiču na sam proces obuke, i 2) *težinski parametri* – vrednosti koje se dobijaju kao ishod obuke (treninga).

Kod nadgledanog mašinskog učenja, trening skup podataka se odnosi na predstavljanje kolekcije uzoraka na osnovu kojih se tokom obuke podešavaju težinski parametri prediktivnog modela. Osim toga, imamo test podatke koji se koriste sa svrhom evaluacije prediktivnog modela koji je prethodno utreniran na opisani način. Između ostalog, u procesu nadgledanog mašinskog učenja figuriše i funkcija greške, čiji je uloga da izvrši procenu u kojoj meri je predviđena vrednost udaljena (po nekoj metrici) od očekivane. Na osnovu toga, vrši se štelovanje težinskih parametara proporcionalno veličini datih hiperparametara.

Dve najčešće primenjivane tehnike nadgledanog mašinskog učenja, koje se ujedno pominju u ovom radu jesu: 1) *klasifikacija* – predikcija oznake (labele) ili ishoda kategoričke prirode, i 2) *regresija* – predviđanje ishoda koji predstavlja kontinualnu numeričku vrednost. Po završetku faze obuke prediktivnog modela, vrši se procena kvaliteta dobijenih rezultata i to korišćenjem različitih vrsta metrika. Što se prethodno navedenih tehnika tiče, pregled odgovarajućih metrika je dat u **Tabeli 2.10**. Pri tome, imamo da oznake u formulama imaju sledeće značenje: 1) TP – pravi pozitivni, predikcije kod kojih imamo da se ishod slaže sa očekivanim 2) FP – lažno pozitivan, slučaj kada rezultat predikcije modela označava da uzorak pripada nekoj kategoriji, a očekivano je suprotno 3) TN – pravi negativan, odnosi se na na predviđanja gde je prediktovana vrednost da uzorak ne pripada nekoj kategoriji, a to se slaže sa

očekivanjem 4) FN – lažno negativan, ishod predviđanja da ne pripada kategoriji, a zapravo se očekuje suprotan slučaj. Pri tome, predikcija se odnosi na vrednost koju kao ishod predviđanja daje model mašinskog učenja, dok je očekivana vrednost ona koja je izmerena u stvarnom svetu.

Tabela 2.10 Metrike performansi modela nadgledanog mašinskog učenja

Tehnika	Metrika	Formula
Klasifikacija	Preciznost (*Precision*)	TP/(TP+FP)
	Opoziv *(Recall*)	TP/(TP+FN)
	Tačnost (*Accuracy*)	(TP+TN)/ (TP+FP+TN+FN)
	F1-score	2*(Precision*Recall)/ (Precision+Recall)
Regresija	Relativna greška	|(predikcija-očekivano)/očekivano| [%]
	Apsolutna greška	|očekivano-predikcija|
	Koren srednje kvadratne greške	$sqrt(\sum_{i=1}^{N}(predikcija-očekivano)^2/N)$

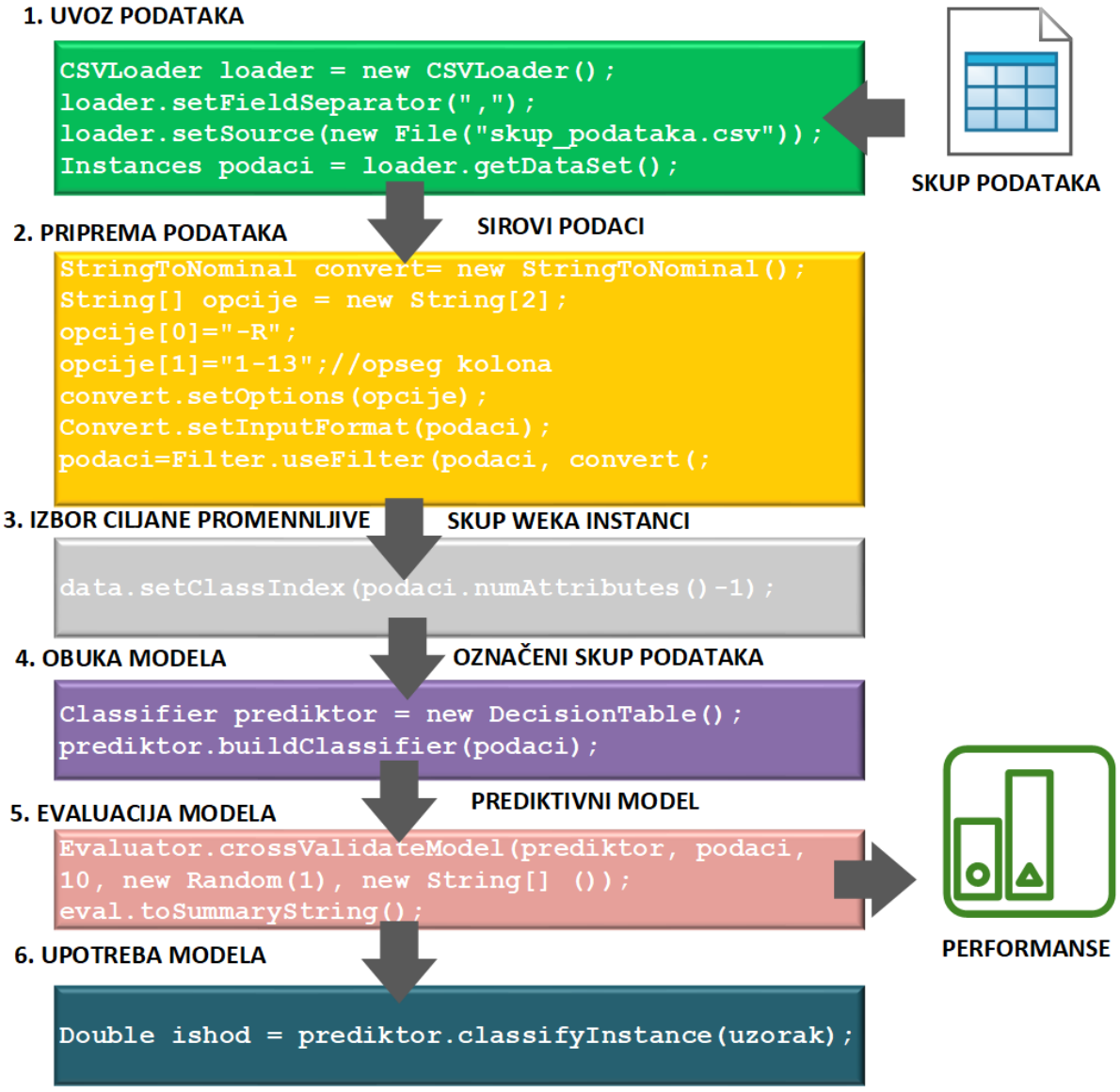
Dalje, po pitanju implementacije elemenata mašinskog učenja, u ovoj disertaciji se oslanjamo se na Java radni okvir pod nazivom Waikato Environment for Knowledge Analysis (Weka)[7]. Inicijalno, Weka je predstavljala skup alata namenjen prediktivnoj analitici podataka, predstavljen 1993., primarno namenjen obradi podataka iz oblasti poljoprivrede. Kasnije, od 1997. godine se usmerava kompletno ka Java ekosistemu. Danas, Weka predstavlja kolekciju alata i biblioteka čiji je cilj analiza i vizuelizacija podataka. Dostupni su po modelu GNU javne licence i besplatni. Međutim, osim vizuelnih alata i pristupa SQL bazama podataka, takođe pruža i programski interfejs (API) u Javi sa katalogom koji obuhvata poprilično veliki broj algoritama namenjenih rešavanju problema nadgledanog mašinskog učenja [97]. Za tu svrhu, mogu se koristiti tehnike koje su zasnovane na stablima odlučivanja (*J48*, *DecisionTable*, *DecisionStump*), zatim k-najbližih suseda (*IBk*), ali i različite varijante linearne regresije. Do odluke da se koristi Weka API u ovom radu, došlo se zbog brojnih prednosti, kao što su intuitivna sintaksa poziva funkcija i upotrebe klasa za nadgledano mašinsko učenje, sprega sa pomoćnim alatima i funkcionalnostima (učitvanaje CSV fajlova, rad sa skupovima podataka), portabilnosti (zahvaljujući Javi), a konačno i mogućnosti besplatne upotrebe uz širok skup dostupnih algoritama.

Dalje, na **Slici 2.12** su prikazani ključni koraci prilikom procesa nadgledanog mašinskog učenja oslanjajući se na Weka API u Java programskom jeziku[8]. Dodatno, krucijalni delovi

[7] https://www.cs.waikato.ac.nz/ml/weka/
[8] https://waikato.github.io/weka-wiki/use_weka_in_your_java_code/

koda su dati za svaki od koraka. Prvo se odgovarajući skup podataka importuje iz Comma Separated Values (CSV) formata, upotrebom *CSVLoader* objekta za ovu svrhu. Posle toga, sirovim podacima se može pristupati upotrebom *Instances* objekta. Opciono, moguće je izvršiti neophodne transformacije sa ciljem pripreme skupa podataka, kao što je konverzija stringova kategoričkih ishoda u format nominalnih vrednosti koji je podržan od strane Weka klasifikatora. Sa ovim ciljem se primenjuje filter na dati opseg kolona (u primeru od prve do trinaeste kolone), korišćenjem *StringToNominal* konverzionog objekta.

Posle toga, potrebno je odabrati koje su među kolonama zapravo ulazne, a koje izlazne (ciljane) promenljve. Dalje, skup podataka se u ovako označenoj formi prosleđuje kao ulaz u metodu koja konstruiše prediktivni model. Nakon toga, moguće je obaviti trening prediktivnog modela za taj, odabrani skup podataka. Po završetku faze treinga, potrebno je izvršiti evaluaciju modela i posmatrati njegove performanse kroz perspektivu neke metrike, kao što su tačnost i F1 za klasifikaciju, a za regresiju srednja relativna greška. Konačno, na ovakakv način obučen prediktivni model se može dalje koristiti unutar aplikacija i servisa sa ciljem predikcije ishoda na osnovu novih, prethodno neviđenih uzoraka podataka.



Slika 2.12 Koraci prilikom implementacije nadgledanog mašinskog učenja uz pomoć Weka API u Javi [97]

2.9.3 Ethereum blockchain i Solidity pametni ugovori

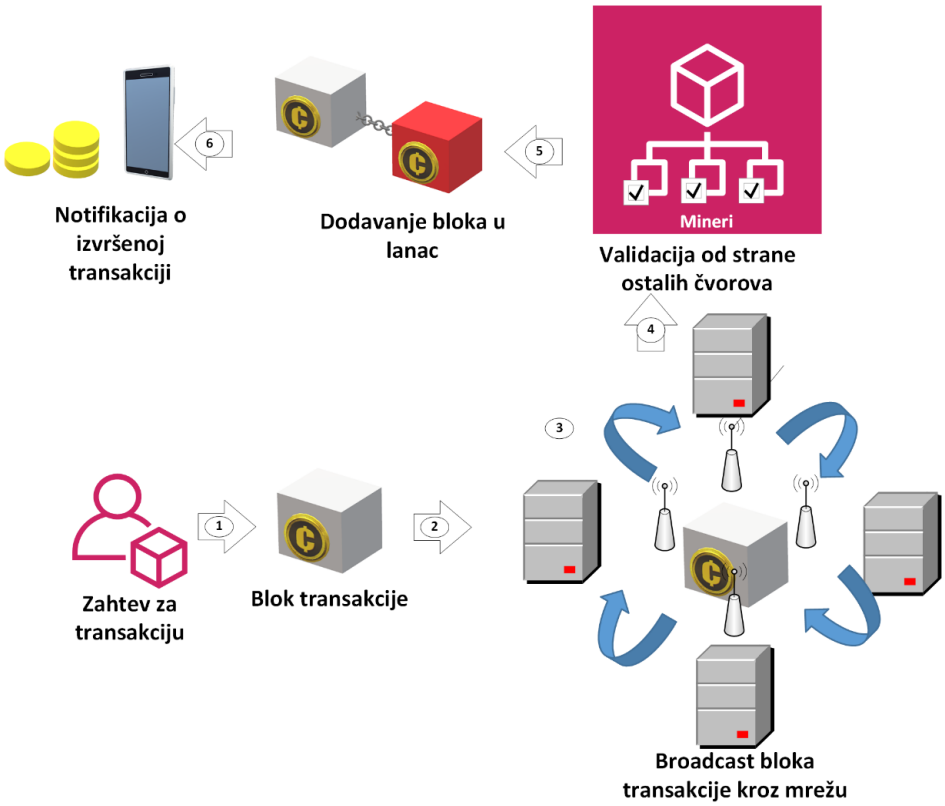
Na blockchain se može gledati kao na globalnu evidenciju obavljenih transakcija, pri čemu su sve informacije javno dostupne i mogu se verifikovati [98]. Odgovarajuća struktura podataka za vođenje evidencije se naziva glavna knjiga ili registar (engl. *ledger*), a on je zapravo i osnova konkretnih blockchain tehnologija, poput Bitcoin-a [99] i Ethereum-a [100]. Ledger daje mogućnost zapisivanja i prenosa informacija o transakcijama na transparentan, bezbedan i sledljiv način. Sastoji se od sekvence blokova, koji drže informacije o transakcijama, koji se mogu nadovezivati jedan na drugi, ali ne i menjati. Ledger se čuva decentralizovano, u okviru distribuiranog sistema koji se oslanja na *peer-to-peer* mrežu čvorova. Svaki čvor predstavlja

korisnika blockchain-a koji se identifikuje jedinstvenom alfanumeričkom adresom, što održava anonimnost, ali istovremeno i pruža transparentnost informacija.

U kontekstu blockchain tehnologija, transakcija predstavlja transfer vrednosti ili vlasništva dobara (materijalnih ili nematerijalnih), u obliku digitalnih tokena, koji se razmenjuju između pošiljaoca i primaoca, a informacije o obavljenoj transakciji čuvaju u vidu blokova na distribuiranom ledger-u. Pri tome, svaki blok poseduje kriptografski hash prethodnog i vremensku markicu, sa ciljem obezbeđivanja da niko ne može da ih obriše ili modifikuje nakon dodavanja u ledger. Prema tome, što više blokova imamo, sam ledger je bezbedniji i pouzdaniji.

U svakom centralizovanom sistemu, postoji neko ko poseduje autoritet održavanja i ažuriranja podataka. Međutim, distribuirani blockchain funkcioniše kao decentralizovani samoregulišući sistem na globalnim razmerama, bez bilo kakvog centralnog autoriteta, već se oslanja na mrežu koja obuhvata hiljade učesnika koji rade verifikaciju transakcija. Prema tome, u mreži čvorova postoje i kopači (engl. *miner*), koji su zaduženi za verifikaciju transakcija izvršavanjem zadataka izračunavanja, pri čemu dobijaju kao nagradu tokene ukoliko prvi reše zadatak. Za ovu svrhu se uglavnom koriste skupe grafičke kartice (GPU) ili specijalizovani procesori, pri čemu verovatnoća pronalaženja rešenja zavisi od brzine izračunavanja hardvera koji je na raspolaganju. Iako može dovesti do profita, zarada na ovakav način zahteva investicije u moćan hardver, ali i veliku količinu električne energije koja se troši od strane tog hardvera, što dovodi do brojnih diskusija na temu održivosti sistema zasnovanih na ovoj tehnologiji.

Međutim, da bi se održala konzistentnost stanja ledgera, potrebno je primeniti mehanizam, koji omogućava da se svi učesnici slože na efikasan, pouzdan i siguran način. Za tu svrhu blockchain tehnologije se oslanjaju na algoritme konsenzusa, pri čemu se među njima ističu dva: *proof of work* (PoW) i *proof of stake* (PoS) [98][101]. U slučaju PoW, potrebno je da jedna strana pokaže ostalima da je određena količina truda potrošena za izračunavanje ili rešavanje nekog problema. Taj zadatak ne treba da bude trivijalan, ali, sa druge strane, treba da bude lako proverljiv od strane drugih učesnika. Bitcoin i Ethereum se oslanjaju na ovaj algoritam, ali je kritikovan zbog potrošnje ogromnih količina električne energije. Sa druge strane, ideja PoS je da učesnik ima veći prioritet da verifikuje blok ukoliko poseduje veći broj tokena. Trenutno se radi na Ethereum-u 2.0 koji ima za cilj da se pređe na PoS mehanizam umesto trenutno korišćenog PoW konsenzusa. Na **Slici 2.13** data je ilustracija tipične blockchain arhitekture.



Slika 2.13 Beleženje informacija o transakciji upotrebom blokčejna

U ovoj disertaciji, u okviru studija slučaja se za pojedine inovativne domenske scenarije koriste pametni ugovori namenjeni Ethereum blockchain platformi, pa će biti prikazano više detalja o tome.

Pametni ugovor predstavlja računarski program ili transakcioni protokol, koji ima za cilj da automatski izvršava, kontroliše ili dokumentuje relevantne akcije i događaje vezane za stavke sporazuma realizovanog tim ugovorom. Najzastupljeniji jezik za implementaciju pametnih ugovora je Solidity [102][103] (prvi put se javlja 2014. godine), koji je prvenstveno namenjen Ethereum platformi. On predstavlja objektno-orijetisan programski jezik visokog nivoa, a njegovi programi utiču na stanje naloga korisnika u okviru Ethereum blockchain platofrme. Statički je tipiziran, što znači da tipovi promenljivih moraju biti poznati za vreme kompajliranja pametnog ugovora. Sa druge strane, karakteristično je i da funkcije mogu imati više povratnih vrednosti. Njegova sintaksa se najviše zasniva na ECMAScript-u, a razlog je, po rečima autora, da bi bio blizak web developerima. Dalje, ovaj jezik podržava nasleđivanje (uključujući i višestruko nasleđivanje), biblioteke i kompleksne korisnički-definisane tipove (hijerarhijska mapiranja i struct-ove).

Generalno, Solidity pametni ugovor se sastoji od sledećih delova : 1) promenljive stanja, 2) definicije struktura, 3) definicije modifikatora pristupa, 4) deklaracije događaja, 5) definicije

enumeracija, i 6) definicije funkcija. U nastavku će biti dat pregled osnovnih elemenata i karakteristika Solidity jezika koji su relevantni za studije slučaja prikazane u disertaciji.

Za pametni ugovor se koristi ključna reč *contract*, a analogan je klasi u ostalim objektno-orijentisanim programskim jezicima. Na početku svakog fajla u kome se definiše Solidity pametni ugovor ide pragma direktiva koja označava minimalnu verziju kompajlera koja je neophodna. Dalje, *import* direktiva se koristi za uvoženje biblioteka i drugih fajlova sa Solidity kodom.

Unutar contract bloka, koji podseća na implementaciju klase u objektno-orijentisanim programskim jezicima, date su definicije funkcija, ali i promenljive od značaja za implementaciju konkretnog pametnog ugovora. Praktično, ove funkcije su zapravo srž samih transakcija koje pametni ugovor realizuje, dok promenljive predstavljaju relevantna stanja na koja utiče izvršenje funkcija ugovora. Postoje četiri modifikatora funkcija, koji se mogu primeniti i na promnljive: *internal*, *external*, *public* i *private*. Ako modifikator nije specificiran, podrazumeva se internal. To znači da se funkcija može koristiti samo u okviru ugovora u kome je definisana i njegovog konteksta (tu spadaju i nasleđeni ugovori ili interne biblioteke) i nikako od spolja. Sa druge strane, external funkcije poseduju dva dela: adresu i potpis funkcije. Prema tome, external funkcije se mogu koristiti kao parametri spoljašnjeg poziva ili vratiti kao rezultat spoljašnjeg poziva. Ukoliko je modifikator *public*, onda je funkcija deo javnog interfejsa ugovora, a može se pozvati bilo iz samog ugovora, bilo od strane nekog drugog, spolja. Za javne promenljive se automatski generišu odgovarajuće *get* i *set* metode. Konačno, privatne funkcije i promenljive su vidljive samo u trenutnom ugovoru. Njima se ne može pristupiti ni iz nasleđenih ugovora i nisu deo interfejsa pametnog ugovora.

Zatim, po pitanju prava pristupa podacima i njihove modifikacije, funkcije mogu biti: *constant*, *pure* ili *payable*. *Constant* funkcije ne mogu modifikovati stanje blockchain-a, ali mogu čitati stanje i vratiti pročitane vrednosti onom koji je pozvao tu funkciju. Ove funkcije ne mogu da menjaju vrednost promenljivih, kreiraju nove ugovore, okidaju događaje, niti da zovu bilo koju drugu funkciju koja utiče na stanje blockchain-a. *Pure* funkcije ne mogu ni da čitaju, niti da modifikuju stanje blockchain-a. Sa druge strane, *payable* funkcije su one koje mogu da prihvate ether od strane pozivaoca. Prema tome, ukoliko pozivalac ne priloži odgovarajuću količinu *ether* (ETH) tokena, može se dogoditi da se funkcija ne izvrši, a primer bi bio ostavljanje depozita.

Osim toga, u kontekstu Ethereum platforme se često pominje i *gas*. To je jedinica izračunavanja cene transakcije u okviru Ethereum platforme, koja predstavlja broj tokena (obično u jednici *Gwei*, što iznosi 10^{-9} ETH), koje pošiljalac mora da plati miner-u koji ubacuje

transakciju obavljenu pametnim ugovorom u blockchain. Svaka operacija koja se može izvršiti u okviru Ethereum platforme ima dodeljenu cenu gasa, koja je aproksimativno proporcionalna količini resursa (za izračunavanje i skladištenje) koje čvor mora da potroši da bi je izvršio. Pre transakcije, pošiljalac mora da specificira graničnu vrednost gasa i cenu gasa.

Dalje, što se naredbi tiče, podržan je *if/else*, a od petlji *while* i *for*, dok Solidity ne poseduje *switch* i *goto*. Međutim, imamo da su *continue* i *break* podržani. Solidity poseduje i *event* mehanizam, koji služe za signalizaciju klijentskim aplikacijama da se desio određeni događaj tokom izvršavanja funkcije pamentnog ugovora. Događaje je neophodno prethodno u ugovoru deklarisati, zajedno sa parametrima i njihovim tipovima. Ukoliko se događaj signalizuje uz pomoć ključne reči *emit*, onda se njegovi argumenti prosleđuju u log transakcije i čuvaju na blockchain-u. U suprotnom, samo navođenjem poziva događaja sa argumentima se informacije ne čuvaju trajno.

Što se upotrebe blockchain-a tiče, tokom poslednjih godina, došlo je njegovog usvajanja u mnogim domenima upotrebe, osim finansija i plaćanja [101]. Recimo, u [104] je prikazana primena blockchain-a sa svrhom izgradnje poverenja u nadzoru poslovnih procesa, kod kojih učestvuje više različitih organizacija. Sa druge strane, [105] i [106] razmatraju primenu blockchain tehnologija sa ciljem podrške zaštiti od širenja COVID-19 zarazne bolesti, pri čemu su obuhvaćena dva slučaja korišćena: nadzor kontakata sa zaraženim osobama i evidencija vakcinisanja. U ovoj disertaciji, pametni ugovori se parametrizuju za izvršavanje transakcija u kontekstu trgovine električnom energijom u prilagodljivim pametnim mrežama. Primer Solidity pametnog ugovora iz ove studije slučaja je dat na **Slici 2.14**.

```
pragma solidity ^0.4.21;
contract TradeEnergy{
      event Sent(address sender, address receiver, uint amount, uint
distirbution_cost, uint generation_cost);
      uint total;
      uint token_price;
      function trade() public {
            total=(amount*Generation_cost+distribution_cost)/token_price;
            if (balances[sender] < total) return;
            balances[sender] -= total;
            balances[receiver] += total;
            emit Sent(sender, receiver, amount, distribution_cost,
generation_cost);
      }
}
```

Slika 2.14 Primer Solidity pametnog ugovora za trgovinu električnom energijom između prosumer-a u pametnj električnoj mreži

3 SEMANTIČKI PRISTUP GENERISANJU KODA KOD DOMENSKI-SPECIFIČNIH INTELIGENTNIH SISTEMA

3.1 Definicija problema

Cilj predloženog radnog okvira je automatizovano generisanje programskog koda kojim se definišu struktura i ponašanje inteligentnog informacionog sistema *s*, polazeći od semantičkog opisa dva ključna aspekta:

1) topologije sistema T – koja predstavlja skup komponenti $(c_1, \ldots, c_N)$, i

2) skup pravila adaptacije (strategija adaptacije) A – $(ar_1, \ldots, ar_N)$.

Prema tome, svaki sistem opisan predloženim radnim okvirom razmatramo kao uniju topologije T i strategije adaptacije A - $s: T \ U \ A$.

Dalje, svakoj komponenti c_i koja pripada topologiji T se pridružuje šablon koda t_i, što se može predstaviti mapiranjem: $(c_i->t_i)$. Dalje, svakom od šablona t_k se dodeljuje skup karakteristika $(p_1 \ldots p_M)$. Cilj procedure za generisanje koda je parametrizovanje svake od karakteristika p_u, odgovarajućim vrednostima parametara v_u, što možemo prikazati kao skup mapiranja, gde svako pojedinačno mapiranje ima formu: $(p_u->t_u)$. Pri tome, šablon koda ima sledeći oblik:

$$početak\_šablona \ \ldots ?p_1 \ldots ?p_2 \ \ldots \ ?p_L \ \ldots kraj\_šablona \tag{3.1}$$

Na **Slici 3.1** koja je kasnije prikazana, popunjavanje šablona dato formulom 3.1 se odvija u koraku 9.

Procedura generisanja koda g ima za cilj da svaki od parametara $?p_u$ zameni odgovarajućim vrednostima na osnovu datih mapiranja, što možemo označiti kao $g(t_u, m_u) = parametrized\_out$. Skup nadovezanih parametrizovanja šablona predstavlja izlaz algoritma generisanja, što u slučaju statičkih aspekata predstavlja topologiju sistema.

Sa druge strane strane, za dinamičke aspekte sistema se polazi od semantičkog opisa pravila adaptacije, pri čemu svako od pravila opisuje komande $(cmd_1, \ldots, cmd_D)$ koje se aktiviraju kada su ispunjeni uslovi zadati sa $cond_r$. Svakom uslovu mogu biti dodeljeni SPARQL upit q_r ili događaj e_r, koji predstavljaju kontekst pod kojim se aktivira skup komandi, što se može predstaviti kao $context_r(cond_r->\{q_r, e_r\})$. Svaka od komandi, slično poput šablona koda,

poseduje skup parametara koji se tokom generisanja koda popunjavaju na način kao što je već prikazano u formuli 3.1.

Prema tome, pravilo adaptacije a_r se može predstaviti u vidu formule na sledeći način:

a_r: If (q_r je ispunjen ili e_r se desio)onda izvršiti skup komandi ($cmd_1 \dots cmd_D$)
nad komponentama ($comp_1 \dots comp_p$) *(3.2)*

Prethodno objašnjeni mehanizam provere pravila adaptacije odgovara koraku 7 na **Slici 3.1**, dok se generisanje komandi odvija u koraku 9.

Prema tome, proces adaptacije *AP* se može definisati kao prelazak stanja sistema iz početnog stanja *S* u novo stanje S' primenom skupa pravila adaptacije *A* nad sistemom *s* kada dođe do događaja *e*:

$S' = AP(S, e, A)$ *(3.3)*

Što se tiče primenjenog mehanizma zaključivanja za realizaciju procesa adaptivnog ponašanja, koristi se lančanje unapred [107]. Ovaj princip predstavlja pristup zaključivanja odozdo naviše (engl. *bottom-up*), tako što od polazeći od trenutnog stanja sistema proverom ispunjenosti *if*-dela pravila adaptacije (uslova i događaja) teži da dođe do ciljanog generisanjem koda definisanog u *then*-delu (videti formulu 3.2). Za svako od pravila adaptacije iz skupa *A* se proverava ispunjenost. Ukoliko je uslov pravila adaptacije istinit, onda se isčak koda iz *then*-dela parametrizuje i nadovezuje (kao u formuli 3.1) na konačni rezultat realizovane strategije adaptacije. Ovakav pristup se obično naziva i podacima-vođenim zato što se od inicijalnoog stanja do ciljanog stiže na osnovu dostupnih podataka o trenutnom stanju sistema.

Konačno, što se procesa koordinacije tiče, ona predstavlja specifičan slučaj dinamičkog ponašanja sistema, koji ima za cilj da uskladi veći broj akcija *(a_1...a_M)*. Ove akcije se dele u disjunktne podskupove, tako da svaki od ovih podskupova akcija izvršava različiti izvršilac (*executor* – x_k) , sa svrhom realizacije nekog zajedničkog cilja razmatranog sistema (domenski cilj - G_m). Sledećom formulom je prikazan opis procesa koordinacije, za slučaj kada je cilj G_m definisan sa ukupno *M* akcija, pri čemu je za njihovo izvršenje odgovorno *N* izvršilaca *(x_1...x_N)*:

G_m: $(x_1 -> (a_1 \dots a_m),\ x_2 -> (a_{m+1} \dots a_{m+i}), \dots\ x_N -> (a_{k+i+1} \dots\ a_M))$ *(3.4)*

Mehanizam opisan ovom formulom se odvija u koraku 11 na **Slici 3.1**.Kod koordinacije se koristi zaključivanje lančanjem unazad [107]. Za razliku od lančanja unapred, ova metoda koristi zaključivanje odozgo naniže (engl. *top-down*). Ovakav mehnizam zaključivanja je adekavatan kada cilj sistema znamo unapred, što je slučaj procesa koordinacije, Sa druge strane, kod adaptacije imamo potencijalno više mogućih ciljeva, zavisno od ispunjenih uslova adaptacije u nekom trenutku.

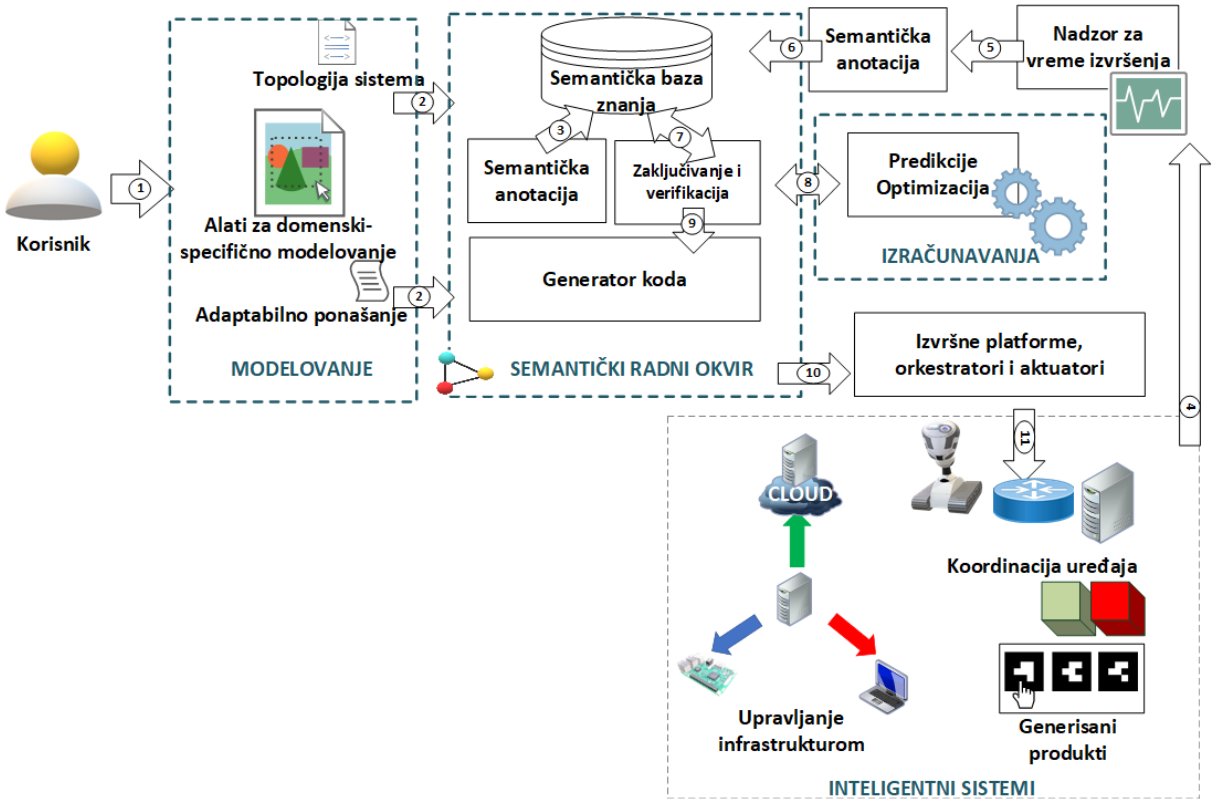
3.2 Princip rada predloženog pristupa

Na **Slici 3.1** je prikazana arhitektura predloženog pristupa i interakcija komponenti na visokom nivou u fazi izvršenja (engl. *run-time*). Na početku, domenski eksperti definišu ciljeve koje žele da ostvare za određeni scenario primene sa svrhom rešavanja nekog problema upotrebom inteligentnog sistema (korak 1). Za ovu svrhu se koriste vizuelni alati sa intuitivnom domenski-specifičnom notacijom, automatski generisani upotrebom semantičkog radnog okvira. Kao izlaz iz ovog koraka se generiše fajl modela rešavanja nekog domenski-specifičnog problema, koji obuhvata opis topologija, ali i ponašanje sistema kao odgovor na promene nastale u okruženju (korak 2). Sa druge strane, nadzire se ponašanje sistema u toku izvršenja i detektuju nastali događaji i promene od interesa (koraci 4 i 5). Nakon toga, model sistema (korak 3) i informacije o ponašanju sistema (korak 6) se transformišu u semantičku reprezentaciju anotiranu na osnovu domenske ontologije, pogodnu za primenu procesa semantičkog zaključivanja (korak 7) i pribavljanja relevantnih informacija za dalje generisanje koda (korak 9). Sa druge strane, semantička reprezentacija pruža i dodatne pogodnosti, poput jednostavne verifikacije sistema na osnovu SPARQL upita, što daje mogućnost provere da li važe zadata pravila na nivou kreirane instance (korak 7). Ovakva reprezentacija kreiranog modela se čuva u okviru semantičke baze znanja, dok se za zaključivanje oslanjamo na skup SPARQL za realizaciju *if-then* pravila koje inkorporišu ekspertsku intuiciju, sa ciljem automatizacije procesa. U *if* delu, SPARQL upiti se koriste za evaluaciju uslova. Dalje, ako je uslov ispunjem, onda se u *then* delu drugi SPARQL upiti koriste za pribavljanje vrednosti koji će biti ubačene u ishod generisanja koda.

Prilikom zaključivanja, uzimaju se u obzir različiti aspekti relevantni za domenski-specifični sistem: topološka organizacija i struktura, zatim prikupljeni podaci o okruženju (vrednosti sa senzora ili događaji detektovani kao ishod analize podataka) i ciljevi definisani od strane eksperata.

Zatim, na osnovu ishoda semantičkog zaključivanja, detektuje se određeni kontekst i zavisno od toga vrši automatsko generisanje koda sa svrhom ostvarivanja zadatog cilja. Tokom generisanja koda, kao sredtsvo za izračunavanje pojedinih vrednosti se koriste neke od pomoćnih metoda i alata (korak 8): prediktivni modeli nadgledanog mašinskog učenja (klasifikacija i regresija) i višeciljna optimizacija za rešavanje problema alokacije komponenti. Osim toga, metode klasifikacije mašinskog učenja se primenjuju i kao pomoćno sredstvo za semantičku anotaciju prikupljenih podataka o okruženju uz pomoć senzora.

Konačno, generisani kod i komande kao rezultat procesa automatskog generisanja se dostavljaju odgovarajućim uređajima i platformama (korak 10) da bi postigao željeni cilj, što zavisi od domena, ali i od konkretnog scenarija upotrebe. Sa jedne strane, to mogu biti softverski artefakti (izvorni kod aplikacije), vrednosti za paramterizovanje pojedinih programa (recimo, za pametne ugovore) ili njihovih metoda, ali i konkretne komande namenjene raznolikim uređajima i aktuatorima (IoT, roboti). Ove komande mogu aktivirati akcije jednog, ali i više sinhronizovanih, koordinisanih uređaja da bi se efikasnije ostvario zadati cilj (korak 11).



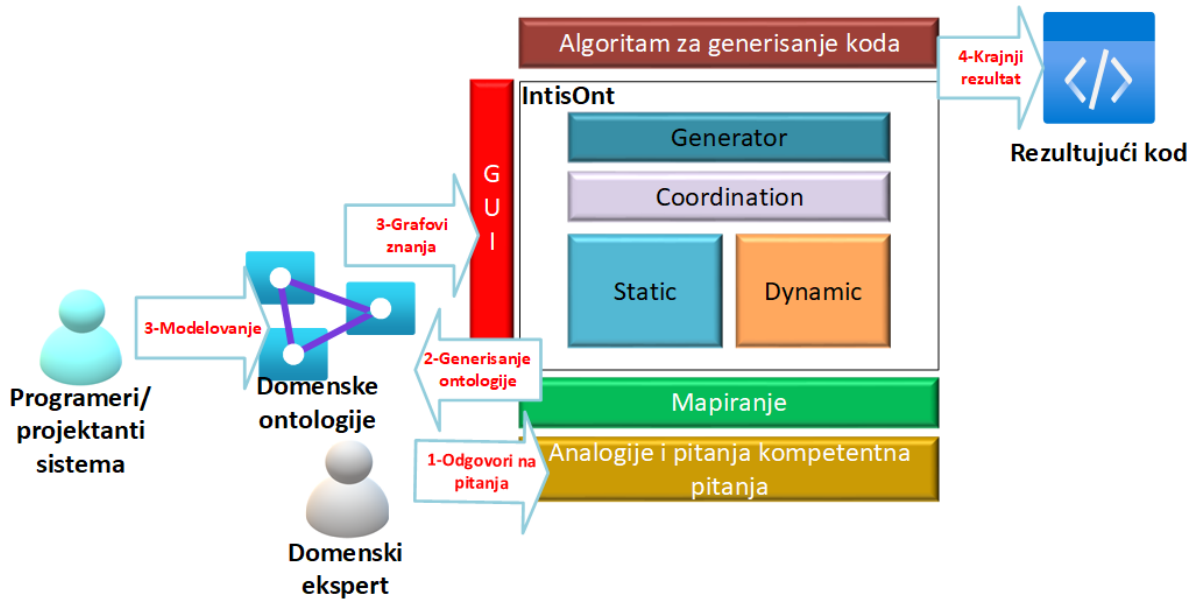
Slika 3.1 Interakcija komponenti na visokom nivou tokom izvršenja: 1-Modelovanje 2-Domenski-specifični aspekti 3, 6- Semantički tripleti 4-Događaji 5-Poruke 7-SPARQL upiti (formula 3.2) 8-Rezultati pomoćnih metoda (predikcije kod mašinskog učenja, matrice alokacije kod optimizacionih modela za raspoređivanje resursa) 9-Ishodi provera uslova (formula 3.2) i pribavljeni parametri (formula 3.1) 10-Rezultati generisanja koda 11-Komande namenjene ciljanim uređajima (formule 3.3 i 3.4)

3.3 Ontološki radni okvir

3.3.1 Pregled ontologija

Ova podsekcija daje detaljni opis radnog okvira ontologija za inteligentne informacione sisteme *Intelligent System Ontology* (IntisOnt) na osnovu kojeg se dalje generišu domenski-

specifične ontologije, dalje iskorišćene za automatsko generisanje koda. Srž ovog ontološkog radnog okvira obuhvata tri aspekta relevantna za rešavanje problema iz nekog domena: 1) statički aspekti – obuhvata topologiju i struktura sistema, što obuhvata način na koji su komponente povezane u sistem, njihova svojstva i karakteristike 2) dinamički aspekti – sirovi rezultati merenja iz okruženja ili detektovani događaji do kojih je došlo u okruženju na osnovu prikupljenih podataka 3) domenski ciljevi – opis ishoda koji domenski eksperti žele postići sa ciljem rešavanja nekog problema. Prema tome, što se opisa sistema tiče, ontologija je upravo na osnovu te podele, razdvojena u dve celine, koje predstavljaju fundamentalne ontologije po svojoj prirodi: *Static* (strukturalni koncepti) i *Dynamic* (obuhvata dinamičke aspekte). Na osnovu ovih ontologija, uz pomoć analogije, odgovarajući na kompetentna pitanja, domenski eksperti dolaze do specifičnih ontologija za rešavanje problema od interesa. Rezultat ovog procesa ontološkog mapiranja jesu domenske ontologije. Osim toga, u okviru IntisOnt radnog okvira postoji i *Generator* ontologija, koja enkapsulira aspekte koji pomažu generisaju koda na osnovu grafova znanja definisanih u skladu sa prethodno dobijenim domenskim ontologijama. Za generisanje koda se koriste mehanizmi koji su kasnije opisani, a korisnici (uglavnom projektanti sistema ili programeri) mogu definisati grafove znanja korišćenjem alata koji se oslanja na vizuelnu notaciju za modelovanje (u ovom slučaju Node-RED). Dalje, četvrta ontologija od značaja je *Coordination* ontologija, koja služi da uskladi dinamičke aspekte više elemenata u sistemu sa cilje m realizacije zajedničkog domenskog cilja upotrebom više od jednog elementa. Postojanje elemenata ove ontologije nije neophodno za generisanje koda, već je opciono. **Slika 3.2** ilustruje strukturu onotloškog radnog okvira, njegovih ontologija i uloga koje odgovarajuće ontologije imaju.



Slika 3.2 Pregled ontološkog radnog okvira i pomoćnih procesa

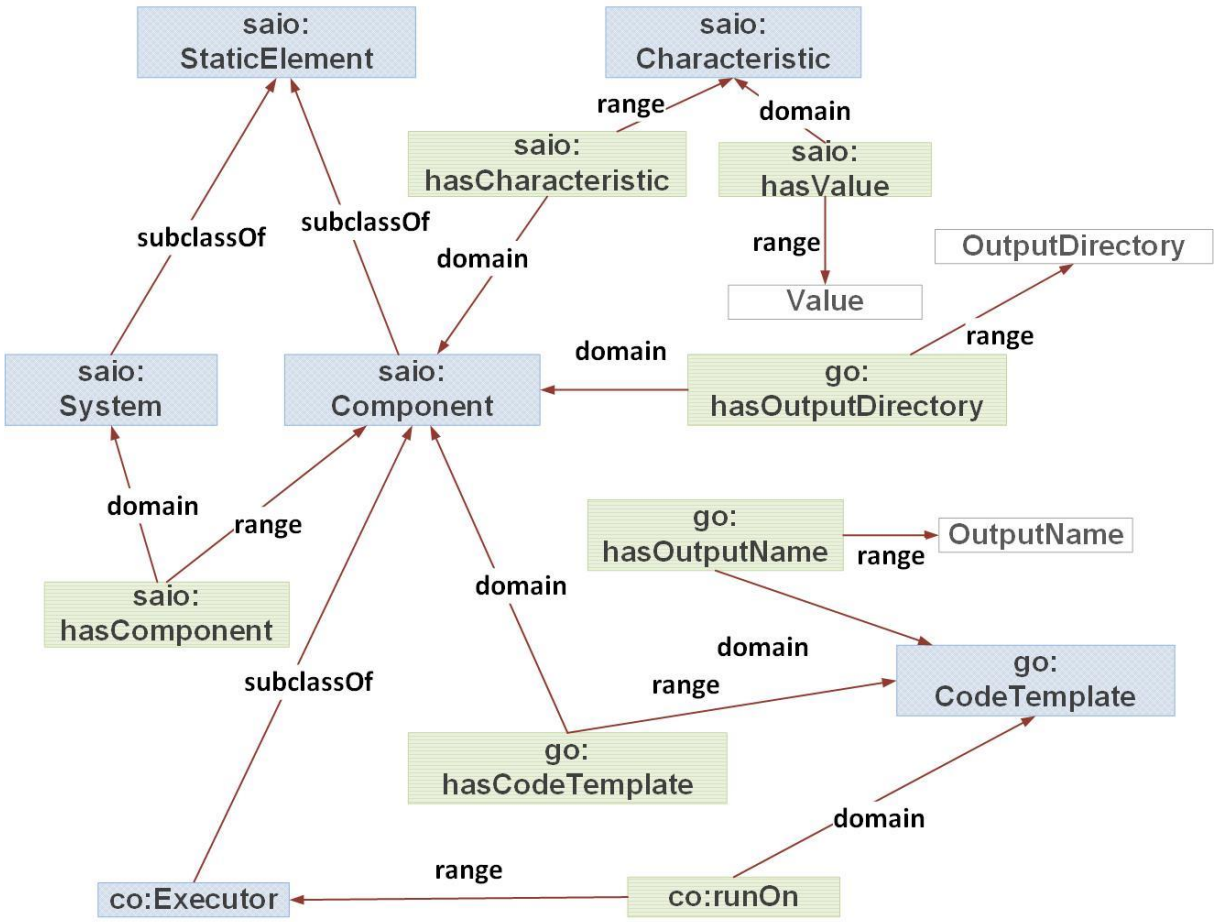
3.3.2 Detaljni opis

Predloženi ontološki radni okvir inspirisan je višim ontologijama SUMO i DOLCE. Što se tiče osnovnih elemenata koji figurišu (objekti, atributi, relacije, događaji, procesi), po tome je sličan SUMO [64] ontologiji, dok po pitanju podele na statičke i dinamičke elemente dolazi inspiracija iz DOLCE [61][62] ontologije, što redom odgovara endurantnim i protrajnim entitetima.

Prvo ćemo razmotriti statički pogled na sistem, nazvan *Static Aspects Intis Ontology* (SAIO) (ilustrovan u okviru **Slike 3.3**). Njegova osnovna uloga je da opiše topološku strukturu sistema – komponente i njihove karakteristike. Koncept najvišeg nivoa statičkog pogleda na inteligentni sistem ove ontologije jeste *System*, koji predstavlja složen statički element. Dalje, System se sastoji od komponenti (*Component*), koje su prosti statčki elementi. I sistem i komponenta predstavljaju statičke elemente i podklase su *StaticElement*. Svaka od komponenti ima svoje karakteristike, dok karakteristika (*Characteristic*) ima konkretnu vrednost. Osim toga, kao karakteristika jedne komponente se može naći i neka druga komponenta, pri čemu se na taj način opisuju aspekti kompozicije koji su potrebni za opis složenih komponenti, koje neku drugu mogu sadržati kao svoj deo. Što se tiče odnosa sa postojećim višim ontologijama, *StaticElement* je podklasa endurantnog entiteta iz perspektive DOLCE ontologije (odnos *subClassOf*), dok predstavlja podklasu fizičkog objekta SUMO ontologije (odnos *subClassOf*). *SSystem* takođe odgovaraju fizičkom objektu (odnos *subClassOf*). Sa druge strane *Characteristic* odgovara atributu iz SUMO ontologije (takođe odnos *subClassOf*).

Što se proširenja neophodnoh za generisanje koda tiče, oni su povezani sa SAIO, ali pripadaju *Generator* ontologiji (GO). Što se statičkih aspekata tiče, komponenta poseduje i šablon koda (*CodeTemplate*), koji se može koristi kao osnova za parametrizovanje vrednostima karakteristika tokom procesa automatskog generisanja koda. Karakteristike u šablonu koda imaju specijalnu oznaku <*karakteristika*> , pri čemu njihov naziv mora da se slaže sa nazivima koncepata koji su povezani sa komponentama preko veze *hasCharacteristic*. Unutar procedure generisanja koda, za svaku od komponenti iz sistema se prvo pribavlja njen šablon koda, a zatim u njemu pronalaze karakteristike, pa svaka od njih konačno pribavlja izvršenjem SPARQL upita. Praktično, može se desiti da jedna ista komponenta ima više šablona koda istovremeno. Dalje, da bi se povećala upotrebljivost generisanih rezultata, Generator ontologija proširuje SAIO i aspektima koji se tiču organizacije generisanih produkata, s obzirom da elementi koji čine sistem mogu biti prilično heterogeni i izvršavati se na različitim lokacijama. U tom kontekstu, za svaku od komponenti je moguće postaviti naziv izlaznog direktorijuma (*OutputDirectory*), dok se za svaki od dodeljenih šablona može odabrati tačan naziv generisanog fajla (*OutputName*) koji nastaje parametrizacijom.

Što se tiče odnosa sa SUMO ontologijom, *hasCharacteristic, runOn, hasValue, hasComponent, hasOutputDirectory, hasOutputName* odgovaraju relaciji (odnos *subPropertyOf*);*OutputName* i *OutputDirectory* atributu (*odnos subClassOf*)., dok *CodeTemplate* odgovara apstraktnom objektu (*odnos subClassOf*).



Slika 3.3 Static Aspects Intis Ontology – SAIO

Tabela 3.1 ilustruje mapiranje ključnih elemenata SAIO ontologije na SUMO i DOLCE.

Tabela 3.1 Mapiranje SAIO na SUMO i DOLCE

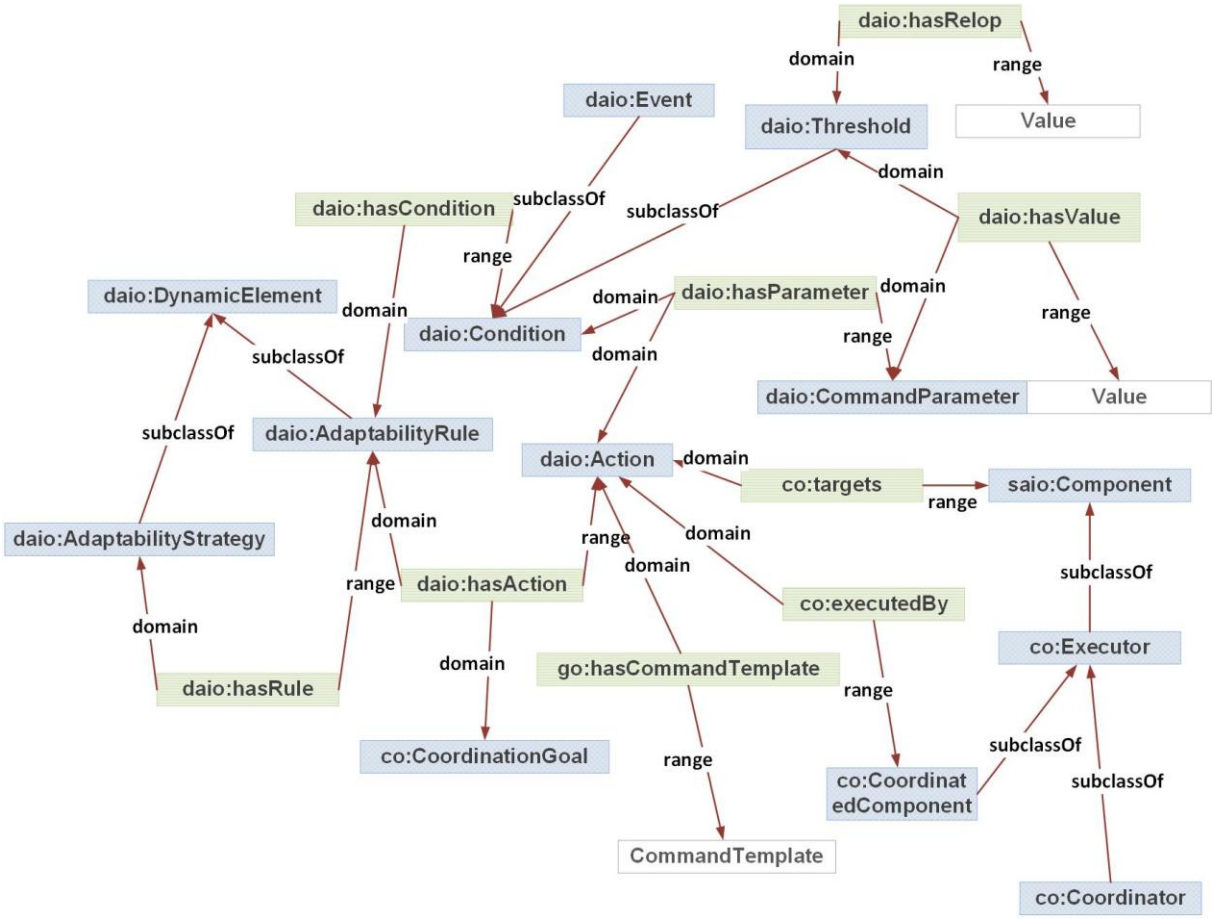
SAIO koncept	SUMO	DOLCE
System	*subClassOf* PhySicalObject	*subClassOf* EndurantEntity
StaticElement		
Characteristic	*subClassOf* Attribute	-
OutputName		
OutputDirectory		
CodeTemplate	*subClassOf* AbstractObject	-
hasCharacteristic	*subPropertyOf* Relation	-
runOn		
hasValue		
hasComponent		
hasOutputDirectory		
hasOutputName		

Sa druge strane, ontološki radni okvir podrazumeva i *Dynamic Aspects Intis Ontlogy* (DAIO), koji obuhvata aspekte ponašanja sistema, koji se ispoljavaju u toku izvršenja, kao što je prikazano na **Slici 3.4**. U ovom slučaju, koncept najvišeg nivoa predstavlja *Adaptability*

Strategy. Ona se, dalje sastoji od skupa pravila adaptacije – *Adaptability Rule*, pri čemu svako pravilo ima tri dela: 1) kontekst, odnosno skup uslova pod kojima se aktivira pravilo – *Condition* 2) akcija kao odgovor na detektovani kontekst – *Action* 3) ciljanu komponentu - *Target*, što predstavlja *Component* iz SAIO ontologije na koju ta akcija utiče. Prema tome, svako od pravila adaptacije se može predstaviti kao skup ($condition_i$, $action_j$, $target_k$), što se može protumačiti kao akcija $action_i$ koja se izvršava nad komponentom $target_j$ u slučaju kada je ispunjen uslov $condition_k$.

Dalje, svaki *Condition* može biti neka promena vrednosti u odnosu na graničnu (*Threshold*) ili pojava određenog događaja (*Event*). Dalje, Threshold ima graničnu vrednost i znak relacije ($<$, $>$, $=$, $\leq$, $\geq$), što se tokom izvršenja u sistemu koristi za proveru da li je razmatrani uslov ispunjen. Recimo, kao primer događaja u sistemu za nadzor – da je došlo do požara ili broj obrađenih frejmova u sekundi od strane nadzorne kamere manji od 10. Sa druge strane, akcije predstavljaju komande, koje se izvršavaju kao odgvor na prethodno detektovani kontekst (skaliranje sistema dodatnim instancama za primer sa frejmovima da bi se poboljšale performanse). Dalje za svrhu generisanja koda, svaka akcija sadrži i šablon komande (go:*CommandTemplate*), koji se parametrizuje relevantnim vrednostima – *CommandParameter*, koji se primenjuju na sličan način kao i kod statičkih elemenata.

Što se tiče odnosa sa SUMO ontologijom, *AdaptabilityStrategy* je podklasa (*subClassOf*) procesa; *AdaptabilityRule*, *Action* i *Command* predstavljaju podklasu (*subClassOf*) apstraktnih objekata, dok *Condition* predstavlja podklasu događaja (takođe *subClassOf*). Iz perspektive DOLCE ontologije, svi ovi koncepti su podklase (*subClassOf*) perdurantnih entiteta, zato što njihovo figurisanje zavisi od vremena. Konačno, veze između pojmova, kao što su *hasRule*, *hasAction*, *hasCommandTemplate*, *executedBy* i *targets* se izvode iz relacija (odnos *subPropertyOf*) u SUMO ontologiji, dok su *Threshold* i *CommandParameter* atributi (odnos *subClassOf*).



Slika 3.4 Dynamic Aspects Intis Ontology - DAIO

Veza između ove dve ontologije je ostvarena relacijama između sistema iz SAIO i strategije prilagođavanja, koja dolazi iz DAIO:

$$\text{saio: System} : \textit{hasAdaptability}\ \text{daio: AdaptabilityStrategy} \tag{3.4}$$
$$\text{daio: Action} : \textit{targets}\ \text{saio: Component} \tag{3.5}$$

Konačno, *Coordinator* ontologija (CO) služi da uskladi izvršenje akcija iz DAIO od strane različitih, pojediničnih elemenata, da bi se ostvario neki zajednički cilj u okviru sistema. Sa tom svrhom, uvodimo poseban tip komponente u ovoj ontologiji, koji predstavlja izvršivu instancu i naziva se *Executor*. Dalje, imamo dva tipa komponenti koje mogu učestvovati u izvršenju: 1) *Coordintor* – pokriva slučaj centralizovane koordinacije, gde jedan entitet upravlja ostalim elementima sa zadatim ciljem 2) *CoordinatedComponent* – predstavlja, u logičkom smislu komponentu koja je zadužena za akciju prilikom koordinacije većeg broje elemenata, međutim, samo izvršenje može biti realizovano od strane *Coordinator-a*. Izvršne instance uvodimo kao komponent, zato što i one mogu imati parametre, a pogotovu su od važnosti oni zahvaljujući kojima možemo pristupiti samom uređaju koji izvršava odgovarajuće akcije, poput IP adrese u slučaju SSH pristupa. Na primer, ovo može biti neki server koji ima ulogu master-

a u klasteru (centralizovana koordinacija) ili, u slučaju višerobotskih okruženja, jedan od robota. U ovom kontekstu, *CoordinationGoal* uvodimo kao cilj koji se realizuje usklađivanjem izvršenje akcija više Executor-a.

Ako gledamo iz perspektive SUMO ontologije, *CoordinationGoal* je proces, dok su *Executor*, *Coordinator* i *CoordinatedComponent* fizički objekti (odnos *subClassOf*), a istovremeno i endurantni entiteti u DOLCE ontologiji (takođe odnos *subClassOf*).

Tabela 3.2 ilustruje mapiranje ključnih elemenata DAIO i CO ontologijae na SUMO i DOLCE.

Tabela 3.2 Mapiranje DAIO i CO na SUMO i DOLCE

DAIO/CO koncept	SUMO	DOLCE
Coordintor	*subClassOf* PhySicalObject	*subClassOf* EndurantEntity
CoordinatedComponent		
Executor		
AdaptabilityStrategy	*subClassOf* Process	*subClassOf* PerdurantEntity
AdaptabilityRule	*subClassOf* AbstractObject	
Action		
Command		
CodeTemplate		
Condition	*subClassOf* Event	
hasRule	*subPropertyOf* Relation	-
hasAction		
hasCommandTemplate		
executedBy		
targets		

3.4 Mehanizam kreiranja domenskih ontologija uz pomoć analogija i kompetentnih pitanja

Sa ciljem olakšavanja postupka kreiranja domenskih ontologija, pored vizuelne notacije, koristi se i skup intuitivnih pitanja namenjenim ekspertima. Na osnovu datih odgovara, po principu sličnosti u odnosu na predložene gornje ontologije koje se tiču statičkih i dinamičkih aspekata sistema, izvode se konkretne domenske ontologije za rešavanje specifičnog problema. Cilj ovih pitanja je da se uspostavi veza između specifičnih domenskih koncepata i elemenata ontologija višeg nivoa. Na ovaj način, ekspertima je olakšano kreiranje ontologija, jer je samo neophodno da odgovore na pitanja, pri čemu nije neophodno poznavanje detalja o višim ontologijama niti je potrebno poznavanje metodologija za inžinjerstvo ontologija. Dalje,

pojmovi koji figurišu u višim ontologijama i na osnovu kojih se vrši analogija su opisani svojim nazivom, a svako pitanje sadrži i opis ukoliko neko koristi alat po prvi put. Ovakav pristup ima za krajni cilj da smanji vreme i trud koji je potrebno uložiti za kreiranje domenskih ontologija od strane eksperata koji nemaju predznanje o samoj problematici i mehanizmima koji se tiču semantičkih tehnologija. Osim toga, kreiranje domenskih ontologija u skladu sa unapred definisanim ontologijama višeg nivoa u ovom slučaju obezbeđuje i kompatibilinost sa algoritmima za generisanje koda, koji su projektovani tako da raspolažu sa konceptima na višem nivou iz kojih izvodimo domenski-specifične elemente. U **Tabeli 3.3** je dat algoritam za generisanje domenskih ontologija na osnovnu analogija i kompetentnih pitanja, zasnovan na osnovama datim u radovima [71] i [82]. U okviru ovog mehanizma se definišu i pravila validacije, čiji je cilj da se proveri da li u novonastaloj ontologiji važe polazni aksiomi koji potiču iz bazne, osnovne ontologije. Što se implementacije tiče, ova validaciona pravila su realizovana kao SPARQL i primenjuju se nad izvedenim konceptima u novonastaloj ontologoiji. Pretpostavka je da mapiranjem na osnvou analogije polazni aksiomi koji važe u osnovnoj, takođe važe i u novodobijenoj domenski-specifičnoj ontologiji. Ovakav postupak se ponavlja za elemente sistema koji obuhvataju sva tri relevantna aspekta: statički, dinamički i koordinacija.

Tabela 3.3 Pseudokod algoritma generisanja domenske ontologije na osnovu ontologija višeg nivoa predloženog radnog okvira

Ulaz: *BO* (base ontology): bazna/polazna ontologija
Izlaz: *TD* (to be done): ciljana, domenski-specifična ontologija
Koraci:

1. Uspostavljanje ekvivalenata u *TD* terminologiji
   Za svaku klasu *C* iz *BO*
   1.1 Skupom pravila generisati pitanje: *Šta je ekvivalentno konceptu C u vašem domenu?*
   Primer: Šta odgovora statičkoj komponenti ? Objašnjenje: Statička kompoennta je topološki element sistema i opisuje fizičku konfiguraciju sistema
   - Odgovor: Server

   1.2 [Ako je odgovor koncept *Ni*] Uspostaviti subclass relaciju sa pojmom *C* iz *BO* (*Ni subclassOf C*)
   1.3 Da li ima još koncepata u *TD* ekvivalentnih *C*?
   [Ako je odgovor DA] Vratiti se n 1.2
   Kraj petlje

2. Uspostaviti sve relacije između navedenih koncepata u *TD* koje nisu *part_of* tipa
3. Uspostavljanje atributa koncepata iz *TD*

   Za svaku klasu *C* iz *TD*
   3.1. Generisati pitanje: *Koja je karakteristika koncepta C?*
   Primer: Koja je karakteristika servera? Odgovor: Memorija

3.2. Da li element C ima još karakteristika?
[Ako je odgovor DA] Vratiti se na 3.1

Kraj petlje

4. Kreiranje instanci ontologije

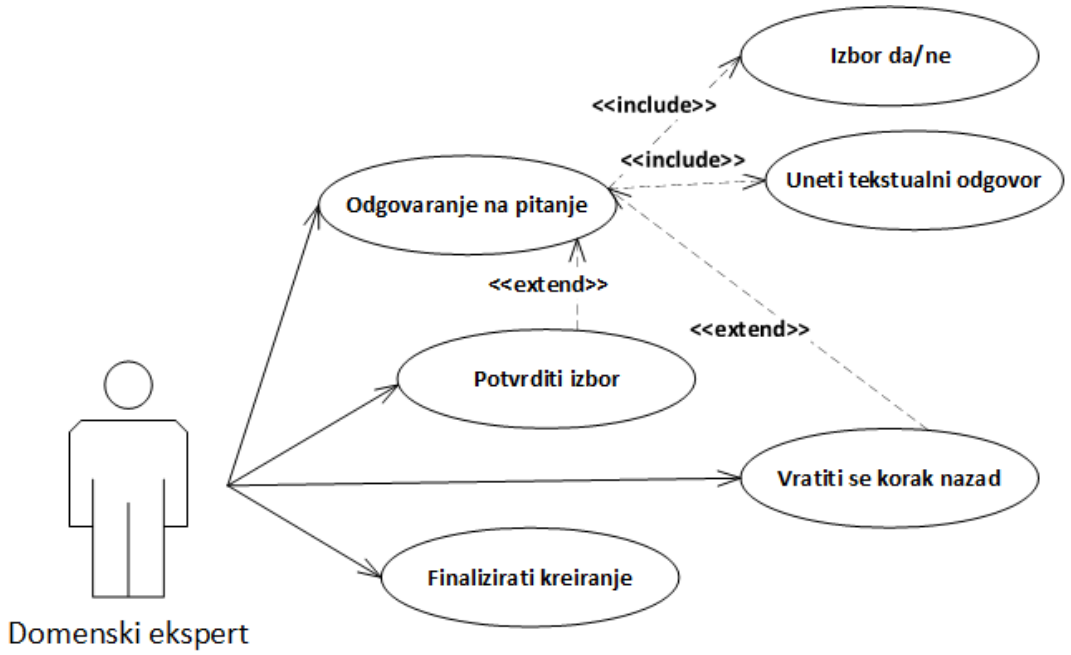
Za svaku klasu *C* iz *TD*
4.1. Postaviti pitanje: *Navesti konkretan primer instance klase C*
Primer: *Navesti konkretan server – RaspberryPi single-board računar*
4.2. Za svaki atribut *a* iz skupa atributa koncepta *C*
Navesti konkretne vrednosti za karakteristiku
Primer: *Raspberry Pi ima 4GB memorije*
Kraj petlje

Kraj petlje

5. Validacija novonastale ontologije
5.1. Ako skup validacionih pravila nije prazan
5.2. Za svako pravilo, proveriti da li važi u novonastaloj ontologiji
5.3. [Ako važi] Završiti kreiranje ontologije
5.4. [Ako ne važi] Vratiti na početak procesa

Kraj

Osim toga, na **Slici 3.5** je dat dijagram slučajeva korišćenja alata za kreiranje domenski-specifičnih ontologija od strane domenskih eksperata, oslanjajući se na analogije i kompetentna pitanja.



Slika 3.5 Slučajevi korišćenja alata za kreiranje ontologija uz pomoć analogija i kompetentnih pitanja

3.5 Automatsko generisanje koda

Algoritam za generisanje koda topologije sistema na osnovu semantičke reprezentacije statičkih aspekata sistema je prikazan u **Tabeli 3.4**. Što se SPARQL upita tiče, korišćena je notacija koja nakon navedenog identifikatora semantičke baze znanja (*sb* u ovom slučaju) u zagradi daje opis navigacije kroz semantičku bazu znanja i uslove filtriranja (po potrebi). Osim toga, kao prefiks je stavljen znak „?“ za onu vrednost koja se pribavlja.

Tabela 3.4 Pseudokod algoritma generisanja topologije sistema na osnovu SAIO ontologije

Ulaz: sb: semantička baza znanja, id_s: identifikator sistema
Izlaz: out_static: kod topologije
Koraci:
1. Pribaviti System element datog identifikatora
   system = sb(?system.hasName.id_s);
2. Naći sve komponente za *system*
   components = sb(?system.hasComponent.Component);
3. Za svaku komponentu *c* iz *components*
4. Pribaviti šablon koda *code_template* za *c*
   code_template = sb(c.hasCodeTemplate.?CodeTemplate);
5. Pribaviti sva karakteristike *characteristics* komponente *c*
   characteristics = sb(c.hasCharacteristic.?Characteristic);
6. Za svaku karakteristiku *p* iz *characteristics*
7. Pribaviti vrednost karakteristike
8. **param_value = sb(p.hasValue.?Value);**
9. staviti *param_value* umesto <*p*> u *code_template*;
10. Kraj petlje
11. Nadovezati generisani kod komponente na izlaz
12. **out_static = out_static.append(code_template);**
13. Kraj petlje
14. Kraj

U prvom koraku se pribavljaju komponente sistema (po svojoj prirodi statički elementi) na osnovu datog identifikatora semantičkog grafa. Nakon toga, za svaku od statičkih komponenti se pribavlja odgovarajući šablon koda, koji se zatim parametrizuje. Parametrizacija se obavlja tako što se jednim upitom pribave sve karakteristike date statičke komponente, a nakon toga za svaku od karakteristika pribavljaju vrednosti koje se umeću u šablon koda. Rezultati generisanja za svaku od pojedinačnih komponenti se nadovezuju, tako je konačni rezultat procesa generisanja sumarni kod kojim je opisana topolgija sistema.

Sa druge strane, **Tabela 3.5** prikazuje pseudokod algoritma koji služi za generisanje komandi kojima se obezbeđuje zadato ponašanje sistema, zavisno od trenutnog konteksta, pri

čemu se kao osnova koristi DAIO ontologija. Ova ontologija omogućava semantičku anotaciju podataka o nastalim događajima, ali i zadavanje ciljeva relevantih za domen, definisanih od strane eksperata. Osim toga, obuhvaćeni su i elementi koordinacije koji potiču iz CO onologije, tako što se za svaku od komandi koja odgovara akciji adaptacije pridružuje informacija o tome koji uređaj treba da je izvrši.

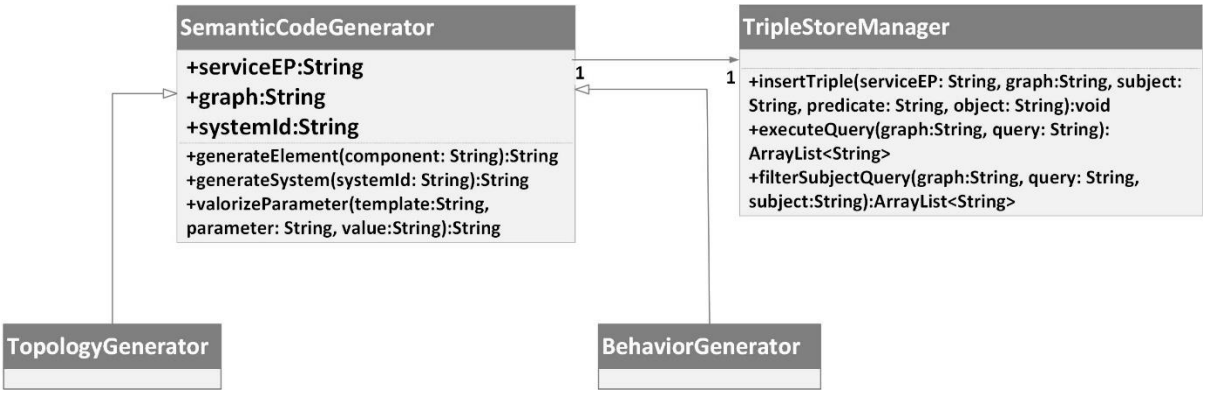
Tabela 3.5 Pseudokod algoritma generisanja komandi za adaptivno i koordinativno ponašanje sistema na osnovu DAIO i CO ontologija

Ulaz: sb: semantička baza znanja, id_s: identifikator sistema
Izlaz: out_dynamic: lista komandi
Koraci:
1. Pribaviti AdaptabilityStrategy element za dati system
   strategy = sb(?system.hasName.id_s.hasAdaptabilityStrategy.?AdaptabilityStrategy);
2. Pribaviti sva pravila prilagođavanja iz strategije
   rules = sb(?strategy.hasRule.?AdaptabilityRule);
3. Za svako pravilo *r* iz *rules*
4. Pribaviti upit uslova *c* koji mora da bude ispunjen da bi se komanda izvršila
   c_query = sb(r.hasCondition.Condition.hasQuery.?ConditionQuery);
5. Izvršiti pribavljeni upit nad semantičkom bazom znanja
   c_result = sb(c_query);
6. Ako je (c_result == True)
7. Pribaviti sve akcije koje se izvršavaju kada je uslov ispunjen
   actions = sb(r.hasAction.?Action);
8. Za svaku akciju *a* iz skupa *actions*
9. Pribaviti šablon komande za datu akciju
10. **command_template = sb(a.hasCommandTemplate.?CommandTemplate)**
11. Pribaviti sve parametre šablona komande
12. **parameters = sb(a.hasParameter.?CommandParameter);**
13. Za svaki parametar *p* iz *parameters*
14. Pribaviti vrednost parametra
15. **param_value = sb(p.hasValue.?Value);**
16. staviti *param_value* umesto <*p*> u *command_template*;
17. Kraj petlje
18. **executor = sb(a.executedBy.?Executor)**
19. Dodati *executor*-a u *command_template*;
20. Nadovezati generisanu komandu na izlaz
21. **out_dynamic = out_dynamic.append(command_template);**
22. Kraj provere uslova
23. Kraj petlje
24. Kraj

U prvom koraku generisanja koda za ispoljavanje dinamičkih aspekata sistema, pribavlja se strategija adaptacije sistema i njeni elementi, koji predstavljaju pravila prilagođavanja. Dalje,

za svako od pravila adaptacije se realizuje mehanizam zaključivanja na sledeći način: pribavi se uslov koji odgovara pravilu; pribavi se SPARQL upit koji odgovara uslovu; izvrašava se upit koji odgovara uslovu. Ukoliko je uslov ispunjen (rezultat *true*), zaključak je da pravilo adaptacije treba primeniti. U tom slučaju se za pravilo pribavlja skup akcija koji treba primeniti. Dalje, svakoj od akcija odgovara komanda, a ona ima svoj šablon. Za komandu se pribavlja skup parametara, a zatim i njihove vrednosti, pa se na osnovu njih popunjava šablon. Konačno, da bi se aspekti koordinacije uzeli u obzir, za akciju se pribavlja informacija o izvršiocu, što se kao informacija dodaje komandi ili direktno koristi prilikom SSH pristupa odgovarajućem uređaju koji će izvršiti komandu.

Što se implementacije mehanizama za automatsko generisanje koda tiče, realizovana je u Java programskom jeziku, a odgovarajući klasni dijagram koji ilustruje arhitekturu dat je na **Slici 3.6**.



Slika 3.6 Arhitektura Java implementacije semantičkog generatora koda

Kao što se može videti, apstraktna klasa za automatsko generisanje koda je imenovana *SemanticCodeGenerator*. Ona poseduje tri atributa: 1) *serviceEP* – pristupna tačka servisa za semantičku bazu tripleta u formi URL-a 2) *graph* – URI grafa u okviru koga se skladište tripleti (videti sekciju 2.3), 3) *system_id* – identifikator inteligentnog sistema za koji se generiše kod. Dalje, ova klasa ima dve apstraktne metode: 1) *generateElement(String component): String* – generiše kod pojedinačne komponente i vraća ga kao string 2) *generateSystem(String system_id): String* – za sistem sa datim identifikatorom generiše odgovarajući kod topologije ili dinamičkog ponašanja, kao skup nadovezanih rezultata dobijenih za pojedinačne komponente. Osim toga, ova klasa ima i jednu pomoćnu metodu, koja nije apstraktna i koristi se u procesu generisanja koda – *valorizeParameter(String template, String parameter, String value): String*. Ova metoda za dati šablon koda *template*, ubacuje vrednost *value* na mesto parametra pod nazivom *parameter* u šablonu i vraća kao rezultat izmenjeni šablon koda (tipa

string). Na osnovu ove klase, zavisno od svrhe generisanja koda, imamo dve različite implementacije konkretnih klasa: a) *TopologyGenerator* – generator statičkih aspekata sistema na osnovu SAIO ontologije, realizacija algoritma iz **Tabele 3.4** b) *BehaviorGenerator* – konstruiše kod koji je relevantan za dinamičke aspekte sistema na osnovu DAIO, implementacija pseudo-koda algoritma iz **Tabele 3.5**. Sa druge strane, kao pomoćna klasa prilikom generisanja koda se koristi *TripleStoreManager* klasa, koja pruža uslužne funkcionalnosti za rad sa semantičkom bazom znanja – upis tripleta i izvršavanje upita na dva načina: 1) *insertTriplet(String serviceEP, String graph, String subject, String predicate, String object): void* – ubacivanje tripleta definisanog kao *(subjekat, predikat, objekat)* u semantički graf ciljanog triple store-a 2) *executeQuery(String graph, String query): ArrayList<String>* - izvršavanje datog SPARQL upita nad semantičkim grafom i 3) *filterSubjectQuery(String graph, String query, String subject): ArrayList<String>* - slično kao 2), pri čemu je subjekat fiksiran datim stringom.

Celokupni pregled metoda prethodno navedenih klasa zajedno sa njihovim opisom je dat u **Tabeli 3.6**.

Tabela 3.6 Pregled metoda klase za rad sa semantičkom bazom znanja TripleStoreManager

Klasa	Metoda	Opis
TripleStore Manager	`insertTriplet(String serviceEP, String graph, String subject, String predicate, String object): void`	Ubacuje triplet oblika *(subject, predicate, object)* u semantičku bazu znanja datu pristupnom tačkom *serviceEP* unutar grafa čiji je naziv dat promneljivom *graph*.
	`executeQuery(String graph, String query): ArrayList<String>`	Izvršava SPARQL upit dat stringom *query* i to nad semantičkim grafom *graph*. Kao rezultat vraća listu stringova koja odgovara reuzultatima upita.
	`filterSubjectQuery(String graph, String query, String subject): ArrayList<String>`	Slično kao *executeQuery*, pri čemu je subjekat u upitu fiksiran datim stringom subject, tako što se subjekat koji je zadat u okviru upita kao ?s menja unutar upita datim stringom.
Semantic CodeGenerator	`generateElement(String component): String`	Generiše kod pojedinačne komponente, a rezultat generisanja vraća kao string
	`generateSystem(String system_id): String`	Generisanje koda u za sistem datog identifikatora, u vidu stringa nastalog nadovezivanjem rezultata dobijenih za pojedinačne komponente
	`valorizeParameter(String template, String parameter, String value): String`	Za šablon koda template dat kao string ubacuje vrednost value na mesto parametra pod nazivom parameter u šablonu. Kao rezultat izmenjeni šablon koda sa umetnutom vrednošću parametra takođe u vidu stringa.

TopologyGenerator	`generateElement: String (override)`	Generiše kod za pojedinačan element topologije sistema
	`generateSystem: String (override)`	Generiše kod za celu topologiju sistema, uzimajući u obzir skup svih elemenata
BehaviorGenerator	`generateElement: String (override)`	Generiše kod za ispoljavanje jednog elemnta ponašanja sistema
	`generateSystem: String (override)`	Generiše kod za ispoljavanje celokupnog ponašanja sistema, uzimajući u obzir skup svih elemenata koji opisuju ponašanje

Konačno, na **Slici 3.7** je dat isečak koda metode za generisanje koda u slučaju statičkih aspekata sistema, sa ciljem ilustracije realizacije opisanih mehanizama.

```
String generateSystem(String serviceEP, String graph, String system_id){
      String components_query="PREFIX penenadpi: <"+graph+"/>\r\n" +
                  "           SELECT DISTINCT ?o\r\n" +
                  "                 WHERE {\r\n" +
                  "                       GRAPH <"+graph+"> {\r\n" +
                  "                  penenadpi:?s penenadpi:hasComponent ?o.  \r\n" +
                  "          }  \r\n" +
                  "              \r\n" +
                  "}\r\n" +
                  "\r\n" +
                  "";
    System.out.println(components_query);
    ArrayList<String> components=TripleStoreManager.filterSubjectQuery(graph,components_query, system_id);

    String out_static = "";

    //Za svaku staticku komponentu sistema
    for (int i=0;i<components.size();i++)
    {
      String component = components.get(i);
      //out_static = out_static + generateElement(serviceEP, graph, component) + "\n" ;
      generateElement(serviceEP, graph, component);
    }

    return out_static;
}
```

Slika 3.7 Implementacija metode za generisanje koda topologije sistema u Javi na osnovu SAIO

Prvo se izvršava upit kojim se pribavljaju sve topološke komponente sistema, pri čemu se rezultati filtriraju samo za one triplete u kojima je na mestu subjekta sistem dat identifikatorom. Rezultati upita se čuvaju kao niz naziva komponenti. Zatim, za svaku od komponeni iz niza se poziva metoda za generisanje koda pojedinačnog elementa na osnovu šablona, a dobijeni rezultati nadovezuju na izlaznu promenljivu *out_static*, koja predstavlja kod za definisanje topologije sistema.

Dalje, na **Slici 3.8a** je dat prvi deo metode za generisanje koda pojedinačnog elementa u slučaju statičkih aspekata. Ovaj isečak obuhvata korake pribavljanja skupa karakteristika za komponentu, zatim putanju izalaznog direktorijuma rezultata, pribavljanje naziva izlaznog fajla i odgovarajućeg šablone koda uz pomoć SPARQL upita.

```
void generateElement(String serviceEP, String graph, String component){

      String characteristics_query="PREFIX penenadpi: <"+graph+"/>\r\n" +
                  "           SELECT DISTINCT ?o\r\n" +
                  "                 WHERE {\r\n" +
                  "                       GRAPH <"+graph+"> {\r\n" +
                  "                        penenadpi:?s penenadpi:hasCharacteristic ?o.  \r\n" +
                  "          }  \r\n" +
                  "              \r\n" +
                  "}\r\n" +
                  "\r\n" +
                  "";

     ArrayList<String> characteristics=TripleStoreManager.filterSubjectQuery(graph,characteristics_query, component);

     String outputdir_query="PREFIX penenadpi: <"+graph+"/>\r\n" +
                  "           SELECT DISTINCT ?o\r\n" +
                  "                 WHERE {\r\n" +
                  "                       GRAPH <"+graph+"> {\r\n" +
                  "                        penenadpi:?s penenadpi:hasOutputDirectory ?o.  \r\n" +
                  "          }  \r\n" +
                  "              \r\n" +
                  "}\r\n" +
                  "\r\n" +
                  "";

     String output_dir=TripleStoreManager.filterSubjectQuery(graph,outputdir_query, component).get(0);

     String template_query="PREFIX penenadpi: <"+graph+"/>\r\n" +
                  "           SELECT DISTINCT ?o\r\n" +
                  "                 WHERE {\r\n" +
                  "                       GRAPH <"+graph+"> {\r\n" +
                  "                        penenadpi:?s penenadpi:hasCodeTemplate ?o.  \r\n" +
                  "          }  \r\n" +
                  "              \r\n" +
                  "}\r\n" +
                  "\r\n" +
                  "";

     ArrayList<String> templates=TripleStoreManager.filterSubjectQuery(graph,template_query, component);

     //Mozemo imate samo jedan sablon koda za komponentu
     for(int k=0; k<templates.size(); k++) {

     String code_template_name = templates.get(k);

     String code_template = fileToString(code_template_name, "node-red-contrib-name");

     String outputname_query="PREFIX penenadpi: <"+graph+"/>\r\n" +
                  "           SELECT DISTINCT ?o\r\n" +
                  "                 WHERE {\r\n" +
                  "                       GRAPH <"+graph+"> {\r\n" +
                  "                        penenadpi:?s penenadpi:hasOutputName ?o.  \r\n" +
                  "          }  \r\n" +
                  "              \r\n" +
                  "}\r\n" +
                  "\r\n" +
                  "";
     ArrayList<String> outputnames=TripleStoreManager.filterSubjectQuery(graph,outputname_query, code_template_name);

     String outputname = null;
     if(outputnames!=null)
       outputname = outputnames.get(0);
     else
       outputname = null;

     ...
```

Slika 3.8a Implementacija metode za generisanje koda pojedinačnog topološkog elementa u Javi na osnovu SAIO

Dalje, isečak koda sa **Slike 3.8b** prikazuje petlju u kojoj se za svaku od komponenti uz pomoć SPARQL upita pribavlja odgovarajuća vrednost, a potom i parametrizuje šablon statičkog elementa. Iterira se kroz niz karakteristika elementa. Za svaku od karaketristika se pribavlja njena vrednost, a nakon toga vrši zamena unutar šablona uz pomoć funkcije

valorizeParameter. U svakom koraku se trenutni šablon ažurira, tako da je izlaz algoritma šablon koda sa svim vrednostima karakteristika. Konačno, rezultat dobijen parametrizacijom se upisuje u fajl zadatog naziva unutar željenog direktorijumu na osnovu parametara iz prvog dela algoritma **(Slika 3.8a).** Na sličan način je implementirana i metoda za generisanje komandi za ispoljavanje dinamičkih aspekata sistema i njegovog ponašanja.

```
        ...
        //Za svaku karakteristiku komponente
        for (int i=0;i<characteristics.size();i++)
        {
         String characteristic = characteristics.get(i);

              String value_query="PREFIX penenadpi: <"+graph+"/>\r\n" +
                         "          SELECT DISTINCT ?o\r\n" +
                         "               WHERE {\r\n" +
                         "                    GRAPH <"+graph+"> {\r\n" +
                         "                   penenadpi:?s penenadpi:hasValue ?o.  \r\n" +
                         "          }  \r\n" +
                         "               \r\n" +
                         "}\r\n" +
                         "\r\n" +
                         "";

              ArrayList<String> values=TripleStoreManager.filterSubjectQuery(graph,value_query, characteristic);

              String result = valorizeParameter(code_template, characteristic, values.get(0));
              code_template = result;
        }

          try {
              String[] ext = code_template_name.split("\\.");
              System.out.println(code_template_name);
              makeNewDir(output_dir);

                   stringToFile(component+"."+ext[1], output_dir, code_template);
              } catch (IOException e) {
                   e.printStackTrace();
                   System.out.println(e.getMessage());
              }

        }
    }
```

Slika 3.8b Implementacija metode za generisanje koda pojedinačnog topološkog elementa u Javi na osnovu SAIO - nastavak

U **Tabeli 3.7**, dat je pregled upita koji se izvršavaju kao deo mehanizma automatskog generisanja koda.

Tabela 3.7 Pregled ključnih SPARQL upita izvršenih od strane algoritma za generisanje koda

Upit	Opis

PREFIX intisont:
<http://www.example.com/intisont/>
PREFIX primer:
<http://www.example.com/primer>
 	
SELECT DISTINCT ?o
WHERE {
GRAPH <http://www.example.com/primer>
{
primer:Domen1 intisont:hasComponent
?o.
}
}
	Pribavlja sve komponente za dati sistem domena.
	(u ovom primeru se sistem zove *Domen1*)

PREFIX intisont:
<http://www.example.com/intisont/>
PREFIX primer:
<http://www.example.com/primer>
 	
SELECT DISTINCT ?o
WHERE {
GRAPH <http://www.example.com/primer>
{
primer:element1
intisont:hasCharacteristic ?o.
}
}
	Pribavljanje svih karakteristika date komponente
	(u ovom primeru za komponentu koja se zove *element1*)

PREFIX intisont:
<http://www.example.com/intisont/>
PREFIX primer:
<http://www.example.com/primer>
 	
SELECT DISTINCT ?o
WHERE {
GRAPH <http://www.example.com/primer>
{
primer:element1
intisont:hasCodeTemplate ?o.
}
}
	Pribavljanje šabliona koda za datu komponentu
	(u ovom primeru za komponentu koja se zove *element1*)

PREFIX intisont:
<http://www.example.com/intisont/>
PREFIX primer:
<http://www.example.com/primer>
 	
SELECT DISTINCT ?o
WHERE {
GRAPH <http://www.example.com/primer>
{
primer:param1 intisont:hasValue ?o.
}
}
	Pribavljanje vrednost za datu karakteristiku (atribut) neke komponenete
	(u ovom primeru za parametar1 komponente *element1*)

3.6 Vizuelno okruženje za modelovanje

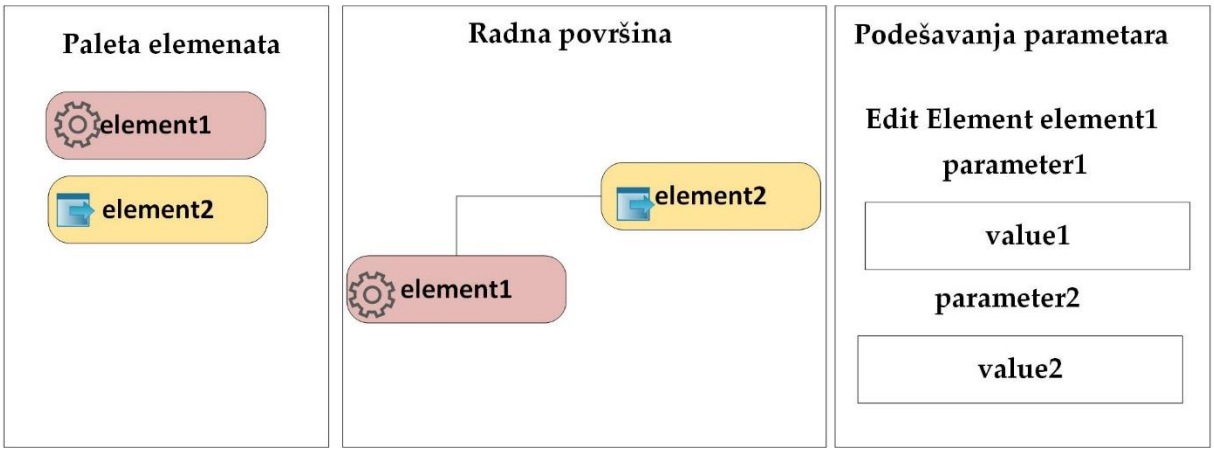
Vizuelno okruženje za modelovanje ima dve uloge:

1) kreiranje domenskih ontologija na osnovu ontologija višeg nivoa (neophodno je poznavanje ovih ontologija, namenjeno naprednijim korisnicima i softverskim/ontološkim inženjerima, i

2) kreiranje instanci domenskih ontologija (namenjeno domenskim ekspertima, ali i svima ostalima).

Srž samog alata zasnovana je na Node-RED radnom okviru, koji je inicijalno razvijen kao rešenje otvorenog koda za IoT prototipe. Međutim, zbog svog intuitivnog korisničkog interfejsa koji se zasniva na jednostavnom prevlačenju komponenti iz palete i njihovom povezivanju, zatim fleksibilnosti i jednostavne integracije sa eksternim servisima, Node-RED je korišćen i kao alat za modelima-vođeno softversko inženjerstvo [9].

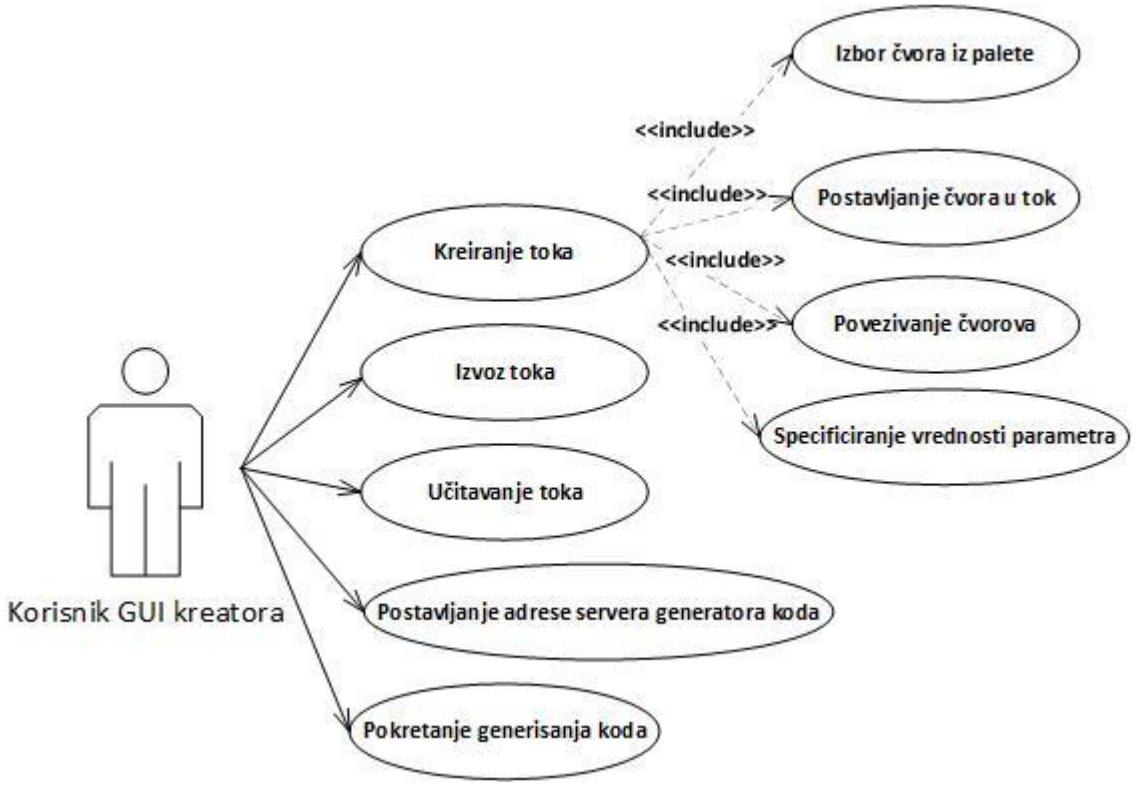
Nakon definisanja ontologije, pramaterizacijom SAIO elemenata odgovarajućim šablonima se generiše okruženje za modelovanje zasnovano na Node-RED alatu. U ovom slučaju se za svaku od komponenti (statičku ili dinamičku) dobija poseban blok u paleti, koji je moguće prevući u radnu površinu. Osim toga, svaki od blokova ima i skup karakteristika (atributa) čije vrednosti može da zada korisnik, kao što je ilustrovano na **Slici 3.9**. Kao što se može videti, korisnički interfejs okruženja za modelvoanje zasnovan na Node-RED ima tri glavna dela: 1) paleta elemenata – sadrži sve dostupe komponente koje koristimo u modelovanju za dati domen od interesa, 2) radna površina – elementi koje smo odabrali iz palete dostupnih tako da ulaze u sastav sistema koji opisujemo, zajedno povezani u tok, i 3) podešavanja parametara – deo interfejsa za postavljanje željenih vrednosti za atribute upotrebljenih elemenata.



Slika 3.9 Struktura grafičkog interfejsa alata za domensko modelovanje

Što se tiče upotrebe grafičkog alata za modelovanje, dijagram slučajeva korišćenja iz perspektive krajnjeg korisnika je dat na **Slici 3.10**. S obzirom da se ovaj alat oslanja na Node-RED, on nasleđuje skup osnovnih funkcionalnosti: interfejs za prevlačenje i povezivanje komponenti (čvorova) iz palete, specificiranje njihovih parametara, povezivanje čvorova u tokove. Osim toga, od osnovnih mogućnosti Node-RED okruženja korisnici imaju mogućnost i da izvezu tokove kreirane na ovaj način u JSON formatu, a kasnije ponovo učitaju iz fajlova. Dodatno, kao proširenje osnovnog skupa funkcionalnosti jeste mehanizam koji klikom na Node-RED dugme „Deploy“ aktivira okidač koji šalje HTTP zahtev ka serveru na kome se

nalazi semantički generator koda. Prema tome, još jedna dodatna mogućnost u odnosu na fabrički Node-RED jeste mogućnost postavljanja URL-a tog servera za generisanje koda.

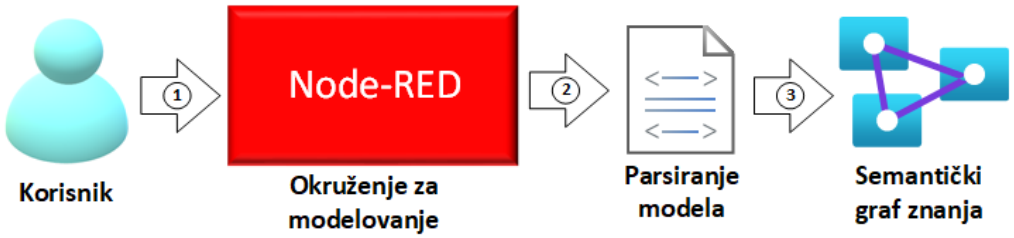


Slika 3.10 Slučajevi korišćenja GUI alata za modelovanje zasnovanog na Node-RED okruženju

Za generisanje okruženja za modelovanje, oslanjamo se na algoritam koji se tiče generisanje statičkih aspekata sistema, kao što je već pokazano u **Tabeli 3.4**. Pre poziva algoritma za generisanje, vrši se kreiranje semantičkog grafa koji predstavlja okruženje za modelovanje, pri čemu se svakoj komponenti ontologije dodeljuje Node-RED blok sa svim neophodnim šablonima. Za svaku od komponenti imamo tri šablona koda: 1) html fajl – definiše izgled bloka, boju, parametre koje možemo uneti, 2) js fajl – definiše ponašanje bloka, način na koji se prosleđuju ulazni parametri sa ciljem formiranja izlaza komponente, i 3) json – opisuje komponentu kao Node-RED modul sa podacima o nameni, autoru i verziji, koji je moguće instalirati u vidu ekstenzije. Nakon instalacije Node-RED modula, blok postaje dostupan u paleti okruženja i može se dodati na radnu površinu. Prema tome, nakon generisanja koda za svaku od komponenti, vrši se automatska instalacija korišćenjem *npm* instalacionog menadžera za Node.js.

Nakon završetka modelovanja, kao rezultat Node-RED generiše izlazni JSON fajl. U JSON fajlu su sadržane informacije o dodatim gradivnim blokovima - čvorovima (*node*), njihovim karakteristikama i načinu međusobnog povezianja. Ovaj JSON fajl se šalje generatoru

RDF grafa, koji parsira JSON fajl i na osnovu datih blokova i njihovih parametara formira triplete koji se ubacuju u semantičku bazu znanja. Tok aktivnosti i interakcija komponenti nakon modelovanja su prikazani na **Slici 3.11**.



Slika 3.11 Interakcija komponenti u fazi modelovanja: 1-Korisnički unos; 2-JSON fajl; 3-Semantički tripleti.

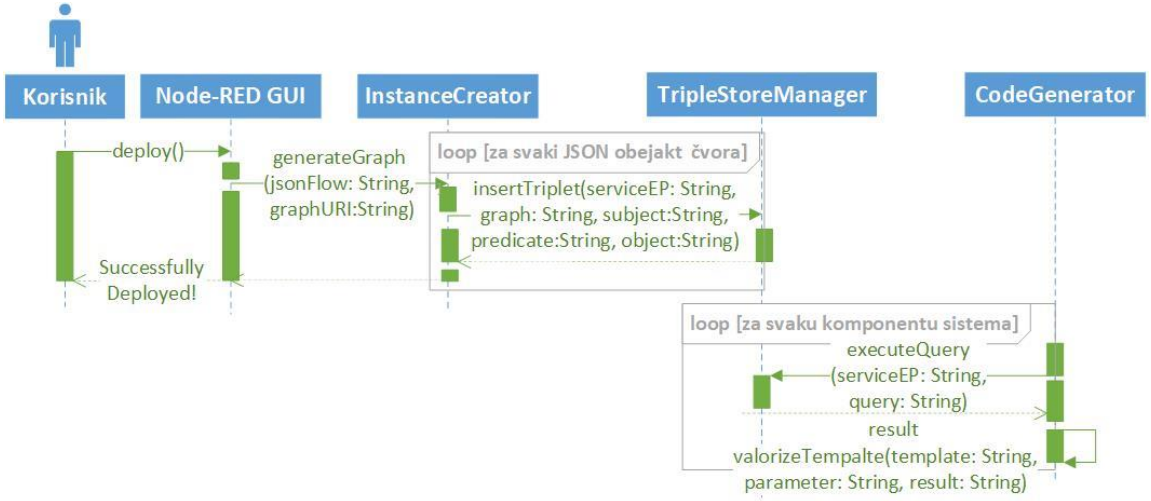
U **Tabeli 3.8** je dat algoritam koji na osnovu kojeg se vrši obrada JSON fajla koji je produkat Node-RED okruženja. U ovom fajlu, nalazi se skup JSON objekata koji predstavljaju odgovarajuće elemente koje je korisnik specificirao. Da bi se izvršila semantička anotacija, za svaki pojedinačan izdvojeni objekat se ubacuje odgovarajući triplet za pripadnost instance klase (predikat *rdf:type*), dok se za svako svojstvo pored toga ubacuje i triplet kojim se postavlja sama vrednost odgovarajućeg atributa. Na ovaj način, korisnik može specificirati semantički graf znanja o strukturi i ponašanju konkretne instance razmatranog sistema, oslanjajuću se na predloženi ontološki radni okvir.

Tabela 3.8 Generisanje semantičkog grafa na osnovu JSON fajla iz Node-RED

Ulaz: json_niz: Node-RED JSON tok povezanih elemenata, g_uri: URI grafa
Izlaz: sg: semantički graf
Koraci:
brojač:=0
sg:=novi semantički graf(g_uri)
Za svaki objekat *o* iz *json_niz*
  sg.insertTriplet(o.id, “rdf:type”, o.class)
  Za svaki ključ atributa *k* objekta *o*
    vrednost:=o[*k*]
    sg.insertTriplet(o.id, hasCharacteristic, *k*+broj)
    sg.insertTriplet(o.id, hasValue, o[*k*])
  Kraj petlje
  brojač++
Kraj.

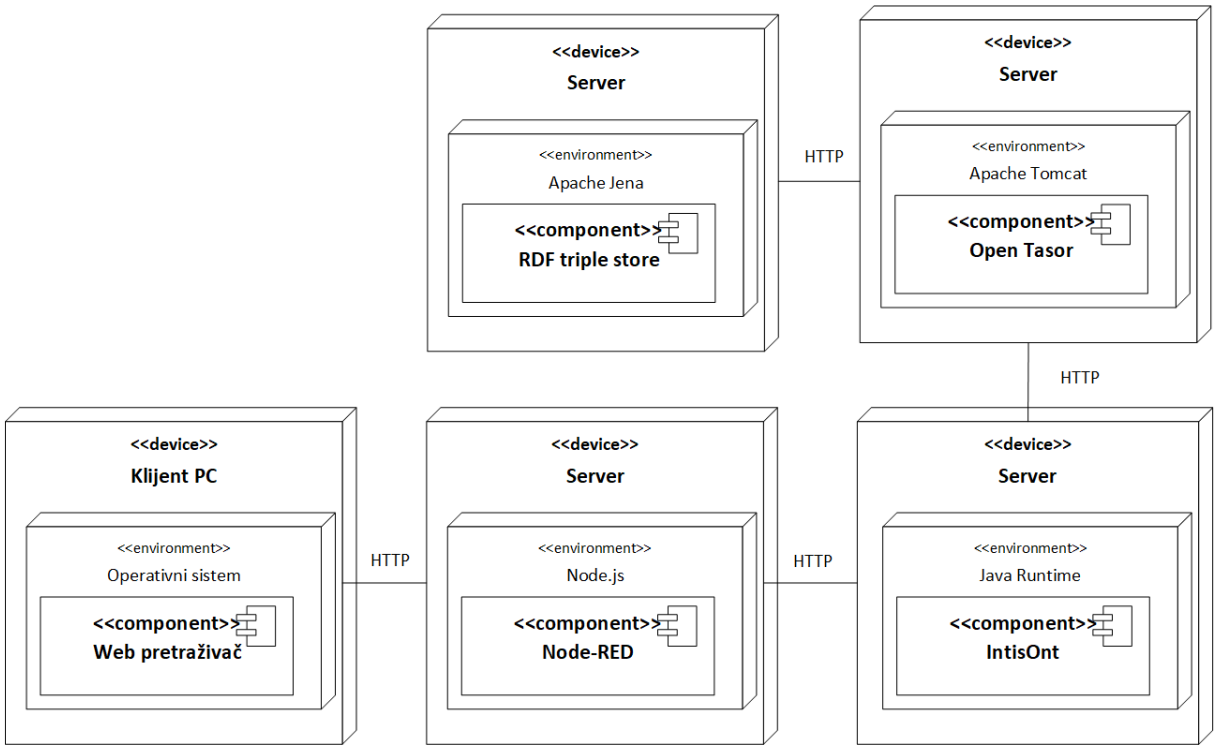
3.7 Interakcija komponenti u izvršenju i raspoređivanje sistema

U nastavku, na **Slici 3.12** je dat sekvencni dijagram koji odgovara interakciji komponenti tokom generisanja semantičkog grafa na osnovu JSON fajla, izvezenog iz Node-RED okruženja. Prethodno korisnik kreira Node-RED tok koji odgovara instanci modela sistema iz određenog domena, specificiranjem elemenata i njihovih parametara. Kao rezultat ovog procesa se generiše JSON, koji IntisOnt parsira i svaki od izdvojenih elemenata ubacuje u semantički triple store. Nakon toga, generator koda izvršava SPARQL upite nad semantičkim grafom koji sadrži triplete nastale na osnovu date instance sistema. Rezultati izvršenih upita se dalje koriste kao parametri odgovarajućih šablona koda.



Slika 3.12 Sekvencni dijagram generisanja semantičkog grafa na osnovu JSON fajla iz Node-RED okruženja

Konačno, **Slika 3.13** prikazuje dijagram raspoređivanja ključnih komponenti sistema. Kao što se može videti, korisnik okruženju pristupa preko web pretraživača. Što serverske strane tiče, uočavamo četiri ključne komponente od značaja: 1) Node-RED – na njemu se unutar Node.js servera izvršava grafički alat za kreiranje instanci sistema 2) IntisOnt – implementacija predstavljenog radnog okvira za semantičko generisaje koda, srž tehničkog dela ove disertacije, zapakovan kao .jar fajl koji se izvršava zahvaljujući Java Runtime-u, a raspoređen na Apache Tomcat serveru s obzirom da pruža REST API 3) Open Tasor – izvršava se kao serverska Java aplikacija unutar Apache Tomcat servera, pruža interfejs visokog nivoa za rad sa semantičkim podacima, jedini u ovom ekosistemu direktno pristupa semantičkim grafovima i ontologijama 4) RDF triple store – služi da čuva definicije ontologija i semantičke baze znanja u skladu sa ontologijama i to u RDF formatu, oslanja se na Apache Jena. Osim toga, kao što se može videti, komponente međusobno komuniciraju HTTP protokolom i to uz pomoć REST API poziva.



Slika 3.13 Dijagram raspoređivanja komponenti IntisOnt ekosistema

4 STUDIJE SLUČAJA

Ovo poglavlje prikazuje studije slučaja iz različitih domena sa ciljem da se ilustruje upotreba predloženog radnog okvira za razvoj i održavanje inteligentnih sistema zasnovanih na ontologijama, obuhvatajući statičke i dinamičke aspekte kao i koordinaciju. Ove studije slučaja predstavljaju reviziju i nadogradnju visokocitiranih i nagrađivanih radova autora disertacije iz ovih oblasti, nastale inspirisane sugestijama kolega iz akademskih i industrijskih krugova, ali i *online* čitalaca.

4.1 Upravljanje računarskim infrastrukturama za mašinsko učenje kod računarstva u magli

Prva prikazana studija slučaja odnosi se na primenu ontologija za opis topologije i ponašanja infrastrukture sistema koji su zasnovani na konceptima računarstva u magli, a bazirana je na ključnim radovima autora [9][108][109]. Cilj ove studije slučaja jeste automatizacija koraka za raspoređivanje servisa u formi Docker kontejnera sa ciljem podrške računarstva u magli upotrebom ontologija, pri čemu su svi neophodni ručni koraci i mehanizmi na kojima se zasniva rad infrastrukture detaljno opisani u master radu autora [110]. U ovoj disertaciji se mehanizmima automatskog generisanja koda na osnovu ontologija teži eliminaciji neophodnih ručnih koraka, sa svrhom ubrzanja procesa raspoređivanja servisa i konfiguracije infrastrukture, čak i tokom izvršenja, zahvaljući mehanizmima adaptacije i koordinacije.

4.1.1 Uvod

Kontinualna integracija i isporuka postaju de facto standard i neizostavni elementi procesa razvoja softvera u današnje vreme. Cilj praktikovanja takozvanih *DevOps* principa jeste težnja da se usaglase aktivnosti razvoja softverskih artefakata sa poslovnim ciljevima koji su zahvaljujući njima omogućeni, tako da organizacija za koju se razvija softverski proizvod može što pre da postigne određene benefite, kao što su pružanje nove usluge krajnjim korisnicima, njeno unapređenje, ušteda ili povećanje efikasnosti [111]. Međutim, operacije vezane za upravljanje infrastrukturom na koju se sistem oslanja postaju sve kompleksnije, između ostalog zbog heterogenosti samih servisa, uređaja i sve većih zahteva u pogledu performansi od strane korisnika i relevatnih scenarija upotrebe.

Pored toga, s obzirom na ogromne količine podataka koje bi bilo potrebno preneti na oblak, ali i na regulative vezane za prikupljanje i prenos podataka o licima (kao što je General Data Protection Regulation - GDPR[9]), primena paradigme računarstva u magli ima sve veću upotrebnu vrednost u praksi. Cilj računarstva u magli je da osim što iskorišćava servere visokih performansi u oblaku, omogući i obradu podataka bliže samim izvorima podataka [112], iskorišćavajući uređaje na ivici mreže (takozvane *edge* uređaje), među kojima pored konvencionalnih servera, desktop i laptop računara može biti i minijaturnih IoT računara, zajedno sa senzorima, poput Rasperry Pi [113].

U ovoj studiji slučaja, cilj je redukovati kompleksnost aktivnosti i isporuku servisa u arhitekturama zasnovnim na računarstvu u magli, oslanjajući se na automatsko generisanja koda koristeći ontologije. Jedan od doprinosa ove studije slučaja obuhvata domensku ontologiju izvedenu na osnovu IntisOnt radnog okvira, a služi za definisanje statičkih i dinamičkih aspekata infrastruktura računarstva u magli. Dalje, automatski generator koda na osnovu reprezentacije u skladu sa ovom ontologijama generiše komande za upravljanje infrastrukturom, poput raspoređivanja servisa, njihove replikacije ili migracije. Konačno, mehanizmima koordinacije se ostvaruje i interoperabilnost uređaja sa ciljem realizacije zajedničkih ciljeva u sistemu (poput povećanja QoS).

U prethodnim radovima autora, metamodelovanje je zajedno sa ontologijama primenjeno na automatizvoano raspoređivanje zadataka za procesiranje podataka u okviru infrastruktura računarstva u magli [9][108][109]. Međutim, ova studija slučaja predstavlja njihovu nadogradnju gde je infrastrukturna platforma za orkestraciju kontejnera koju je činio Docker Swarm[10], sada zamenjena Kubernetesom[11].

4.1.2 Kontejnerizacija upotrebom Docker-a i Kubernetes platforma za orkestraciju

Pojam kontejnerizacije se odnosi na metodu virtuelizacije unutar jednog operativng sistema, koja omogućava izolovano izvršenje više različitih okruženja sa zasebnim aplikacijama, pri čemu ta okruženja koriste isti kernel. U ovom kontekstu, Docker [114] je opšteprihvaćeno rešenje, a inicijalno je bio implementiran u programskom jeziku Go oslanjajući se na Linux kontejnere (LXC) i *cgroups* mehanizme[12] izolacije u sinergiji sa

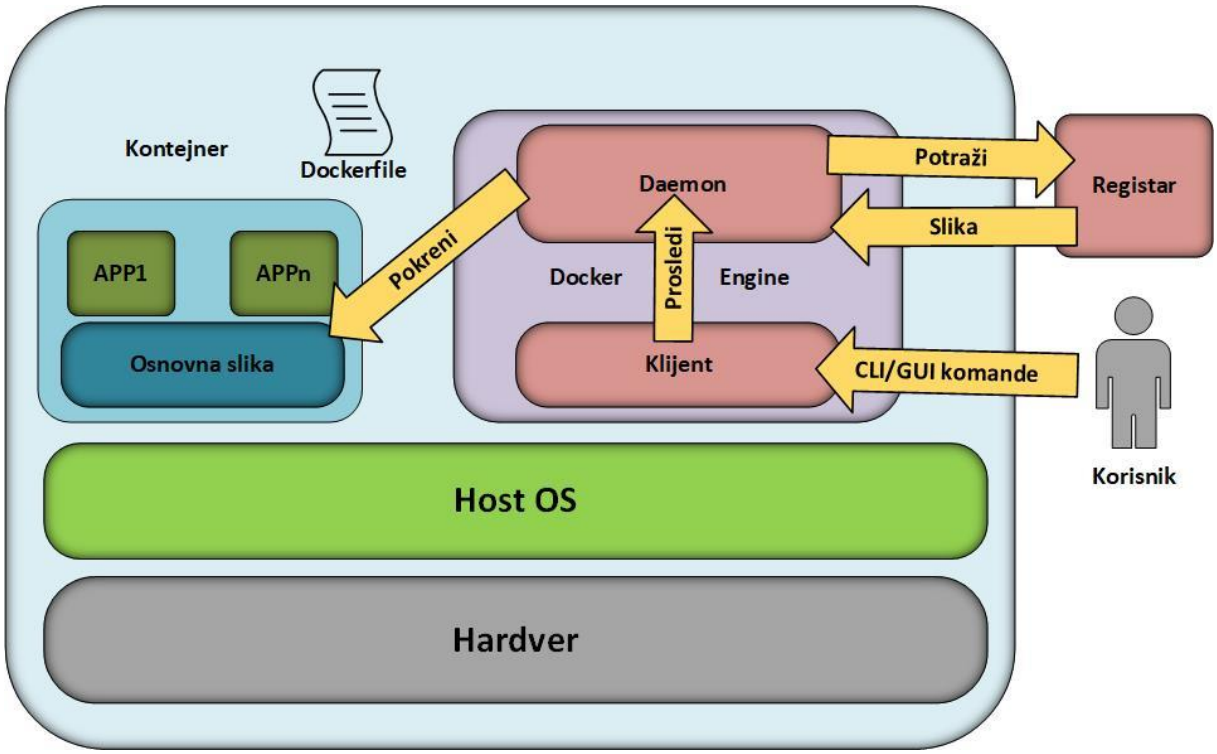
[9] https://gdpr-info.eu/
[10] https://docs.docker.com/engine/swarm
[11] https://kubernetes.io/
[12] https://linuxcontainers.org/lxc/manpages/man1/lxc-cgroup.1.html

funkcionalnostima samog Linux kernela. Na ovaj način, bez obzira na to što se deli isti operativni sistem, ova izolovana okruženja imaju zasebne mrežne konfigracije, fajl sisteme, procese i promenljive okruženja. Prema tome, za razliku od tradicionalnih hipervizorskih virtuelnih mašina, ovaj pristup je manje zahtevan u pogledu resursa, s obzirom da nema potrebe instancirati ceo gostujući operativni sistem za svaki kontejner, što dovodi do povećane efikasnosti upotrebe resursa [108][110]. Srž ove tehnologije je takozvani Docker Engine, koji se zasniva na client-server arhitekturi, iako se obe komponente obično izvršavaju na istoj mašini, dok su ostali bitni koncepti sa objašnjenjima u **Tabeli 4.1**.

Tabela 4.1 Ključni pojmovi za kontejnerizaciju u kontekstu Docker tehnologije

Koncept	Opis
Slika	Izvorni kod koji definiše strukutru nekog kontejnera i način kako se izgrađuje. Obuhvata definisanje bazne slike kao osnove za izgradnju, ali i skup komandi koje se izvršavaju za instaliranje biblioteka, montiranje direktorijuma hosta, bash komande, kopiranje fajlova i slično. Definiše se u okviru tekstualnog fajla bez ekstenzije, pod nazivom Dockerfile, a izgrađuje u direktorijumu fajla komandom `docker build -t ime_slike .`
Kontejner	Izvršna instance Docker slike, koja predstavlja jednu ili više izolovanih aplikacija koje se izvršavaju zajedno, a opciono im se može pristupiti od spolja u vidu servisa preko odgovarajućih portova koji su otvoreni, a kontejner pokreće na osnovu slike komandom: `docker run -d --name ime_kontejnera -p spoljni_port:interni_port repozitorijum/ime_slike`
Registar slika	Repozitorijum u kome se smeštaju Docker slike. Može biti lokalni (sadrži sve preuzete i izgrađene slike) ili javni, kao što je centralni Docker Hub repozitorijum.
Daemon	Serverska komponenta Docker engine-a koja je odgovorna za izgradnju, pokretanje i upravljanje životnim ciklusom kontejnera.
Klijent	CLI ili GUI interfejs koji omogućava zadavanje komandi daemon-u od strane korisnika.
Volumen	Direktorijum host računara koji se montira unutar kontejnera i koristi za trajno čuvanje podataka.

Što se tiče kreiranja novih Docker slika, obično polazimo od takozvanih osnovnih (baznih) slika dostupnih na javnom Docker Hub repozitorijumu. To su obično minimalistički, ogoljeni Linux sistemi koji izvršavaju neku serversku tehnologiju, poput Apache Tomcat-a za Javu, Flask-a za Python programski jezik ili Node.js. Dalje, povrh njih dodajemo odgovarajuća prilagođavanja, kao što je montiranje volumena podataka, instalacija dodatnih biblioteka ili izvršavanje bash komandi. **Slika 4.1** ilustruje odnose između ovih koncepata i tok izgradnje Docker slike, a zatim njeno pokretanje i instanciranje odgovarajućeg kontejnera.



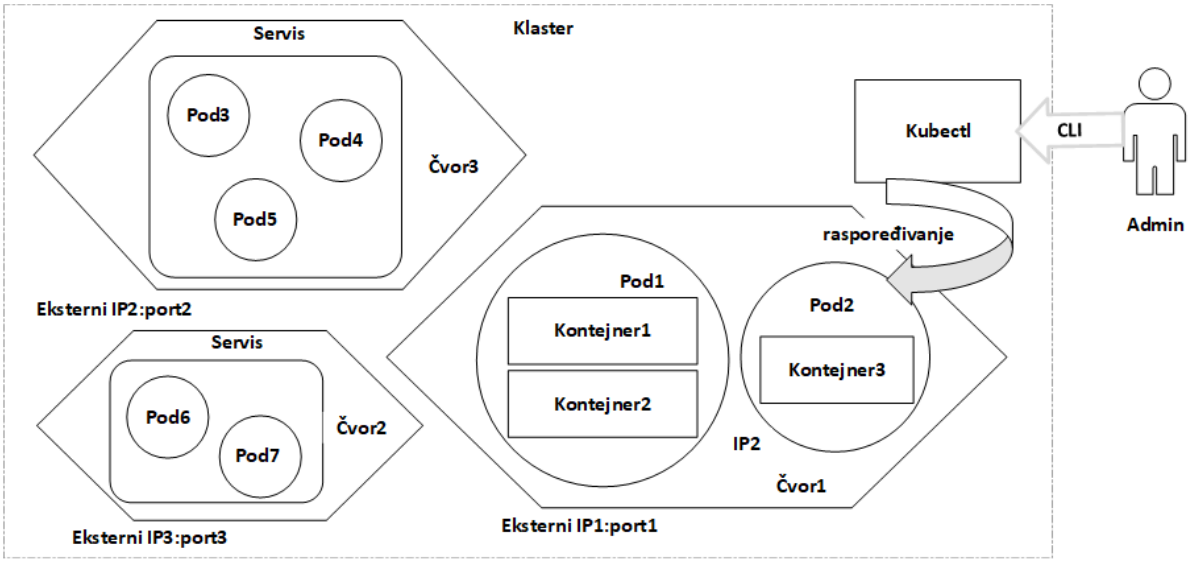
Slika 4.1 Docker koncepti i njihove veze

Sa druge strane, Kubernetes [115] je platforma otvorenog koda, čiji je cilj da omogući olakšano upravljanje životnim ciklusom kontejnera, ali i njihovo raspoređivanju u klasteru računara. Dalje, pruža i širok skup dodatnih mogućnosti, poput skalabilnosti (upravljanje brojem replika), mehanizama otpornosti na greške i deklarativne notacije za konfiguraciju. **Tabela 4.2** prikazuje pregled terminologije relevantne za Kubernetes.

Tabela 4.2 Kubernetes terminologija i ključne komande

Koncept	Opis
Čvor (node)	Računar u klasteru koji izvršava kontejnerizovane aplikacije grupisane u podove
Pod	Najmanja jedinica raspoređivanja, sastoji se od jedne ili više kontejnerizovane aplikacije. Ovi kontejneri dele skladište podataka, specifikaciju mreže i konfiguraciju.
Servis	Logička celina koja se sastoji od skupa podova
Kontroler (control plane)	Komponenta koja upravlja klasterom, odgovorna za globalne odluke i raspoređivanje
kubectl	CLI alat za izvršacanje komandi upravljanja nad klasterom, poput: -raspoređivanja: `kubectl apply deployment.yaml` i `kubectl create deployment ime_raspoređivanja --image=ime_docker_slike` -skaliranja: `kubectl scale --replicas=željeni_broj_replika ime_resursa` -pribavljanje informacija o čvorovima, podovima i servisima: `kubectl get pods, nodes, services`

Bez obzira na to što Kubernetes, između ostalog, pruža i mogućnosti automatskog raspoređivanja zadataka računarima u klasteru, postoje situacije kada želimo da rasporedimo pod na tačno određeni čvor. U ovom slučaju, koristimo labele čvorova. Odgovarajuća komanda za označavanje čvora ima sledeću formu: `kubectl label nodes <node_name> label_name=label_value`. Posle toga, kada želimo da kreiramo pod korišćenjem odgovarajućeg YAML konfiguracionog fajla, potrebno je postaviti svojstvo `nodeSelector`, a unutar njega parameter `label_name` bi trebalo da dobije vrednost `label_value`. Arhitektura zasnovana na Kubernetesu je ilustrovana na **Slici 4.2**. U ovom radu, koristimo kontejnerizovane servise raznih namena kao takozvane zadatke, a da bi im se pristupalo od spolja, koristimo HTTP protokol.



Slika 4.2 Kubernetes arhitektura

4.1.3 Primena IntisOnt semantičkog radnog okvira za računarstvo u magli

Što se statičkih aspekata tiče, za njihovu definiciju polazimo od SAIO ontologije, a izvodimo dve ključne komponente – *Device* (uređaj) i *Task* (zadatak). Uređaji su ujedno i entiteti koji imaju mogućnost izvršenja zadataka, pa se izvode iz *Executor*-a (videti sekciju 3.1 – formulu 3.4 i **Sliku 3.4**), a dalje dele na: 1) *Servers* (serveri u užem smislu) i 2) *Network devices* (specijalizovani mrežni uređaji koji mogu izvršavati virtuelne mrežne funkcije).

Ilustracija primene SAIO onotlogije u okviru domena rauanrstva u magli, data je na **Slici 4.3**. Serveri se odnose na infrastrukturu provajdera koja obezbeđuje usluge, a obuhvataju širok skup heterogenih uređaja, od tradicionalnih serverskih mašina do manjih single-board uređaja, kao što je Raspberry Pi. U pogledu procesorskih arhitektura, to su najčešće tradicionalni x86

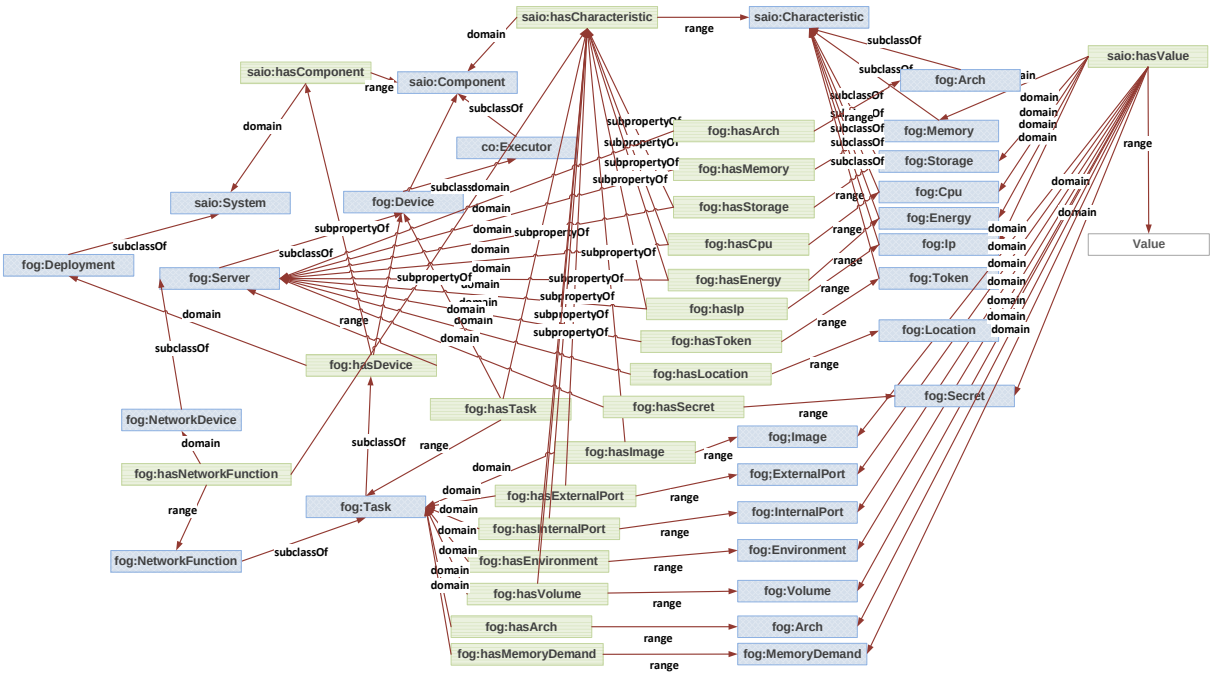
(konvencionalni serveri, laptopovi i kućni računari) ili ARM (single-board računari i pametni telefoni) uređaji. Zajedno, svi oni imaju za cilj da podrže različite scenarije upotrebe od strane krajnjih korisnika – od društvenih mreža, pametne televizije i zabave do podrške funkcionalnosti pametnih domova. Za svaki od uređaja, potrebno je postaviti IP adresu, informacije o lokaciji (*edge* ili *cloud*) i arhitekturu. U scenarijima računarstva u magli, lokacija izvršenja igra vrlo bitnu ulogu, s obzirom da se na osnovu toga može omogućiti da polise privatnosti, bezbednosti i prenosa podataka budu ispoštovane [110]. Osim toga, kao dodatne parametre moguće je pružiti informacije o kapacitetu memorije, diska i procesorskoj moći (prikazano u desnom delu **Slike 4.3**), što se dalje može iskoristiti za optimizaciju performansi – sa ciljem redukcije vremena izvršenja ili smanjenja kašnjenja u mreži. Dalje, zavisno od primenjivane tehnologije za upravljanje infrastrukuturom u pozadini (čist Docker, Docker Swarm ili Kubernetes), dodatni parametri mogu biti postavljeni kao što su korisničko ime i šifra za SSH pristup, token za pridruživanje klasteru. Konačno, i aspekti potrošnje električne energije od strane servera se mogu uzeti obzir sa ciljem postizanja veće energetske efikasnosti, ali i uštede.

Sa druge strane, mrežni uređaji su odgovorni za raspoređivanje mrežnih i komunikacionih protokola, izvršenjem takozvanih mrežnih funkcija, kao što su virtuelni ruteri, mrežni prekidači/komutatori/svičevi (engl. *switch*), firewall uređaji, load balancer, analizatori anomalija u saobraćaju, kontrola pristupa i slično (leva polovina **Slike 4.3**) . Između ostalog, među mrežnim uređajima su i SDN kontroleri koji služe za primenjivanje pravila za oblikovanje saobraćajem u softverski definisanim mrežama pod njihovom nadležnošću.

Dalje, *Task* predstavlja apstrakciju izvršivih komponenti – zadatka, zahvaljujući kojima se realizuju usluge koje provajderi nude krajnjim korisnicima, sa ciljem zadovoljavanja njihovih potreba. U prikazanoj studiji slučaja, svakom od zadataka odgovara po jedan Docker kontejner koji je moguće dodeliti nekom od servera koji su dostupni od strane provajdera infrastrukture za izvršenje. Docker kontejnere u ovom slučaju identifikujemo kombinacijom koja se sastoji od imena odgovarajućeg Docker Hub repozitorijuma i konkretnog naziva odgovarajuće slike na osnovu koje se instancira kontejner. Dalje, specifični tip zadatka je *Virtual Network Function* (VNF), koji predstavlja virtuelnu mrežnu funkciju u softverski-definisanoj arhitekturi mreže. S obzirom da Task-u odgovaraju Docker kontejneri, potrebno i uzeti u obzir i sledeće parametre koje razmatramo kao relevantne za njihovo raspoređivanje i upotrebu (donji deo **Slike 4.3**): 1) *external port* – spoljašnji port kojem se može pristupiti servisu 2) *internal port* – unutrašnji port, koji se koristi interno od strane engine-a Docker host-a, ali mu se ne može pristupiti od spolja na ovaj način 3) *volume directory* – direktorijum na

serveru koji je montiran na Docker kontejner sa ciljem trajnog perzistiranja podataka, što je relevantno za *Task*-ove koji po svojoj prirodi imaju ponašanje trajnog skladišta podataka *4) architecture* - predstavlja kompatibilnu arhitekturu za koju postoji odgovarajući Docker image – *x86*, *ARM* ili *any* (u slučaju da postoje obe verzije image-a) *5) environment* - predstavlja atribut *Task*-a kojim se specificira okruženje u kome bi trebalo da se izvrši, a može imati vrednost *cloud* (u oblaku), *edge* (na ivici mreže) i *any* (bilo gde) 6) *memory demand* – zahtev zadatka po pitanju memorije koju zahteva odgovarajući Docker kontejner. Na ovaj način, omogućeno je da se ispoštuju ograničenja kretanja podataka, što može biti od ključnog značaja za organizaciju. Dalje, specijalizovani tip zadataka jesu mrežne funkcije – *Network function*, a one se mogu izvršavati na *Network devices* (može se videti u donjem levom delu **Slike 4.3**).

Osim toga, semantičkim upitima nad instancama se može izvršiti verifikacija modela sistema, sa ciljem utvrđivanja da li su ograničenja ispoštovana po pitanju slaganja lokacije izvršenja, arhitekture i tipa zadataka (mrežne funkcije ili ostalo). Sa druge strane, ako IP adresa za neki od zadataka nije specifirana u opisu sistema, onda će biti dodeljen nekoj od mašina po internoj polisi koja se primenljuje od strane infrastrukture u pozadini.



Slika 4.3 Statički aspekti domena računarstva u magli izraženi uz pomoć SAIO ontologije

Ilustracija primene DAIO i CO ontologija sa ciljem opisa dinamičkih aspekata i koordinacije u domenu računarstva u magli, data je na **Slici 4.4**. Strategija adaptacije u ovom slučaju sastoji od sekvence pravila adaptacije, pri čemu svako pravilo predstavlja trojku koja se može prikazati kao *(condition, action, task/server).* Na ovaj način označavamo da, ukoliko je ispunjen uslov, onda se aktivira akcija adaptacije koja se odnosi bilo na neki server ili zadatak

koji se izvršava. Primer za situaciju kada se pravilo odnosi na zadatak jeste preopterećenje nekog od servisa, pa onda pravilo adaptacije ima za cilj da izvrši skaliranje, tako što se napravi zadati broj replika odgovarajućeg zadatka koji odgovara tom servisu. Sa druge strane, primer za drugi tip pravila bi bila detekcija napada na neki od servera, tako da se u slučaju da je napadač pristupio serveru, kao odgovor na to izvršava akcija da se on ugasi ili izbacuje iz klastera, sa ciljem da se zaštiti ostatak infrastrukture.

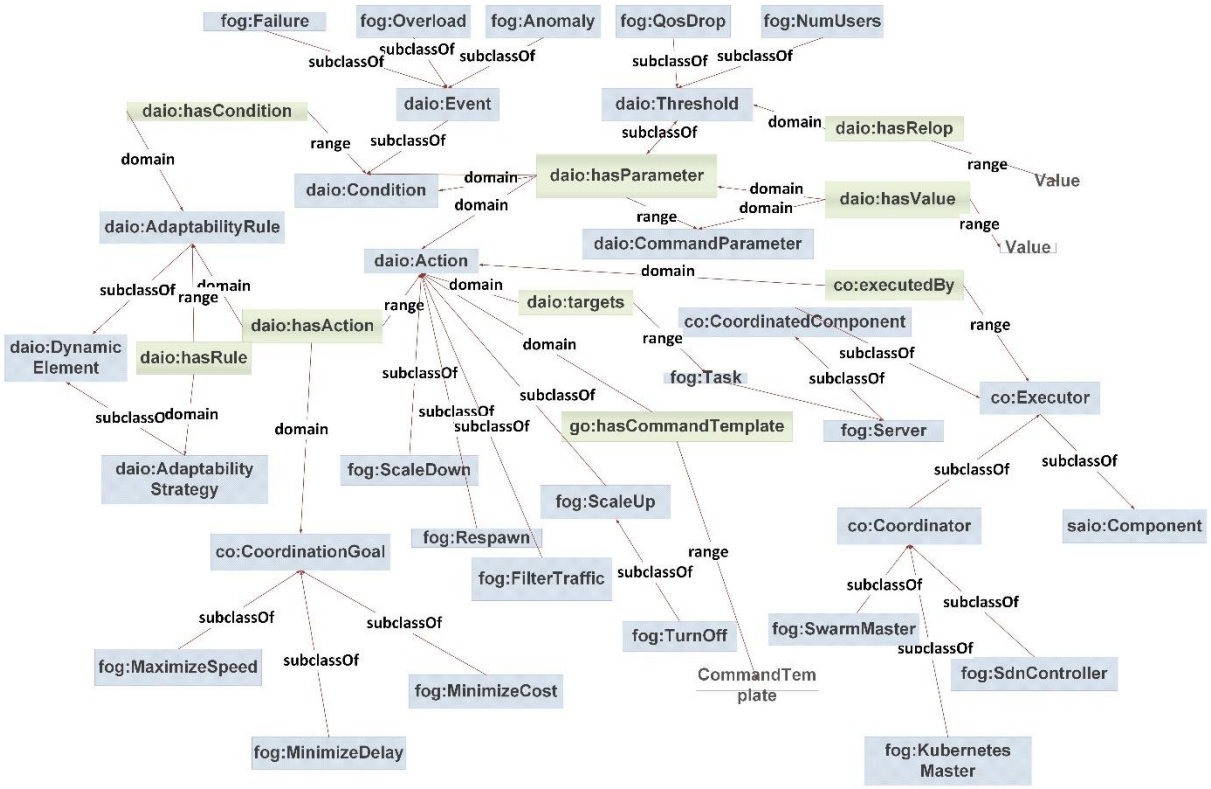
Sa druge strane, uslov aktivacije može biti opisan kao (gornji deo **Slike 4.4**): a) specifičan događaj – kao što je pad servisa (*Failure*), anomalija (*Anomaly*), napad na infrastrukturu (*Attack*) ili preopterećenje (*Overload*) b) relacioni izraz koji se odnosi na neku metriku, poput indikatora za kvalitet servisa – Quality of Service (QoS). U ovoj notaciji, kao jedan od mogućih indikatora možemo koristiti broj aktivnih korisnika servisa, dok događajima smatramo pojave poput anomalija i napada detekotvanih kao rezultat analize serverskog log-a. Primeri odgovarajućih SPARQL upita za proveru ovih uslova su naknadno prikazani u **Tabeli 4.4**.

Dalje, što se akcija tiče, one se sastoji od skupa atomičnih operacija, koje nastaju parametrizacijom šablona koda dinamičkog elementa (centralni deo **Slike 4.4**). U ovom kontekstu, što se prilagođavanja računarskih infrastruktura tiče, kao odogovor se mogu primeniti pravila skaliranja, gašenje/paljenje servisa, oslobađanje resursa, prioritizovanje saobraćaja u SDN mrežama i slično. Kao primer akcija imamo pravila za povećanje broja replika servisa – *ScaleUp*, smanjenje broja replika servisa – *ScaleDown*, gašenje servera – *TurnOff*, ponovno inicijalizovanje servisa na nekom od dostupnih servera – *Respawn.* Osim toga, što se softverski definisanih mreža tiče, možemo imati i prioritizaciju odgovarajućeg tipa mrežnog saobraćaja – *PrioritizeTraffic*, filtriranje mrežnog saobraćaja zabranjivanjem nekog od portova - *FilterTraffic*. Kao metu, ove operacije mogu ciljati bilo server ili neki od zadataka.

Iz perspektive koordinacije, ove operacije su izvršive od strane *Executor* elemenata, kao što su Docker Swarm ili Kubernetes gospodar (*master*) klastera ili kontroler u SDN mrežama (donji desni ugao **Slike 4.4**). Međutim, zavisno od tipa infrastrukture, ali i tipa operacije, izvršilac može biti i sam server koji je meta te akcije, što je slučaj tradicionalne infrastrukutre zasnovane na Docker kontejnerima ukoliko se serveri ne nalaze u okviru klastera.

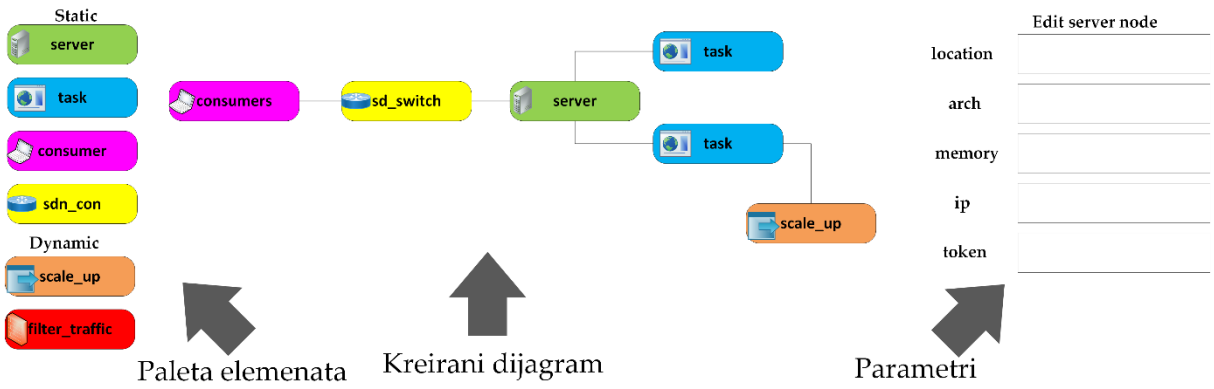
Konačno, imamo nekoliko mogućih primera ciljeva na nivou sistema (donji levi ugao na **Slici 4.4**): 1) *MaximizeSpeed* – rasporediti zadatke tako da se postigne najbrža moguća brzina izvršenja 2) *MinimizeDelay* – rasporediti zadatke tako da se postigne minimalno kašnjenje u prenosu podataka 3) *MinimizeCost* – minimizacija cene, u našem konkretnom sluaju se odnosi na postizanje najveće moguće energetske efikasnosti sistema u datom trenutku. Za realizaciju

koordinacije sistema u ovoj studiji slučaja se koriste modeli linearne optimizacije, čija struktura je data u dodatku.



Slika 4.4 Dinamički aspekti adaptacije i koordinacije domena računarstva u magli izraženi uz pomoć DAIO i CO ontologija

Ilustracija Node-RED okruženja za kreiranje instanci modela sistema računarstva u magli generisano na osnovu datih domenski-specifičnih ontologija je prikazana na **sl. 4.5**. Kao što se može videti, paleta elemenata je razdvojena na statičke i dinamičke aspekte. U datom primeru, u Node-RED tok smo prevukli server koji izvršava dva zadatka, pri čemu je strategija adaptacije za jedan od njih skaliranje ukoliko je broj korisnika veći od graničnog. Korisnike predstavljamo specifičnim elementom, za koji je potrebno specificirati broj aktivnih potrošača. Dalje, sa desne strane je skup elemenata za podešavanje parametara od značaja za selektovani element iz Node-RED toka.



Slika 4.5 Node-RED okruženje za modelovanje statičkih i dinamičkih aspekata sistema računarstva u magli

4.1.4 Korišćeni šabloni generisanja koda

Dalje, razmotrićemo skup korišćenih šablona za automatsko generisanje koda u okviru studije slučaja računarstva u magli. Primetimo da ovaj skup šablona nije konačan niti ograničen već predstavlja praktičnu ilustraciju predloženog koncepta. U **Tabeli 4.3** je dat pregled najbitnijih korišćenih šablona za generisanje koda, sa ciljem realizacije statičkih i dinamičkih karakteristika sistema računarstva u magli, predstavljenih u skladu sa prikazanim semantičkim radnim okvirom ilustrovanim na **Slici 4.3** i **Slici 4.4**. Kao što se može videti, srž mehanizama za generisanje koda u ovoj studiji slučaja jeste u parametrizaciji Docker i Kubernetes komandi. Za svaki od šablona je, osim skupa relevantnih parametara i opisa, takođe dato i ko zapravo izvršava taj kod, sa ciljem ostvarivanja koordinacije u sistemu.

Što se tiče vrste br. 1 - prikazan je skup korišćenih komandi kada u eksperimentu ne postoji orkestrator kontejnera poput Kubernetesa i onda moramo da imamo detalje o konkretnoj mašini, kao što su IP adresa i podaci za SSH pristup serveru kome želimo da dodelimo neki zadatak. U ovom slučaju, prva komanda jeste SSH pristup serveru koju treba da izvrši zadatak - vrsta 1a.Zatim se na osnovu date Docker slike instancira Docker kontejner kojem se može pristupiti od spolja preko porta zadatog broja – vrsta 1b. Jedna od pretpostavki jeste da se u nazivu Docker image-a na kraju nalazi oznaka za arhitekturu – *x86* ili *arm*, da bi se na taj način omogućio pravilan izbor varijante slike. Opciono, moguće je dati i informacije o montiranom volumenu za perzistenciju podataka, što se predstavlja kao mapiranje tipa *direktorijum_hosta:direktorijum_kontejnera*.

Ukoliko se koristi Kubernetes, potrebno je prethodno inicijalizovati klaster od strane gospodara (treća komanda). Nakon toga, za svaki od servera, potrebno je izvršiti pridruživanje prethodno kreiranom klasteru, pri čemu je neophono postaviti vrednost tokena, koja se može videti na samom gospodaru prilikom inicijalizacije klastera. Dalje, za svaki od servera se i

dodaju labele, koje specificiraju odgovarajući tip arhitekture, ali i lokaciju izvršenja, što se kasnije koristi za selekciju servera kome se neki zadatak dodeljuje.

Komanda data u vrsti br. 4 opisuje strukutru Kubernetes deskriptora za kreiranje poda odgovarajućeg zadatka (pojmovi prethodno objašnjeni u **Tabeli 4.2**) . U prvom delu se generiše YAML deskriptor (vrsta 4a), a u drugom on primenjuje uz pomoć *apply* komande (vrsta 4b). Od bitnih parametara, tu su naziv odgovarajućeg Docker image-a za kreiranje kontejnera, spoljašnji port kojem se pristupa kontejneru, ali i lokacija gde se izvršava dati zadatak, što je specificirano *nodeSelector* parametrom. Na taj način, moguće je razgraničiti da li zadatak se izvršava u *cloud* ili *edge* okruženju.

Vrsta br. 5 se odnosi na komandu kojom se realizuje jedna od akcija za adaptaciono pravilo skaliranja nekog zadatka, povećanjem ili smanjenjem broja replika, U ovom slučju, to je *kubectl* scale komanda, a što se tiče parametara, od ključnog su značaja broj željenih replika, ali i naziv ciljanog zadatka.

Dalje, vrsta br. 6 se odnosi na prevenciju saobraćaja sa datog porta da bi se sistem zaštitio u slučaju detekcije napada na osnovu loga. Za svrhu analize logova i detekcije anomalija se koristi prediktivni model opisan u dodatku.

Vrsta br. 7 prikazuje pravilo adaptacije kojim se zaustavlja izvršenje servisa i oslobađaju alocirani resursi brisanjem odgovarajućeg poda uz pomoć *kubectl delete* komande. Konačo, vrsta br. 8 prikazuje pravilo koje predstavlja sekvencu u kojoj se prvo izvršava komanda iz vrste br. 7 za oslobađanje resursa, a zatim obe komande iz vrste br. 4 za ponovno raspoređivanje zadatka na drugom serveru.

Tabela 4.3 Šabloni koda studije slučaja računarstva u magli

Br	Element	Opis	Šablon koda	Izvršava
1	Task	Pokretanje Docker kontejnera.	a)`ssh {Server.hasUser.User}@{Server.hasIp.Ip}`	Server kome je dodeljen zadatak
			b)`docker run -d --name {Task} -v {Task.hasVolume.Volume} -p {Task.hasExternalPort.ExternalPort}: {Task.hasInternalPort.InternalPort} {Task.hasImage.Image}{Task.hasArch.Arch}`	
			Ograničenje: `Server.hasTask.Task`	
2	Server	Pridruživanje čvora Kubernetes klasteru	a)`kubeadm join {Sever.hasToken.Token}`	Server koji želimo pridružiti klasteru
			b)`kubectl label nodes {Server} arch={Server.hasArch.Arch} location={Server.hasLocation.Location}`	

3	Kubenetes Master	Inicijalizacija Kubernetes klastera	`kubeadm token create --print-join-command`	Gospodar Kubernetes klastera

4	Task	Raspoređivanje kontejnera u Kubernetes klasteru koji odgvara nekom od zadataka na osnovu generisanog YAML fajla	a)
			`apiVersion: v1`
			`kind: Pod`
			`metadata:`
			`  name: {Task}`
			`  labels:`
			`  task: {Task}`
			`  arch: {Task.hasArch.Arch}`
			`  env: {Task.hasEnvironment.Environment}`
			`  spec:`
			`  containers:`
			`  - name: {Task}`
			`  image: {Task.hasImage.Image}{Task.hasArch.Arch}`
			`  ports:`
			`  - containerPort: {Task.hasExternalPort.ExternalPort}`
			`  nodeSelector:`
			`  loc: {Task.hasLocation.Location}`
			b)
			`kubectl apply -f {Task}.yaml`
				Gospodar Kubernetes klastera

5	ScaleUp/ ScaleDown	Kreiranje zadatog broja replika servisa koje odgovaraju nekom od zadataka	`kubectl scale --replicas={ScaleUp.hasParameter.ReplicaNumber.hasValue} {ScaleUp.targets.Task}`	Gospodar Kubernetes klastera
6	FilterTraffic	Zabranjivanje saobraćaja na datom portu	`port vlan exclude {FilterTraffic.hasPortNumber.Port}`	SDN kontroler
7	TurnOff	Prekinuti izvršenje zadatka	`kubectl delete pod {Turnoff.targets.Task}`	Gospodar Kuberentes klastera
8	Respawn	Prekinuti izvršenje zadatka, a zatim ga ponovo pokrenuti na drugom serveru	Sekvenca od dva prethodno prikazana šablona: 1) *TurnOff* 2) *Task*, pri čemu se svuda gde je originalnim šablonima bio dat parametar kao `{Task}`, za ovaj slučaj menja sa `{Respawn.targets.Task}`	Gospodar Kubernetes klastera

Dalje, što se tiče ostvarivanja domenskih ciljeva sistema, koriste se modeli linearne optimizacije koji su dati u dodatku na kraju tekućeg poglavlja. Jedan od ovih ciljeva se odnosi na optimalno raspoređivanje zadataka serverima po nekom od kriterijuma – brzina izvršenja ili minimalna potrošnja, dok drugi ima za cilj da minimizuje mrežno kašnjenje oblikovanjem saobraćaja u softverski definisanoj mreži.

4.1.5 SPARQL upiti za verifikaciju sistema i proveru uslova adaptacije

Jedna od prednosti predloženog semantički zasnovanog pristupa jeste mogućnost verifikacije uslova koji moraju biti zadovoljeni u sistemu za sve instance izvršenjem SPARQL upita, s obzirom na formu reprezentacije sistema u vidu semantičkih tripleta koji su usklađeni sa prikazanim ontološkim radnim okvirom. Osim toga, uslovi za pravila adaptacije se takođe mogu svesti na SPARQL upite, koji će takođe biti dati u nastavku.

U ovoj studiji slučaja, s obzirom na prirodu razmatranog domena računarstva u magli, od ključnog značaja za verifikaciju su sledeća četiri aspekta:

1) lokacija izvršenja – svaki od zadatak se mora poklapati po izvršnom okruženju (*cloud*, *edge*) sa lokacijom servera kome je dodeljen na izvršenje;

2) arhitektura – potrebno je da se serveru dodeli zadatak koji poseduje odgovarajuću varijantu Docker slike, kompatibilnu sa arhitekturom procesora samog servera (*x86* ili *ARM*);

3) tip zadataka – virtuelne mrežne funkcije se mogu izvršavati samo na softverski-definisanim mrežnim kontrolerima, i

4) memorijski zahtevi – serveru ne možemo dodeliti na izvršenje zadatak čiji odgovarajći Docker image prevazilazi njegove memorijske kapacitete.

U skladu sa tim, **Tabela 4.4** prikazuje odgovarajuće upite za pokrivanje ovih aspekata verifikacije. upiti su implementirani tako da vraćaju neprazan skup rezultata u slučaju kada je ishod pozitivan. U suprotnom, ako nije vraćen niti jedan element, smatra se da u samoj instnaci sistema postoje elementi koji ne zadovoljavaju date uslove. Kao alterantivni pristupu u implementaciji se mogu koristiti ASK upiti (videti sekciju 2.3), ali je zadržana upotreba SELECT u konačnoj implementaciji radi održavanja uniformnosti interfejsa za rad sa upitima koji vraćaju konkretne vrednosti kao rezultat. U nastavku ove tabele su i upiti koji odgovaraju uslovima za aktivaciju odgovarajućih pravila adaptacije.

Upit br. 1 se odnosi na uslov poklapanja lokacije izvršenja servera i zadatka – ako je zadatak predviđen da se izvršava na edge, onda mora da bude raspoređen server koji nije u uslov se odnosi na uslov skaliranja, koji se aktivira u slučaju kada je predviđeni broj korisnika

servisa koji odgovara nekom zadatku veći od graničnog. Dalje, upit br. 7 se odnosi na proveru da li je detektovana anomalija na nekom serveru. U tom slučaju, aktiviraće se zabrana saobraćaju na portu koji odgovara dodeljenom zadatku tom serveru, sa ciljem sprečavanja napada koji može potencijalno dovesti do otkaza u radu celokupnog sistema. Oba SPARQL upita za proveru uslova kao rezultat vraćaju listu zadataka za koje se aktivira odgovarajuće pravilo adaptacije.

Tabela 4.4 SPARQL upiti za verifikaciju instanci sistema i proveru ispunjenosti uslova pravila adaptacije za studiju slučaja računarstva u magli

Br.	Aspekat	Upit

1	Poklapanje lokacije izvršenja	`PREFIX fog: <http://www.example.com/fog/>`
		`SELECT DISTINCT ?s ?t ?e`
		`WHERE {`
		`GRAPH <http://www.example.com/fog> {`
		`?s fog:hasLocation ?l.`
		`?t fog:hasEnvironment ?e.`
		`FILTER(STR(?l)=STR(?e))`
		`}`
		`}`

2	Poklapanje arhitekture	`PREFIX fog: <http://www.example.com/fog/>`
		`SELECT DISTINCT ?s ?t ?as`
		`WHERE {`
		`GRAPH <http://www.example.com/fog> {`
		`?s fog:hasArch ?as.`
		`?t fog:hasArch ?at.`
		`FILTER(STR(?as)=STR(?at))`
		`}`
		`}`

3	Primer kompletnog poklapanja arhitekture i lokacije izvršenja za konkretan par (server, zadatak)	`PREFIX fog: <http://www.example.com/fog/>`
		`SELECT DISTINCT ?s ?t ?e`
		`WHERE {`
		`GRAPH <http://www.example.com/fog> {`
		`?s fog:hasLocation ?l.`
		`?s fog:hasArch ?as.`
		`?t fog:hasEnvironment ?e.`
		`?t fog:hasArch ?at.`
		`FILTER(STR(?l)=STR(?e) && STR(?as)=STR(?at) &&`
		`regex(STR(?s), "server1") &&`
		`regex(STR(?t), "task1"))`
		`}`
		`}`

4	Uslov da se virtuelne mrežne funkcije izvršavaju na mrežnim uređajima	`PREFIX fog: <http://www.example.com/fog/>`
		`SELECT DISTINCT ?d ?t`
		`WHERE {`
		`GRAPH <http://www.example.com/fog> {`
		`?d rdf:type fog:NetworkDevice.`
		`?t rdf:type fog:NetworkFunction.`
		`?d fog:hasTask ?t.`

		} }

5	Memorijsko ograničenje	PREFIX fog: <http://www.example.com/fog/>
		SELECT DISTINCT ?s ?t
		WHERE {
		GRAPH <http://www.example.com/fog> {
		?s fog:hasMemory ?ms.
		?t fog:hasMemoryDemand ?mt.
		FILTER(?mt < ?ms)
		}
		}

6	Uslov sklairanja	PREFIX fog: <http://www.example.com/fog/>
		SELECT DISTINCT ?t
		WHERE {
		GRAPH <http://www.example.com/fog> {
		?a fog:targets ?t.
		?a fog:hasNumUsers ?nu.
		FILTER(?nu > 100)
		}
		}

7	Uslov prisustva anomalije	PREFIX fog: <http://www.example.com/fog/>
		SELECT DISTINCT ?t
		WHERE {
		GRAPH <http://www.example.com/fog> {
		?a fog:targets ?t.
		?a rdf:type fog:Anomaly.
		}
		}

4.1.6 Dodaci

4.1.6.1 Optimalna dodela Docker kontejenera upotrebom linearne optimizacije

Za implementaciju mehanizma alokacije kontejnera serverima koristi se linearna optimizacija, čije su osnove prethodno prikazane u podsekciji 2.9.1. Model linearne optimizacije koji se koristi u ovoj studiji slučaja biće opisan u nastavku.

Cilj ovog modela linearne optimizacije jeste da rasporedi *m* aplikacionih kontejnera iz zadatog skupa $(k_1, k_2...k_m)$, tako da se izvršavaju na *n* servera $(s1, s2…s_n)$, tako da bude zadvojen unapred definisani cilj, dok su sva ograničenja istovremeno ispunjena. Što se ciljeva optimizacije tiče, to može biti raspoređivanje kontejnera sa svrhom postizanja maksimalne brzine izvršenja ili, sa druge strane, minimalnom cenom. Osim toga, potrebno je uzeti u obzir i ograničenja koja se odnose na različite aspekte: dostupna memorija na svakom od servera, potrebe za memorijom od strane kontejnera dodeljenih serveru, ali i lokacija izvršenja – *cloud* ili *edge*, što je od ključne važnosti kod računarstva u magli.

Svakom paru servera i kontejnera se pridružuje promenljiva odlučivanja koja označava da li je posmatrani kontejner raspoređen da se izvrši na tom serveru ili ne:

dodela[i,j]=1, ako kontejner$_j$ je raspoređen na server$_i$

dodela[i,j]=0, u suprotnom (4.2)

Jedna od mogućih funkcija cilja u ovom slučaju jeste maksimizacija brzine izvršenja, tako što se kontejneri raspoređuju serverima koji imaju najveću procesorsku moć:

$$maximize \sum i \in Serveri, j \in Kontejneri\ brzina[i,j]dodela[i,j] \quad (4.3)$$

Pri tome, brzina predstavlja matricu koja pokazuje koliko je *server$_i$* efektivan što se tiče izvršenja zadatka kome odgovara *kontejner$_j$* .

Dalje, sledeća ograničenja se takođe uzimaju u obzir:

a) Suma memorije koju zauzimaju kontejneri dodeljeni jednom serveru (označeno kao *potražnja*) ne sme biti veća od ukupnog kapaciteta RAM memroije za *server$_i$*, što se može zapisati kao:

$$\sum j \in Kontejneridodela[i,j]potražnja[j] \leq kapacitet[i], gde\ je\ i\ \in\ Serveri \quad (4.4)$$

b) Svaki kontejner se dodeljuje tačno jednom serveru:

$$\sum i \in Serveri\ dodela[i,j] = 1, j\ \in\ Kontejneri \quad (4.5)$$

c) Za svaki od kontejnera, njegovo izvršno okruženje (*cloud* ili *edge*) mora da se slaže sa fizičkom lokacijom servera, što izražavamo kao:

$$\sum j \in Kontejneri|lokacija\_servera[i] - okruženje\_kontejnera[j]| * dodela[i,j] = 0, gde\ je\ i\ \in\ Serveri \quad (4.6)$$

U ovom slučaju imamo da su i *lokacija_servera* i *okruženje_kontejnera* binarni vektori, pri čemu imamo da vrednost "0" označava izvršenje na ivici mreže (*edge*), dok "1" označava da se obavlja u oblaku (*cloud*). Slično, za svaki od kontejnera se njegova verzija slike (x86 ili ARM) mora slagati sa arhitekturom procesora servera kome se on dodeljuje. I ovaj aspekat je karakterističan za računarstvo u magli, s obzirom na heterogenost uređaja koji su deo infrastrukture, posebno ako izvršno okruženje ima podršku za izvršenje na različitim arhitekturama:

$$\sum j \in Kontejneri|arhitektura\_servera[i] - verzja\_kontejnera[j]| * dodela[i,j] = 0, gde\ je\ i\ \in\ Serveri \quad (4.7)$$

Sa druge strane, energetska efikasnost se uočava kao jedan od bitnih aspekata računarstva u magli. Prema tome, druga varijanta funkcije cilja bila bi minimizacija ukupne potrošnje električne energije od strane servera na kojima su raspoređeni kontejneri, dok imamo informaciju o prosečnoj potrošnji energije servera:

$$minimize \sum i \in Serveri, j \in Kontejneri\ potrošnja[i]dodela[i,j] \quad (4.8)$$

Konačno, kompletan AMPL kod prethodno opisanog linearnog modela optimizacije se može pronaći u radu [9].

4.1.6.2 Oblikovanje saobraćaja unutar softverski-definisane mreže

Pretpostavimo da imamo robota opremljenog kamerom i povezanog na mrežni prekidač koji podržava softverski definsane mreže označenog brojem "1". Sa druge strane, odgovarajući server na kome je raspoređen servis računarskog vida, označen je brojem "4". Cilj je generisati skup SDN pravila za oblikovanaje saobraćaja tako da je kašnjenje kroz mrežu minimalno. Problem svodimo na pronalaženje najkraće putanje od čvora "1" do čvora "4" u grafu u kome mrežni prekidači predstavljaju čvorove dok veze između njih predstavljaju potege. Svi mrežni prekidači u ovoj mreži su pod kontrolom jednog istog SDN kontrolera, koji ima mogućnost da primenjuje pravila oblikovanja saobraćaja koja će biti generisana na osnovu izlaza procesa linearne optimizacije.

U nastavku je dat opis odgovarajućeg linearnog programa.

Svakom od potega dodeljujemo promenljivu odlučivanja *x* koja označava da li taj poteg ulazi u putanju ili ne:

x[i,j] = 1, Ako je poteg[i,j] deo najkraće putanje

x[i,j]=0, u suprotnom (4.12)

Funkcija cilja teži da minimizuje kašnjenje putanje, što je jednako sumi kašnjenja pojedinačnih potega koji ulaze u putanju:

$$minimize \sum i,j \in Čvorovi\ kašnjenje[i,j]x[i,j] \quad (4.13)$$

Dalje, sledeće ograničenje mora biti ispoštovano u okviru modela: putanja mora da se prostire od čvora „1" do čvora „4". Ako izuzmemo početni čvor *početak* i zadnji čvor *kraj*, na svakom čvoru *i*, duž putanje, moramo imati jedan poteg koji ulazi u taj čvor, dok, sa druge strane, tačno jedan napušta taj čvor:

$$\sum h,i \in Čvorovi\ x[h,i] - \sum i,j \in Čvorovi\ x[i,j] = bi, gde\ važi:$$

$$Ako\ if\ (i = početni) then\ bi = 1,$$

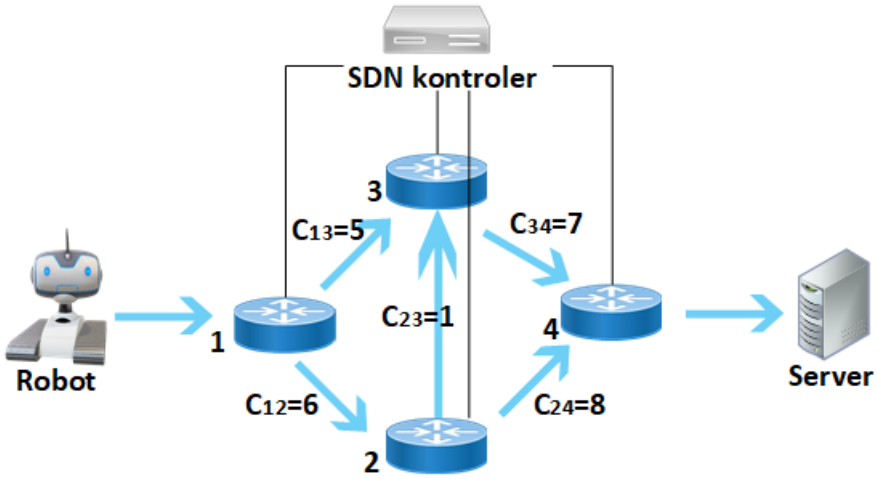
$$Ako\ (i = e) onda\ b_i = -1, u\ suprotnom\ b_i = 0 \quad (4.14)$$

Dalje, mora biti zadovoljeno i:

$$Ako\ nije\ (\ (i,j) \in P\ )\ onda\ kašnjenje[i,j] > 0 \quad (4.15)$$

AMPL kod ovog modela linearne optimizacije se može naći u okviru rada [96]. Ilustracija jedne instance problema može se videti na **Slici 4.6**. U ovom slučaju, najkraća putanja je preko 1-3-4, pri čemu je ukupno kašnjenje 12, što je manje od 14 (oba preostala imaju veće kašnjenje).

Na ovaj način, optimalnim oblikovanjem saobraćaja je, u datom primeru sa slike, smanjeno kašnjenje skoro 15%.



Slika 4.6 Primer dinamičkog oblikovanja saobraćaja u softverski-definisanoj mreži

4.1.6.3 Predikcija anomalija na serveru klasifikacijom logova uz pomoć Weka biblioteke u Javi

Za trening ovog prediktivnog modela, korišćen je javno dostupan skup podataka sa Kaggle[13]-a. On predstavlja Coburg Intrusion Detection Data Sets (CIDDS) skup podataka, koji je namenjen evaluaciji sistemima za detekciju anomalija u mrežnim sistemima. Korišćeni log se sastoji se od 172 838 Wireshark zapisa, a njegova struktura je ilustrovana u **Tabeli 4.5**. Prikazana metodologija se zasniva na radu [116]. Sivom bojom je označena ciljana promenljiva, koja se dobija kao ishod predikcije.

Prediktivni model korišćen u studiji slučaja računarstva u magli je obučen korišćenjem ukupno 5000 zapisa iz loga, a *cross validation* metoda je korišćena za njegovu evaluaciju. S obzirom da je ishod kategoričke prirode (prisutna anomalija/napad ili ne), upoređeno je nekoliko klasifikacionih metoda iz Weka biblioteke za ovu svrhu.

Tabela 4.5 Korišćeni skup podataka za analizu logova

Promenljiva	Opis
Date first seen	Trenutak početka sesije
Duration	Ukupno trajanje sesije
Protocol	Upotrebljavani protokol, poput TCP ili UDP
Src IP Addr	IP adresa pošiljaoca zahteva
Src Pt	Port na kome je poslat zahtev
Dst IP Addr	IP adresa servera koji prima zahtev
Dst Pt	Port servera koji prima zahtev
Packets	Ukupan broj paketa

[13] https://www.kaggle.com/code/kerneler/starter-server-logs-suspicious-575834c0-8

Bytes	Ukupna količina saobraćaja u bajtovima
Flows	Broj tokova
Flags	Mrežni flegovi
Tos	
Class	Ishod: anomalija ili normalan saobraćaj

Konačno, dobijeni rezultati za različite Weka metode klasifikacije su prikazani u **Tabeli 4.6**. Na osnovu rezultata, može se videti da je Instance-Based *kNN* (IBk) metod pokazao najbolje performanse (99.22% za *accuracy* metriku) u pogledu kvaliteta predikcija, zajedno sa prilično kratkim vremenom potrebnom za obuku. Dalje, *J48* metod je malo gori po pitanju kvaliteta predikcija, ali je brži. *DecisionTable* je sporiji od oba, a ima i gore performanse, pri čemu *DecisionStump* ima najslabije performanse predikcije od upoređivanih modela, dok mu je vreme treninga umereno. U proseku, vreme potrebno za predikciju ishoda za jedan unos u logu je reda veličine desetine od sekunde.

Tabela 4.6 Rezultati klasifikacije logova severa sa ciljem detekcije anomalija upotrebom Weka biblioteke u Javi

Algoritam	Vreme obuke [s]	Accuracy [%]
IBk	1.81	99.22
J48	1.35	98.72
DecisionTable	6.12	96.52
DecisionStump	2.14	84.30
Jedna predikcija	Prosečno vreme izvršenja 0.124 s	

4.2 Koordinacija mobilnih robota za nadzor prostorija

4.2.1 Uvod

Četvrta industrijska revolucija je otvorila nove horizonte upotrebe robota u sajber-fizičkim sistemima i interakcije između čoveka i mašine. Sa druge strane, roboti postaju sve više zastupljeni u svakodnevnom životu i to ne isključivo u industrijskom okruženju, već i unutar domova i to sa različitim primenama - od pametnih usisivača do zabave i rehabilitacije. Bez obzira na to što je opšteprihvaćeno da koordinisano ponašanje većeg broja (robotskih) uređaja može efikasnije dovesti do realizacije zadatog cilja [8], ova činjenica nije dovoljno eksploatisana u većini dostupnih aplikacija koje se oslanjaju na robote.

U ovom kontekstu, druga prikazana studija slučaja razmatra primenu predloženog ontološkog radnog okvira za opis topologije i ponašanja multirobotskog sistema, čiji je cilj nadzor prostorija i intervencija u slučaju detektovnih opasnosti, nepravilnosti ili nepoštovanja

propisa u industrijskim i drugim okruženjima. Za ovu svrhu, teži se ostvarivanju koordinisanog ponašanja robota sa ciljem ostvarivanja brže i efikasnije intervencije. Ova studija slučaja zasniva se na elementima iz prethodnih radova autora, pri čemu su neki od njih realizovani u okviru RAWFIE-SCOR H2020 projekta [117]. U radovima objavljenim na osnovu rezultata ovog projekta, razmatrani su različiti aspekti koordinacije robota: 1) predložen je ontološki radni okvir za eksperimentisanje sa robotima [118] 2) implementacija semantičkih mehanizama koordinacije robota na osnovu predloženih ontologija [119] 3) aspekti verifikacije koordinisanog ponašanja robota [120] 4) rezultati eksperimenata semantičke koordinacije robota u realnim testbedovima [8]. U ovoj disertaciji se teži da se prethodni progres u oblasti koordinacije robota generalizuje i uklopi u predloženi ontološki radni okvir, dok se umesto EDL domenski-specifičnog jezika korišćenog od strane drugih autora, za definisanje misija robota koristi sveobuhvatan semantički opis u skladu sa IntisOnt radnim okvirom. Za realizaciju ove studije slučaja se koriste Turtlebot roboti zasnovani na ROS-u, slični onima koji su originalno korišćeni na SCOR projektu.

4.2.2 Robot Operationg System (ROS) i Turtlebot roboti

Robot Operating System (ROS) obuhvata skup softverskih paketa i komponenti otvorenog koda, čiji je cilj omogućavanje pojednostavljenog razvoja aplikacija namenjenih različitim robotskim platformama, pri čemu poznavanje specifičnosti konkretnog robotskog hardvera nije neophodno [121][122]. U suštini, ROS se često kategoriše i kao “meta-operativni sistem”, zbog toga što praktično pruža skup funkcionalnosti tradicionalnog operativnog sistema u domenu robotike, ali se, sa druge strane i dalje oslanja na OS domaćina. U užem smislu, ROS ima za svrhu da pruža funkcionalnosti kojim se olakšava interakcija korisnika i operativnog sistema domaćina, a sa druge strane - robotske opreme, senzora i aktuatora. ROS pruža hardversku apstrakciju zahvaljujući kojoj možemo upravljati komponentama robota, pri čemu nije neophodno poznavati detalje hardverske realizacije svake od njih. Na primer, ako želimo da pomerimo pokretnog robota, zadaćemo ROS komandu, koja zatim poziva odgovaraujuću skriptu implementiranu od strane proizvođača tog modela robota, koja na nižem nivou upravlja hardverom za promenu položaja točkova. Što se izdavanja komandi i nadzora senzorskih podataka u okviru ROS-a tiče, oni se realizuju po objavi/pretplati se (engl. *publish-subscribe*)

principu. ROS u potpunosti podržava klijentske biblioteke za više različitih programskih jezika: Python (*rospy*[14]), C ++ (*roscpp*[15]) i JavaScript (*roslibjs*[16]).

Osnovne komponente ROS ekosistema su čvorovi, gospodar i poruke. Čvor (engl. *node*) je osnovna izvršna jedinice u ROS-u i predstavlja proces koji izvršava određene zadatke. Svaki čvor može biti odgovoran za različite funkcije robotskog sistema, kao što su kontrola motora, obrada senzorskih podataka, ili planiranje putanje kod pokretnih uređaja. Čvorovi mogu komunicirati jedni s drugima slanjem i primanjem poruka. Poruke (engl. *messages*) su strukture podataka koje se koriste za razmenu informacija između čvorova. Poruke su tipizirane i mogu sadržavati različite vrste podataka, kao što su tekstualni sadržaj, numeričke vrednosti, ali i merenja sa senzora i drugi kompleksniji podaci. Sa druge strane, ROS gospodar (engl. *master*) je serverska komponenta koja igra ključnu ulogu u upravljanju komunikacijom između čvorova. Gospodar ima sledeće uloge: 1) praćenje tema i usluga - vodi evidenciju o svim aktivnim temama i uslugama u ROS mreži čvorova 2) registracija novih čvorova - kada se novi čvor pokrene, potreno je da se registruje kod odgovarajućeg gospodara kako bi dalje mogao da komunicira sa ostalim čvorovima 3) upravljanje parametrima – gospodar omogućava centralizovano upravljanje parametrima, što olakšava konfiguraciju i prilagođavanje sistema.

Primarni mehanizam za razmenu podataka između ROS čvorova je slanje i primanje poruka po principu objavi/pretplati se. Ako čvor želi da deli informacije, pomoću objavljivača (engl. *publisher*) šalje podatke na odabranu temu (engl. *topic*). Sa druge strane, čvor koji ima potrebu da prima informacije koristi pretplatu (engl. *subscription*) na istu tu temu, pa iz perspektive razmene poruka ima ulogu pretplatnika (engl. *subscriber*).

U ovom kontekstu, tema je osnovni komunikacioni mehanizam za asinhronu razmenu podataka između čvorova. Teme omogućavaju čvorovima da komuniciraju jedni s drugima slanjem i primanjem poruka određenog tipa, bez potrebe da direktno poznaju jedni druge, što zapravo predstavlja suštinu komunikacije po principu objavi/pretplati se. ROS teme imaju sledeće karakteristike: 1) jedinstveno ime - svaka tema ima jedinstveno ime koje omogućava čvorovima da identifikuju temu preko koje žele da komuniciraju; 2) tipizacija: teme su tipizirane, što znači da sve poruke poslate na određenoj temi moraju biti istog tipa; 3) asinhronost - komunikacija preko tema je asinhrona, što znači da čvorovi mogu slati i primati poruke nezavisno jedni od drugih; 4) tema se koristi za komunikaciju više na više (engl. *many-to-many*) - mnogi objavljivači mogu slati poruke na istu temu, a i veći broj pretplatnika može

[14] http://wiki.ros.org/rospy
[15] http://wiki.ros.org/roscpp
[16] http://wiki.ros.org/roslibjs

da ih prima; 5) objavljivači i pretplatnici mogu se kreirati i uništavati bilo kojim redosledom, nezavisno od teme; i 6) poruka se može objaviti na neku temu čak i ako nema aktivnih pretplatnika.

ROS održava listu unapred definisanih tema koje služe za razmenu poruka, a mogu se pregledati komadnom `rostopic list`. Jedan od primera koji se koristi u ovom radu jeste *Twist* tema, a njena svrha jeste pomeranje robota. U konzolnom interfejsu, ova komanda se zadaje na sledeći način:

$$\text{rostopic pub /tb3\_0/cmd\_vel geometry\_msgs/Twist } -\text{r } 10 \; -\!- \; -\!- \; '[0.2,0.0,0.0]' \; '[0.0,0.0,0.0]' \tag{4.1}$$

Ako želimo da promenimo položaj robota, šaljemo ROS poruku sa sledećim argumentima: 1) *-r broj*: broj izdatih komandi u sekundi; 2) linearne koordinate; 3) ugaone koordinate. Za komandu koja je data u (4.1), robot se pomera unapred za 0.20 koraka duž X ose, dok se izdaju 10 komandi ovakve strukture po sekudni.

U **Tabeli 4.7** je dat pregled ključnih ROS komandi inicijalizacije gospodara i čvorova, koje se kasnije koriste kao šabloni za generisanje koda.

Tabela 4.7 Pregled ključnih ROS komandi

Korak	Komanda	Opis
Podešavanje robota	`ssh username@ip`	Pristup robotu povezanom na mrežu preko SSH protokola
	`export TURTLEBOT3_MODEL= ROBOT_MODEL`	Prilikom pokretanja aplikacija koje rade sa apstraktnim hardverom, pomoću već definisanih tema, neophodno je definisati model konkretnog hardvera sa kojim radimo. Parametar ROBOT_MODEL zapravo predstavlja model robota, što može imati neku od datih vrednosti: TB3_MODEL (za Turtlebot 3), WAFFLE ili WAFFLE_PI,
	roslaunch turtlebot3_bringup turtlebot3_robot.launch	Kada je konfiguracija sistema završena i server pokrenut, na robotu se pokreće fabrički dobijena aplikacija koja vrši apstrakciju hardvera robota i kreira adekvatne topic-e na serveru u zavisnosti od njegovog hardverskog sklopa.

Podešavanje ROS mastera	`sudo nano ~/.bashrc`
	`ROS_MASTER_URL= http://MASTER_IP:ROS_PORT`
	`ROS_HOSTNAME = MASTER_IP`
		Na računaru koji predstavlja ROS gospodara, potrebno je izmeniti konfiguracioni fajl .bashrc, tako što konfigurišu ROS_MASTER_URL i ROS_HOSTNAME, tako da ukazuju na jednu te istu adresu, odnosno ip adresu samog udaljenog računara sa ulogom ROS gospodara.

	`roscore`	Pokretanje ROS serverske komponente na gospodaru
Pomoćne ROS komande za rad sa senzorima	`rostopic list`	Listanje dostupnih ROS topic-a za razmenu poruka
	`rostopic echo/topic_name`	Čitanje vrednosti sa senzora za datu ROS temu

Što se konkretnih modela koji podržavaju ROS tiče, u ovom radu se oslanjamo na TurtleBot [106] robotsku platformu, predstavljenu 2010. godine. Ona obuhvata seriju robota različitih mogućnosti i karakteristika, u formi kopnenih vozila. Svi oni imaju baterisjko napajanje, potpuno su mobilni, malih dimenzija i relativno pristupačne cene u odnosu na opremljenost i mogućnosti koje pružaju. U trenutku rada na ovoj studiji slučaja, bile su aktuelne tri generacije modela TurtleBot proizvoda, pri čemu je u ovom radu konkretno razmatrana Turtlebot3 serija, predstavljena 2017. godine. Robote iz ove serije karakeriše potpuna SLAM (Simultaneous Localization and Mapping) podrška, a obuhvata tri različita modela: Burger, Waffle i Waffle Pi. Za eksperimente su nam konkretno bili na raspolaganju Burger i Waffle Pi modeli, a njima je zajedničko da se zasnivaju na Raspberry Pi 3 single-board računaru, dok oba poseduju merni instrument za prostorno lasersko skeniranje (Light Detection and Ranging – LiDAR) od 360 stepeni. Međutim, Waffle modeli, za razliku od Burger-a imaju i dodatnu kameru, koja je povezana za Raspberry Pi, što otvara novi horizont mogućnosti, pogotovu što se tiče analize scene, detekcije različitih vrsta objekata i primene računarskog vida, uopšte. Pored toga, ovi roboti se oslanjaju i na OpenCR [124] mikrokontroler, koji omogućava upravljanje senzorima i hardverom na niskom nivou za funkcionalnosti koje se tiču pokretanja motora, skretanja i slično. Što se primene robota iz ove familije u praksi tiče, bez obzira na solidan set mogućnosti, navedeni modeli su prvenstveno namenjeni razvoju prototipa, eksperimentisanju i edukaciji u oblasti robotike.

4.2.3 Primena IntisOnt semantičkog radnog okvira za koordinaciju nadzornih robota

U sistemu nadzornih robota (koncept referenciran kao *rco:RobotSystem*, izveden iz *saio:System*), možemo uočiti sledeće relevantne statičke aspekte:

1) *Robot* – predstavlja robotski uređaj, koji može biti pokretan (*MobileRobot*) ili fiksan (*FixedRobot*), a iz perspektive ROS-a odogovara pojedinačnom čvoru;

2) *Master* – server koji ima ulogu ROS gospodara i zadužen je za vođenje evidencije o robotima, dostupnim ROS temama i omogućava pridruživanje novih robota koaliciji robota kojom upravlja;

3) *Mission* – zaduženje koje robot u višerobotskom sistemu ima kao svoj zadatak, i 4) *AddOn* – dodatak, zaseban uređaj koji je povezan sa robotom i pod njegovom je kontrolom, a koristi se sa ciljem da omogući realizuaciju neke misije. Dodatak može biti po prirodi:

a) *Sensor* – senzor koji beleži podatke i merenja o sredini u kojoj se nalazi, ili

b) *Actuator* – dodatak kojim robot može da utiče na stanje sredine u kojoj se nalazi.

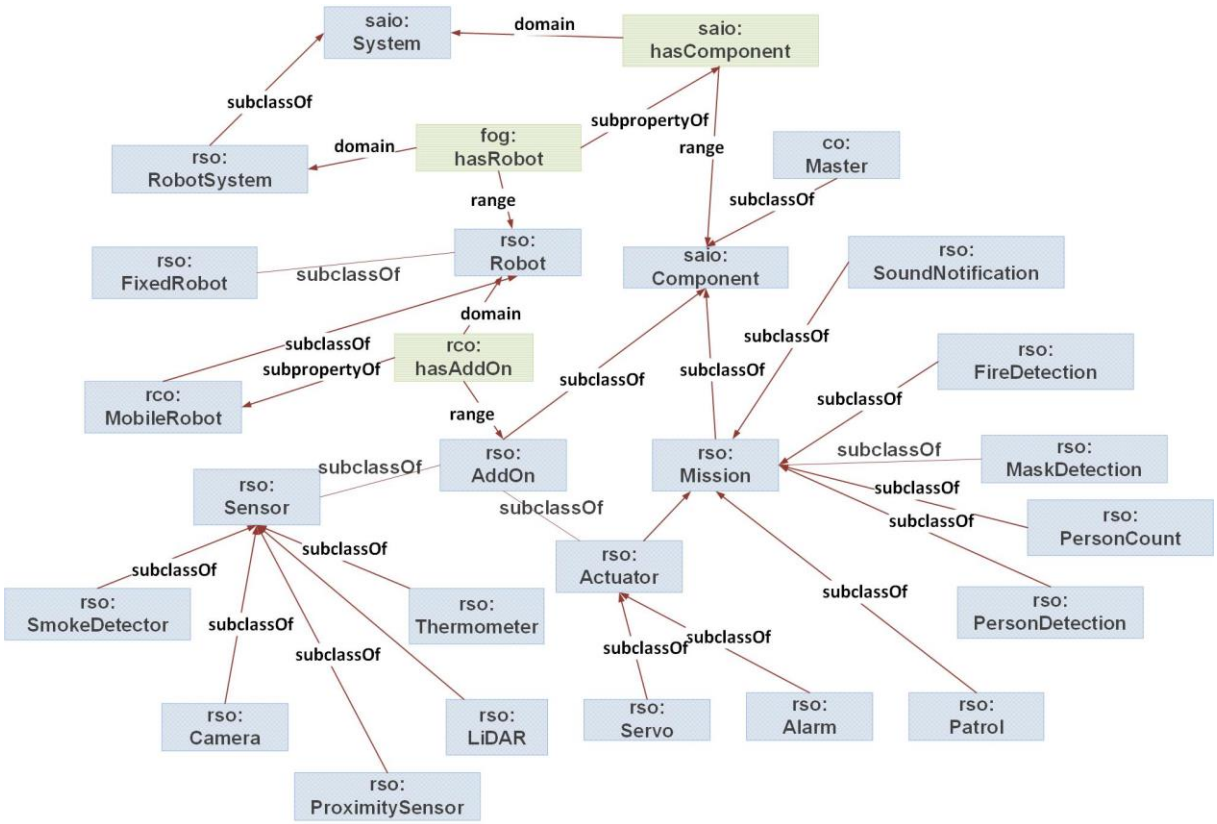
Što se parametara od značaja za robota tiče, oni imaju svoju oznaku modela (poput Turtlebot i Waffle), zatim tip uređaja kontrolne jedinice (Raspberry Pi 3 u našem slučaju za oba robota), trenutno stanje baterije, brzinu kretanja, dimenzije (širina i visina). Dalje, za SSH pristup prilikom ROS inicijalizacije, neophodni su korisničko ime i IP adresa računara odgovornog za upravljanje datim robotom (u našem slučaju su to Raspberry Pi uređaji). Osim toga, kao opcioni parametar se može razmatrati i trenutna lokacija robota zadata kooridnatama *(X, Y, Z)* ili oznaka prostorije u kojoj se nalazi. Konačno, pokretni i fiksni roboti se razlikuju po tome što kod pokretnih figuriše i brzina kretanja kao jedan od parametara, što nije primenjivo za fiksne.

Dalje, ROS gospodar najčešće predstavlja tradicionalni x86 server pod nekom od Linux distribucija. Povrh toga, neophodno je na tom serveru instalirati i ROS serverski softver. Od bitnih karakteristika ove komponente, tu su njegova IP adresa i odgovarajući port koji će biti korišćen za serversku ROS komponentu. Zavisno od verzije ROS-a, šabloni koda koji će biti korišćeni za odgovarajuće komande inicijalizacije i upravljanja robotima se mogu razlikovati.

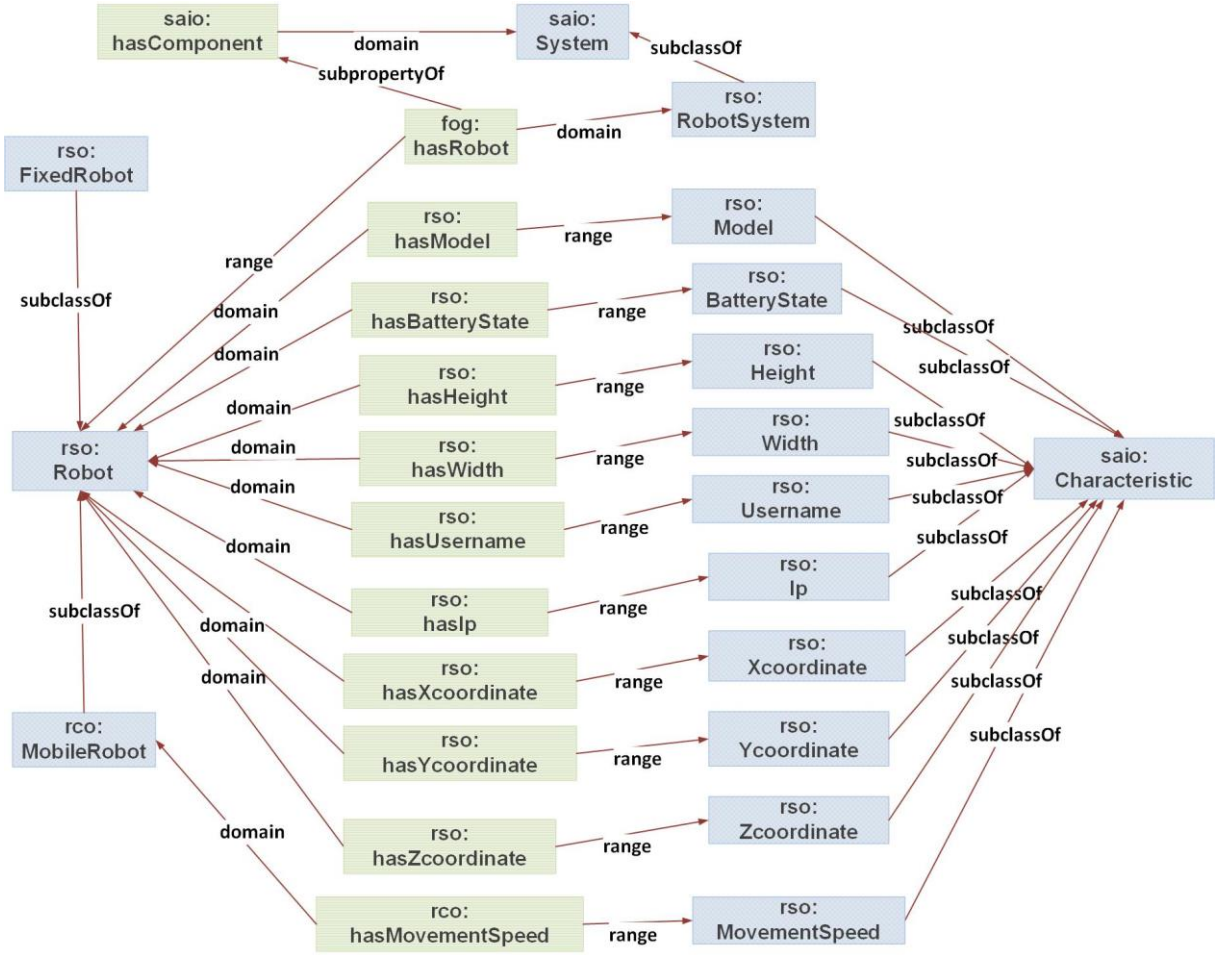
Misija obuhvata skup raznolikih zadataka koje nadzorni roboti mogu da obavljaju, poput patroliranja (obilaska prostorija), detekcije požara, detekcije određenog tipa objekata (poput prisustva ljudi ili određenih predmeta iz okoline), detekcija da li osoba nosi masku (u kontekstu zaštite od COVID-19 tokom pandemije) i slično. Misija poseduje kao opcioni parametar naziv odgovarajuće Python skripte, Java klase ili drugog pomoćnog programa koji omogućava prikupljanje željenih podataka. Na osnovu toga se formira komanda za poziv odgovarajućeg pomoćnog programa. U prethodnim radovima autora, prikazani su primeri implementacije algoritama za detekciju različtih događaja u robotskim nadzornim sistemima upotrebom računarskog vida [125][126]. S obzirom na to što se misija oslanja na upotrebu senzorskih ili aktuatorskih uređaja, potrebno je uzeti u obzir i zahteve po pitanju opremljenosti robota koje

treba da je obave. Prema tome, uvode se još dve karakteristike misije – jedna kojom se definišu neophodni senzori za realizaciju misije, a drugom - neophodni aktuatori. U konkretnom primeru, recimo, kamera je neophodna za detekciju maske i brojanje osoba. Sa druge strane, za indetifikaciju požara se može korsititi bilo kamera, bilo senzor za detekciju dima ili temperaturni sensor. Dalje, za mogućnost patroliranja, neophodan je servo motor koji omogućava kretanje robota i slično.

Što se *AddOn* komponenti tiče, dodaci kojima roboti mogu biti opremljeni sa ciljem realizacije neke misiju mogu biti *Sensor* ili *Actuator*. Po pitanju senzora, na raspolaganju mogu biti različiti dodatni uređaju koji imaju za svrhu da detektuju neke promene u spoljašnjoj sredini, poput temperaturnog senzora, senzora za vlažnost vazduha, kamere, senzor blizine, senzor za detekciju dima, LiDAR i slično. Svaki od senzora meri neku fizičku veličinu, dok rezultate vraća u određenoj jednici mere. Sa druge strane, aktuatori predstavljaju dodatne uređaje poput servo motora za kretanje, termičkih i rashladnih uređaja, zvučnog signala ili alarma kojim mogu doprineti sprovođenju akcije kojima se uzrokuju promene u okruženju i slično. Na **Slici 4.7a** se može videti pojednostavljena ilustracija prethodno opisane ontologije, dok je detaljniji pregled karakteristika robota dat kao dopuna na **Slici 4.7b.**



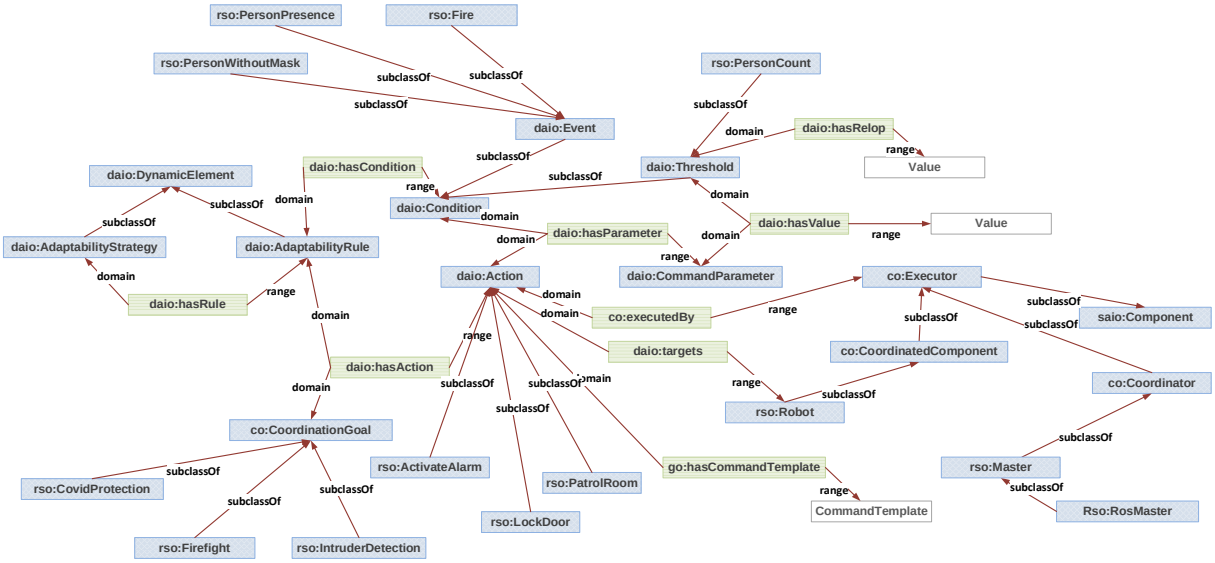
Slika 4.7a Primeno SAIO ontologije na domen robotskih nadzornih sistema – Robot Surveillance Ontology (RSO)



Slika 4.7b Robot Surveillance Ontology (RSO): pregled karakteristika robora

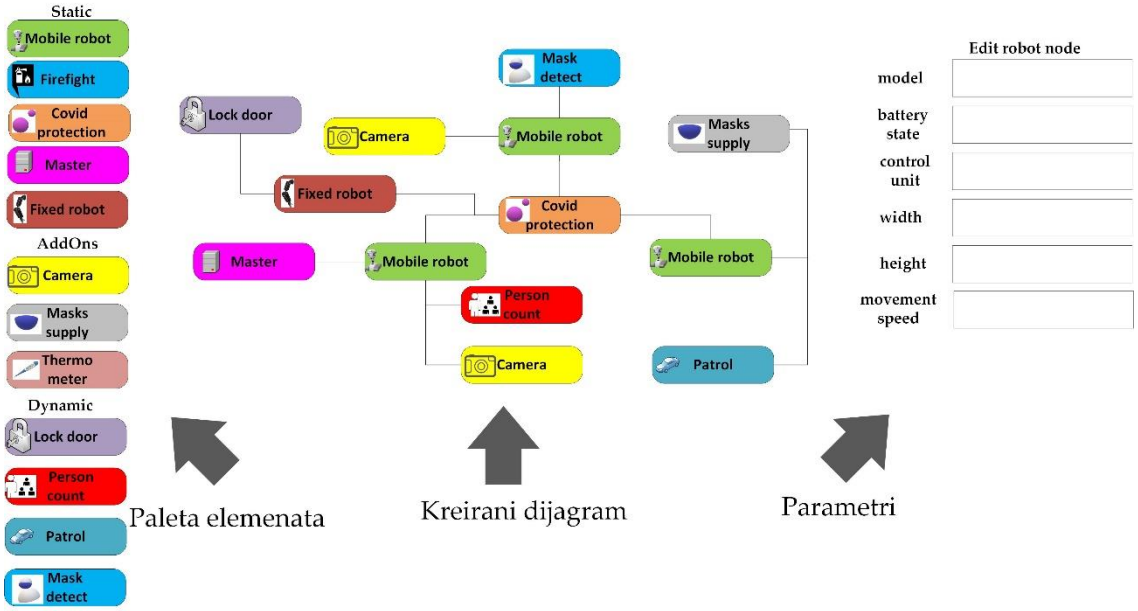
Sa druge strane, što se primene koordinacije tiče, razmatramo sledeći skup primera ciljeva na nivou robotskog nadzornog sistema: 1) *Firefight* – zaštita od požara 2) *CovidProtection* – žaštita od širenja COVID-19 respiratorne infekcije u zatvorenim prostorijama 3) *IntruderDetection* – detekcija neželjenih osoba u prostoriji. Dalje, ilustracija upotrebe DAIO i CO ontologija sa ciljem opisa dimačkih aspekata i koordinacije u robotskom nadzornom sistemu, data je na **Slici 4.8**. Razmatrajući dostupne akcije, žaštita od požara obuhvata detekciju požara uz pomoć robota koji poseduje kameru, senzor dima ili temperaturni senzor, a nakon toga bi kao odgovor bio aktiviran alarm od strane fiksnog robota ili nekog drugog pokretnog robota. Dalje, drugi robot bi obišao prostoriju u slučaju požara i prebrojao prisutne osobe. Sa druge strane, što se zaštite od COVID-19 tiče, pravila adaptacije obuhvataju detekciju osoba koje ne nose masku, ali i brojanje osoba koje ulaze/izlaze iz prostorije. Ukoliko su detektovane osobe bez maske, patrolira robot sa stalkom koji sadrži neiskorišćene maske. Sa druge strane, ukoliko je trenutni broj osoba u prostoriji veći od dozvoljenog, aktivira se zvučni signal ili zatvore vrata. Konačno, u slučaju zaštite od provale, robot u nekoj konkretnoj prostoriji u kojoj

nije dozvoljeno prisustvo osoba, ukoliko kamerom ili senzorom blizine detektuje nečije prisustvo, dolazi do aktivacije alarma i zaključavanja prostorije.



Slika 4.8 Dinamički i aspekti koordinacije robota za nadzor prostorija izraženi uz pomoć DAIO i CO ontologija

Sa druge strane, izgled Node-RED okruženja za kreiranje instanci modela robotskog sistema za nadzor prostorija generisanog na osnovu datih domenski-specifičnih ontologija se može videti na **Slici 4.9**. Paleta elemenata je razdvojena na statičke (robot, misija, master) i dinamičke aspekte koji obuhvataju različite vrste misija. Dati primer prikazuje Node-RED tok u okviru kog su prevučena tri robota sa svrhom zaštite od COVID-19 – jedan detektuje osobe bez maske, drugi broji osobe u prostoriji, a treći patrolira sa kutijom maski da bi dostavio osobama koji ih ne poseduju. S obzirom na to da se misije koje obavljaju prva dva robota oslanjaju na tehnike računarskog vida, kamera im je neophodna kao dodatak da bi bilo moguće izvršenje, dok treći koji patrolira mora da bude robot koji na sebi ima kutiju maski. Četvrti, fiksni robot zaključava vrata ukoliko je broj ljudi veći od dozvoljenog.



Slika 4.9 Node-RED okruženje za modelovanje statičkih i dinamičkih aspekata robotskog sistema za nadzor prostorija

4.2.4 Korišćeni šabloni generisanja koda

U nastavku će biti prikazani ključni šabloni koda korišćeni od strane mehanizma za automatsko generisanje koda, u okviru studije slučaja robotskog nadzornog sistema. **Tabela 4.8** daje pregled najbitnijih šablona korišćenih za generisanje koda u ovoj studiji slučaja, sa ciljem realizacije statičkih i dinamičkih karakteristika robotskog nadzornog sistema, predstavljenih u skladu sa prikazanim semantičkim radnim okvirom ilustrovanog na **Slici 4.7** i **Slici 4.8**. Kao što se može videti, srž primene mehanizama za generisanje koda u ovoj studiji slučaja jeste u parametrizaciji ogovarajućih ROS, ali i pomoćnih Linux komandi, s obzirom da se u našem slučaju pretpostavlja da koristimo ovaj operativni sistem. Opis svakog šablona uključuje, pored bitnih parametara i informaciju ko zapravo treba da izvrši taj kod, da bi se zapravo ostvarila koordinacija u sistemu.

Što se tiče šablona br. 1, njegova je uloga da inicijalizuje ROS gospodara koji kontroliše klaster u kome su povezane upravljačke jedinice robota sa instaliranim ROS-om. Komanda u vrsti br. 1a se koristi za pristup samom serveru koji predstavlja ROS gospodara putem SSH, a zatim podešavaju promeljive okruženja kojima se specificira IP adresa ROS host-a, a zatim specificira i ceo URL serverske ROS komponente gospodara, zajedno sa dodeljenim portom komandama prikazanim u vrstama br. 1b i 1c. Konačno, nakon specifikacije prethodnih parametara, poziva se *roscore* komanda za aktivaciju ROS servisa gospodara (odgovara vrsti br. 1d).

Sa druge strane, šablon br. 2 predstavlja skup naredbi kojim se aktvira ROS na samom robotu i vrši povezivanje na master zadatog klastera. Pored iste sekvence pristupa preko SSH (vrsta br. 2a), podešavanja URL gospodara i IP samog host-a (vrste br. 2c i 2d), potrebno je specificirati i konkretan model robota (vrsta br. 2b). Konačno, nakon ovih konfiguracionih koraka se pokreće ROS komandom *roslaunch* (vrsta br. 2e), dok ostali argumenti zavise od same familije robota, u ovom slučaju Turtle Bot 3.

Dalje, šablonom datim u vrst br. 3 se pokreće skripta kojom je implementirana sama misija – poput brojanaja osoba, detkcije osoba koje ne nose maske, detekcije požara dimnim ili drugim senzorom, izdavanje komande za zaključavanje vrata i slično. Šablon iz vrste br. 4 predstavlja komandu patroliranja robota kojom se zavisno od brzine pomera za veće ili manje rastojanje duž prostorije.

Tabela 4.8 Šabloni koda studije slučaja robotskog nadzornog sistema

Br	Element	Opis	Šablon koda	Izvršava
1	Master	Inicijalizacija ROS gospodara	a) `ssh {Master.hasUser.User}@{Master.hasIp.Ip}`	ROS gospodar
			b) `export ROS_HOSTNAME = {Master.hasIp.Ip}`	
			c) `export ROS_MASTER_URL= http://MASTER_IP:{Master.hasPort.Port}`	
			d) `roscore`	
2	Robot	Pridruživanje robota kubernetes klasteru	a) `ssh {Robot.hasUser.User}@{Robot.hasIp.Ip}`	Robot kojeg želimo pridružiti klasteru
			b) `export TURTLEBOT3_MODEL= {Robot.hasModel.Model}`	
			c) `export ROS_HOSTNAME = {Master.hasIp.Ip}`	
			d) `export ROS_MASTER_URL= http://MASTER_IP:{Master.hasPort.Port}`	
			e) `roslaunch turtlebot3_bringup turtlebot3_robot.launch`	
3	Mission	Pokretanje misije	`roslaunch {Misson.hasScript.Script}`	Robot kome se dodeljuje misija
4	Patrol	Patroliranje robota	`rostopic pub /tb3_0/cmd_vel geometry_msgs/Twist -r 10 -- '[{Robot.hasSpeed.Speed},0.0,0.0]' '[0.0, 0.0, 0.0]'`	Robot koji se pomera

4.2.5 SPARQL upiti za verifikaciju sistema i proveru uslova adaptacije

Što se prikazane studije slučaja tiče, uzimajući u obzir problematiku razmatranog domena nadzora upotrebom višerobotskog sistema, razmatraju se sledeći aspekti verifikacije:

1) dostupnost senzora i aktuatora – svaki robot kome se dodeli neka misija mora da poseduje odgovarajuće senzore, ali i aktuatore koje ona zahteva;

2) lokacija – roboti se moraju nalaziti u istoj prostoriji (soba, sprat ili otvoreni teren) trenutno, da bi kontekst koordinacije bio odgovarajući;

3) dimenzije robota – visina i širina robota koji treba da obavlja misiju u nekoj konkretnoj sobi, zbog prostornih ograničenja mora biti odgovarajućih dimenzija, u datom opsegu (recimo, u slučaju patroliranja po porstoriji sa maskama robot mora biti veće širine od 40 cm), i 4) ostvarenost uslova za koordinaciju – da li je detektovana osoba bez maske ili požar, recimo.

Tabela 4.9 prikazuje odgovarajuće upite za implementaciju prethodno navedenih aspekata verifikacije.

Tabela 4.9 SPARQL upiti za verifikaciju instanci sistema i proveru ispunjenosti uslova pravila adaptacije za studiju slučaja višerobotskog nadzornog sistema

Br.	Aspekat	Upit

1	Dostupnost senzora	PREFIX rso: <http://www.example.com/rso/>
		SELECT DISTINCT ?r ?s1 ?m
		WHERE {
		GRAPH <http://www.example.com/rso> {
		?r rso:hasSensor ?s1.
		?m rso:requiresSensor ?s2.
		FILTER(STR(?s1)=STR(?s2))
		}
		}

2	Dostupnost aktuatora	PREFIX rso: <http://www.example.com/rso/>
		SELECT DISTINCT ?r ?a1 ?m
		WHERE {
		GRAPH <http://www.example.com/rso> {
		?r rso:hasActuator ?a1.
		?m rso:requiresActuator ?a2.
		FILTER(STR(?a1)=STR(?a2))
		}
		}

3	Poklapanje lokacije	PREFIX rso: <http://www.example.com/rso/>
		SELECT DISTINCT ?r1 ?r2 ?l1
		WHERE {
		GRAPH <http://www.example.com/rso> {
		?r1 rso:hasLocation ?l1.
		?r2 rso:hasLocation ?l2.
		?a rso:targets ?r1.
		?a rso:targets ?r2.
		FILTER(STR(?l1)=STR(?l2) &&

		regex(STR(?r1), "robot1") && regex(STR(?r2), "robot2")) } }

4	Ograničenje dimenzija robota	PREFIX rso: <http://www.example.com/rso/>
		SELECT DISTINCT ?r ?m ?cg
		WHERE {
		GRAPH <http://www.example.com/rso> {
		?r rso:hasWidth ?w.
		?cg rso:hasAction ?a.
		?a rso:targets ?r.
		?cg rso:hasConditon ?c.
		?cg rso:hasThreshold ?t.
		FILTER(regex(STR(?m),
		"maskpatrol") &&
		(?w > ?t))
		}
		}

5	Prisustvo osobe bez maske detektovano od strane nekog robota	PREFIX rso: <http://www.example.com/rso/>
		SELECT DISTINCT ?ar
		WHERE {
		GRAPH <http://www.example.com/rso> {
		?ar rso:hasCondition ?c.
		?c rso:hasEvent ?e.
		?ar rso:targets ?r.
		FILTER(regex(STR(?e),
		"PersonWithoutMask")
		&& regex(STR(?e), "robot1"))
		}
		}

6	Ograničenje maksimalnog broja osoba u prostoriji	PREFIX rso: <http://www.example.com/rso/>
		SELECT DISTINCT ?a ?c ?l
		WHERE {
		GRAPH <http://www.example.com/rso> {
		?cg rso:hasAction ?a.
		?cg rso:hasConditon ?c.
		?cg rso:hasThreshold ?t.
		?a rso:targets ?r.
		?r rso:hasLocation ?l.
		FILTER(regex(STR(?m),
		"personcount") &&
		(?w > ?t))
		}
		}

Upit iz vrste br. 1 prikazuje proveru da li postoji robot koji ima neophodan sensor za realizaciju misije. Upit br. 2 daje sličnu proveru za dostupnost aktuatora. Upit iz vrste br. 3

proverava da li su roboti koji učestvuju u istom pravilu adaptacije za neki scenario koordinacije prisutni u istoj prostoriji. Upit br. 4 proverava da li je robot koji bi trebao da patrolira noseći maske zapravo dovoljno širok za ovaj zadatak. Dalje, upit br. 5 prikazuje kako se vrši provera da li je došlo do detekcije osobe koja ne nosi masku od strane nekog robota. Konačno, upit br. 6s proverava da li je broj u nekoj od prostorija veći od dozvoljenog za tu prostoriju.

4.3 Napredni korisnički interfejsi upotrebom proširene stvarnosti

4.3.1 Uvod

Glavna ideja aplikacija proširene stvarnosti jeste kombinacija realnih elemenata (poput prikaza okruženja upotrebom kamere u realnom vremenu), ali, sa druge strane i dodatnih, digitalnih objekata sa kojima korisnik može da na neki način interaguje. Obično se interakcija realnog sveta i fiktivnh objekata realizuje upotrebom optičkih markera ili GPS lokacije. S obzirom, na to da su današnji pametni uređaji većinom opremljeni i GPS modulom i kamerama visoke rezolucije, zajedno sa pristupom brzim mobilnim mrežama, dolazi se do zaključaka da predstavljaju jednu od najpogodnijih platformi za ovakav tip aplikacija [127]. Tokom vremena, proširena stvarnost je našla veliki broj primena i na mobilnim uređajima i na personalnim računarima – od video igara i zabave, preko edukacije, do industrijskih scenarija i scenskog nastupa.

Cilj ove studije slučaja jeste da pokaže primenu prethodno predstavljenog semantičkog radnog okvira za generisanje naprednh korisničkih interfejsa zasnovanih na proširenoj stvarnosti u dva različita domena: 1) daljinsko upravljanje robotima, i 2) izvođenje muzike uživo. Osim statičkih aspekata koji se tiču prisutnih vizuelnih elemenata u aplikaciji, predstavljeni radni okvir pruža mogućnost specifikacije promena uslovljenih pojavom određenih događaja, kao što je prekirvanje markera rukom. Kao rezultat generisanja koda, dobijaju se web aplikacije implementirane upotrebom HTML-a i JavaScript-a, pri čemu se za funkcionalnosti proširene stvarnosti oslanjamo na AR.js biblioteku za JavaScript, koja se uspešno izvršava i u pretraživaču mobilnih telefona. I jedan i drugi primer se zasnivaju na prethodnim radovima autora [128][129].

4.3.2 AR.js biblioteka za razvoj aplikacija proširene stvarnosti

AR.js [130] predstavlja JavaScript biblioteku čija je primarna uloga da omogući intuitivnu implementaciju funkcionalnosti proširene stvarnosti u okviru web stranice i to u samo par linija

HTML i JavaScript koda. Osim toga, ova biblioteka je open-source projekat, a usled svoje jednostavnosti, praktičnosti, intuitivne sintakse, pri čemu su hardverski zahtevi prilično skromni (pokreće se uspešno i na dosta starijim uređajima), stiče tokom godina široku popularnost, pogotovu što se tiče razvoja prototipova. Između ostalog, prosečno je dovoljno desetak linija HTML koda za kompletnu, potpuno funkcionalnu aplikaciju .

Što se podrške tiče, podržava maltene sve široko rasprostranjene web pretraživače, ali uključuje i kompatibilnost sa WebGL i WebRTC tehnologijama, što omogućava besprekoran rad čak i na Android i iOS mobilnim uređajima, što još više povećava upotrebljivost ovog rešenja, s obzirom na pogodnosti primene mobilnih uređaja za ovu svrhu. Konačno, ova biblioteka ne zahteva nikakav dodatni hardver osim kamere za potpuni doživljaj proširene stvarnosti na personalnom računaru ili mobilnom telefonu, a čak i kod vremešnih mobilnih uređaja doseže performanse od 60 slika u sekundi. AR.js objedinjuje i integriše funkcionalnosti više različitih biblioteka i alata za proširenu stvarnost unutar web aplikacija, ali i 3D grafiku, zajedno sa elementima računarske animacije. Među njima su Three.js, a-frame i ARToolKit. Praktično, AR.js se koristi kao *omotač* vrlo visokog nivoa apstrakcije za ove biblioteke, čiji je glavni cilj da olakša implementciju multimedijalno bogatih aplikacija proširene stvarnosti. Između ostalog, AR.js uključuje i pomoćne funkcionalnosti, kao što su mogućnost učitavanja , ali i animacija 3D modela takođe u glTF formatu oslanjajući se na Three.js.

Da bi naša web aplikacija koristila AR.js, neophodno je učitati skriptu *aframe-ar.js*. Zatim, unutar HTML stranice se dodaje scena proširene stvarnosti, koji predstavlja koncept na najvišem nivou u strukturi AR.js aplikacije. Za ovu svrhu se koristi HTML tag `<a-scene embedded arjs>`. Dalje, uzimajući u obzir da će se kao izvor realnih slika koristiti integrisana kamera, onda je potrebno specificirate i atribut kojim se definiše izvor `<a-scene embedded arjs='sourceType: webcam;'>`.

AR.js pruža podršku za dva tipa aplikacija proširene stvatrnosti, zavisno od toga šta se koristi kao okidač u prikazu 3D modela: 1) zasnovana na markerima 2) zasnovana na GPS lokacijama. U ovoj studiji slučaja, primarno je fokus na primeni markera, a to su računarski generisane crno-bele slike koje se obično štampaju i koriste u papirnoj formi, prikačene za neki fizički objekat. Markeri su zapravo jednostvne slike koje se praktično upotrebljavaju kao podloga na kojoj se prikazuju fiktivni 3D objekti kao odgovor na detekciju markera.

Što se vrsta dostupnih markera tiče, u AR.js su ponuđeni standardni simboli – *hiro* i *kanji,* ali i numerički barkodovi. Za ovu svrhu se koristi <*a-marker*> HTML tag. Recimo, numerički barkod vrednosti 3 se definiše unutar scene na sledeći način kao ugnježdeni HTML tag sa

postavljenom vrednošću atributa tipa i odgovarajućom brojevnom oznakom: `<a-marker type='barcode' value='3'></a-marker>`.

Uglavnom, u aplikacijama ovog tipa se kao odgovor na detekciju markera ili lokacije, obično prikazuju različiti 3D objekti. Međutim, da bismo ostvarili vezu markera i 3D modela, potrebno je unutar taga koji odgovara markeru umetnuti i tag koji kreira 3D objekat ili ga učitava iz fajla. U nastavku je data definicija 3D primitive na primeru kocke upotrebom <a-box> taga.

```
<a-box position='0 0.1 0' material='opacity: 0.3; color:red;'>
</a-box>
```

U ovom slučaju *position* atribut služi da definiše koordinate objekta (kocke) u odnosu na poziciju markera. Osim pozicije, postoji mogućnost da se podešavaju vektori skaliranja (*scale*), ali i rotacije (*rotation*). Svi vektori koji figurišu kao parametri AR.js su zadati u formi koja poseduje tri koordinate - *[x, y, z]*. Dalje, moguće je specificirati podešavanja materijala nacrtanih 3D objekata - postavljanjem boje i intenzitet prozirnosti.

Osim jednostavnih 3D oblika, AR.js ima podršku i za rad sa kompleksnim glTF modelima. U nastavku je dat primer kako se obavlja učitavanje 3D modela, kao odgovor na detekciju markera korišćenjem <*a-entity*> taga:

```
<a-entity click1 gltf-model="./assets/models/model1/scene.gltf" rotation="0 45 0" scale="0.2 0.2 0.2" animation-mixer> <a-entity>
```

Kao što se može videti, ovde figuriše atribut *gltf-model*, koji zapravo predstavlja putanju glTF fajla modela koji želimo da prikažemo. Dalje, moguće je aktivirati i unapred kreirane animacije (u nekom eksternom softveru, poput Blender-a) i to dodavanjem parametra *animation-mixer*. Između ostalog, AR.js pruža mogućnost da na jednostavan način kreiramo sopstvene animacije i to postavljanjem vrednosti *animation* atributa. U nastavku je dat primer za rotaciju. Za ovaj postupak, prvi korak je izbor željene transformacije (poput rotacije i skaliranja), zatim podešavanje trajanja, ali i postavka da li se animacija ponavlja (*loop: true*) ili ne.

```
animation="property: rotation; dur: 20000; to: 0 360 0; loop: true"
```

Između ostalog, dodatne resurse i fajlove koje AR.js aplikacija korsiti potrebno je navesti u <*a-assets*> sekciji koja je deo <*a-scene*>. Recimo, za zvučne fajlove se unutar ove sekcije koristi <*audio*> tag, pri čemu src atribut sadrži putanju konkretnog fajla koji odgovara tom resursu.

Konačno, ukoliko želimo da detektujemo klik na 3D model ili dodir od strane korisnika, neophodno je dodeliti *<a-marker>* elementu kursor miša, podešavanjem atributa `cursor='rayOrigin: mouse'`, ali i naziv odgovarajuće *EventListener* metode koja se aktivira kao odgovor na klik *<a-entity>* elementa.

Sa druge strane, za implementaciju AR.js aplikacija zasnovanih na GPS lokacijama su od ključnog značaja sledeće komponente: 1) *gps-camera* – obavezna komponenta za ovaj tip aplikacija (jedna po aplikaciji), pruža apstrakciju kamere sa funkcionalnostima za određivanje relevatnih parametara prikaza, kao što su rotacija i rastojanje objekta zavisno od pozicije posmatrača 2) *gps-entity-place* – dodeljivanje GPS koordinata realnog sveta nekom 3D objektu.

Da bi neka AR.js aplikacija koristila GPS lokacije kao vrstu markera, dodajemo HTML tag kamere na sledeći način.

```
<a-camera gps-camera rotation-reader></a-camera>
```

Za svaku lokaciju od interesa (zadatu geografskom širinom dužinom), dodeljujemo 3D objekat sledećim isečkom koda.

```
<a-entity look-at="[gps-camera]"
          animation-mixer="loop: repeat"
          gltf-model="#animated-model1"
          gps-entity-place="
                     latitude: 42.3313;
                     longitude:22.1332;"
></a-entity>
```

4.3.3 Primena AR.js za razvoj interfejsa namenjenog muzičkom scenskom nastupu

Jedna od glavnih inovacija koju upotreba proširene stvarnosti donosi u oblasti muzičke umetnosti jeste mogućnost upotrebe dodira ili pokreta ruku sa ciljem izvođenja muzike i aktivacije zvučnih efekata, pri čemu je moguće na taj način ostvariti veću ekspresivnost prilikom scenskog nastupa [131].

Što se postojećih primena proširene stvarnosti u oblasti mužičkog izvođenja tiče, različiti instrumenti i interfejsi su do sada predlagani u literaturi. Recimo, [131] prikazuje rešenje koje omogućava da se kompozicije elektronske muzike izvedu interaktivno, uz remiksovanje i modulaciju njihovih elemenata uživo na osnovu manupulacije jednostavnih fizičkih objekata sa markerima. Dalje, u radu [132] je prikazana primena proširene stvarnosti za interakciju sa

publikom. Konačno, u [133] je prikazana igra proširene stvarnosti za motoričku i kognitivnu rehabilitaciju, pri čemu je svakom od markera dodeljen drugi ton, a pacijenti imaju za zadatak da ponove datu sekvencu.

U tom kontekstu, na osnovu prethodno opisanih principa rada AR.js biblioteke, kod kojim se implementira korisnički interfejs za scenski nastup upotrebom unapred snimljenih zvučnih isečaka (*samples, loops*) je dat na **Slici 4.9**. Ceo kod ovog primera se može naći na GitHub linku[17] u okviru naloga autora, kao i u radu [134].

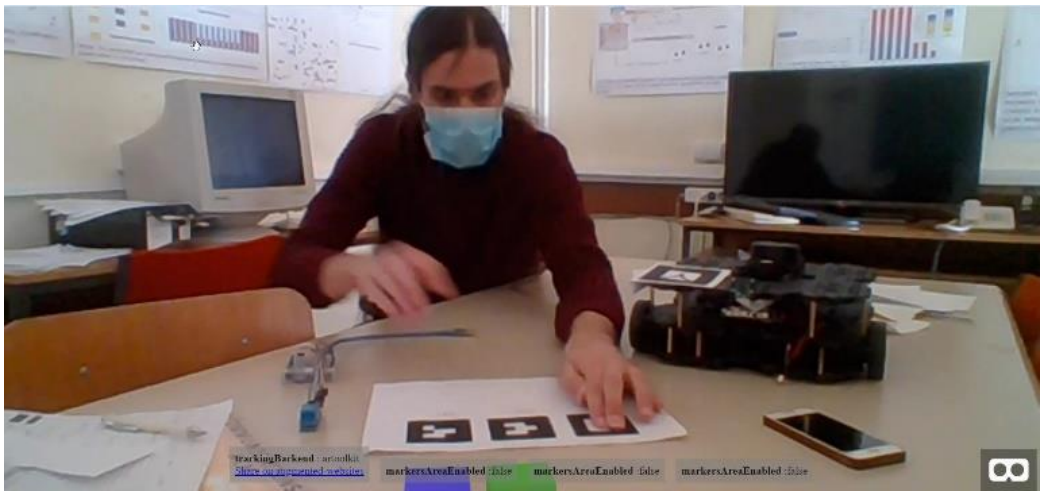
Što se HTML koda tiče, aplikacija je struktuirana tako da unutar a-scene ima više od jednog barkod markera (*<a-marker>* tag). Svakom od *<a-marker>* elemenata je dodeljen različit audio klip, a lista svih zvučnih fajlova koji se koriste je definisana u <a-assets> sekciji unutar scene. Osim toga, unutar svakog od markera je ugnježden i *<a-box>* element, tako da se prikazuje 3D kocka različite boje koja odgovara različitom zvučnom fajlu, radi lakšeg raspoznavanja od strane izvođača.

```
<script src="https://aframe.io/releases/0.8.0/aframe.min.js"></script>
<script src="https://cdn.rawgit.com/jeromeetienne/AR.js/1.6.0/aframe/build/aframe-ar.js"></script>
<script>

AFRAME.registerComponent('soundhandler', {
    tick: function () {
        var m1 = document.querySelector('#sound1');
        ...
        var mN = document.querySelector('#soundN');

        if (document.querySelector('#marker1').object3D.visible == true) {
             m1.play();
        } else {
            m1.pause();
        }
        ...
        if (document.querySelector('#markerN').object3D.visible == true) {
            mN.play();
        } else {
            mN.pause();
        }

    }
});
</script>

<body style='margin : 0px; overflow: hidden;'>
     <a-scene embedded arjs='sourceType: webcam; detectionMode: mono_and_matrix; matrixCodeType: 3x3;'>
          <a-assets>
               <audio id="sound1" src="sample1.wav" preload="auto"></audio>
               ...
               <audio id="soundN" src="sampleN.wav" preload="auto"></audio>
          </a-assets>

          <a-marker id='marker1' type='barcode' value='1'  cursor='rayOrigin: mouse'>
               <a-box id='m1' position='0 0.5 0'  material='opacity: 0.5; color:red;' soundhandler>
               </a-box>
          </a-marker>
                    ...
          <a-marker id='markerN' type='barcode'  value='N' cursor='rayOrigin: mouse'>
               <a-box id='mN' position='0 0.5 0' material='opacity: 0.5; color:blue;' soundhandler>
               </a-box>
          </a-marker>

</body>
```

Slika 4.9 Isečak koda AR.js korisničkog interfejsa namenjenog muzičkom scenskom nastupu

[17] https://github.com/penenadpi/ARjsLoopSampler

Sa druge strane, na **Slici 4.10** se može videti kako izgleda primena od strane izvođača. Kada papir sa markerima uđe u vidno polje kamere, za svaki od njih se prikazuje 3D objekat različite boje. Ovaj korisnički interfejs daje mogućnost da se prelaženjem ruke preko različitih markera i njihovim zakljanjanjem aktivira odgovarajući zvučni fajl, pri čemu kocka koja odgovara aktiviranom zvučnom fajlu nestaje sa scene. Ponovnim otkrivanjem markera se zaustavlja reprodukcija odgovarajućeg zvučnog fajla. Da bi ovakav način rada bio moguć, neophodno je definisati i event handler u JavaScrpitu koji će upravljati redprodukcijom zvučnog fajla (*play* ili *pause*), zavisno od vidljivosti 3D objekta koji odgovara markeru. U praksi, praktična primena ovakvog zvučnog interfejsa za izvođenje muzičke numere uživo se može videti u okviru video snimka[18] javno dostupnog na YouTube-u, postavljenog od strane autora.



Slika 4.10 Primena AR.js korisničkog interfejsa namenjenog muzičkom scenskom nastupu [117]

4.3.4 Primena AR.js za razvoj interfejsa namenjenog daljinskom upravljanju mobilnih robota

Iako glavna prednost mnogih scenarija primene robota upravo jeste u njihovoj kompletnoj autonomiji, pri čemu za njihovo funkcionisanje nije potrebna intervencija operatera i dalje postoje slučajevi koji se zasnivaju na direktnom zadavanju komandi od strane ljudi. Među takvim scenarijima jesu nadzor manje pristupačnih prostorija. Prema tome, cilj drugog primera korisničkog interfejsa u proširenoj stvarnosti uz pomoć AR.js jeste aplikacija slične postavke kao prethodna, s tim da umesto aktivacije različitih zvukova prelaženjem preko markera, u

[18] https://www.youtube.com/watch?v=LL2yGAJ06UQ

ovom slučaju imamo izdavanje komandi za kretanje robota zasnovanih na prethodno opisanoj ROS platformi.

U ovom konkretnom primeru, imamo zadavanje komandi uz pomoć ruku i papira sa odštampanim barkod markerima, kao što prikazuje **Slika 4.11**. Za ovu svrhu, korišćena su tri različita markera, a svakom od njih je dodeljena 3D kocka odgovarajuće boje i služi za aktivaciju jedne od komandi iz skupa dostupnih: 1) crvena - kretanje robota unazad 2) zelena – kretanje robota unapred 3) plava – okretanje robota. Video snimak koji prikazuje demonstraciju[19] ove aplikacije je javno dostupan na YouTube-u. Dalje, prikazuje kako izgleda upotreba ove aplikacije u praksi – korisnik zadaje komande rukama, zakljanjajući markere, a robot se, zavisno od od toga koji je marker zaklonjen kreće napred, nazad ili menja pravac kretanja (rotira).

Što se implementacije tiče, isečak AR.js koda prikazanog primera korisničkog interfejsa za upravljanje mobilnim robotima na daljinu, dat je na **Slici 4.12**. Kao što se može videti, svakom od barkod markera odgovara kocka različite boje. Zakljanjanjem kocke se aktivira event handler u JavaScript-u koji, zavisno od broja koji odgvoara vrednosti barkod markera aktivira poziv ka udaljenom ROS gospodaru, koji upravlja robotima.

Za ovu svrhu, implementirana je i pomoćna Node.js skripta koja se pokreće na ROS gospodaru, oslanjajući se na *roslibjs* biblioteku za JavaScript programski jezik, pri čemu je isečak koji prikazuje njenu implementaciju dat na **Slici 4.13**. Kao što se može videti, ukoliko kod zahteva odgovara broju jedan, onda se aktivira poziv ROS komande za kretanje robota unazad na gospodaru, koji ujedno i pokreće pored ROS komponenti i Node.js server. Slično, komanda broj dva pokreće robota unapred, dok se treća koristi za promenu pravca kretanja robota.

[19] https://www.youtube.com/watch?v=GOE_B0wrMJM

Slika 4.11 Primena AR.js korisničkog interfejsa namenjenog daljinskom upravljanju mobilnih robota: a) papir sa markerima b) kretanje robota [129]

```
<script src="https://aframe.io/releases/0.8.0/aframe.min.js"></script>
<script src="https://cdn.rawgit.com/jeromeetienne/AR.js/1.6.0/aframe/build/
aframe-ar.js"></script>
<script>

var zahtev=0;
const sleep = (milliseconds) => {
  return new Promise(resolve => setTimeout(resolve, milliseconds))
}

AFRAME.registerComponent('roscommhandler', {
    tick: async function () {

    if (document.querySelector('#marker1').object3D.visible == true &&
zahtev==0) {
     zahtev=1;
     const http=new XMLHttpRequest();
     const url='http://192.168.234.94:3000/komanda=1';

     sleep(1000).then(() => {
         http.open("GET",url);
         http.send();
         zahtev=0;
     })

     }
     ... else {
     if (document.querySelector('#marker3').object3D.visible == true &&
zahtev==0) {
     zahtev=1;
         const http=new XMLHttpRequest();
     const url='http://192.168.234.94:3000/komanda=3';

     sleep(1000).then(() => {
         http.open("GET",url);
         http.send();
             zahtev=0;
     })
     }
     }
}
}
});
</script>

<body style='margin : 0px; overflow: hidden;'>
    <a-scene embedded arjs='sourceType: webcam; detectionMode:
mono_and_matrix;          matrixCodeType: 3x3;'>

    <a-marker id='marker1' type='barcode' value='1'  cursor='rayOrigin:
mouse'>
            <a-box id='m1' position='0 0.5 0'  material='opacity: 0.5;
                color:red;' roscommhandler>
            </a-box>
        </a-marker>

    <a-marker id='marker2' type='barcode'  value='2' cursor='rayOrigin:
mouse'>
            <a-box id='m2' position='0 0.5 0' material='opacity: 0.5;
                    color:green;' roscommhandler>
            </a-box>
    </a-marker>

    <a-marker id='marker3' type='barcode'  value='3' cursor='rayOrigin:
mouse'>
            <a-box id='m3' position='0 0.5 0' material='opacity: 0.5;
                color:blue;' roscommhandler>
            </a-box>
    </a-marker>

    </a-scene>
</body>
```

Slika 4.12 Isečak koda AR.js korisničkog interfejsa za daljinsko upravljanje robota

```
const server = http.createServer((req, res) => {
...
const query = url.parse(req.url, true);
const path=query.path;
const command=querystring.parse(path);
const ros_command=command["/command"];

   if(ros_command==1)
   {
          exec.exec("timeout 2s rostopic pub /tb3_0/
cmd_vel geometry_msgs/Twist -r 10 -- '[-0.14,0.0,0.0]'
'[0.0, 0.0, 0.0]'", (error, stdout, stderr) => {
                console.log('stdout: ${stdout}');
          });

     }
   ...
   if(ros_command==3)
   ...
}).listen(port, ip);
```

Slika 4.13 Pomoćna Node.js skripta na ROS gospodaru koja se oslanja na roslibjs

4.3.5 Primena AR.js za aplikaciju namenjenu obilascima turističkih destinacija u doba pandemije COVID-19

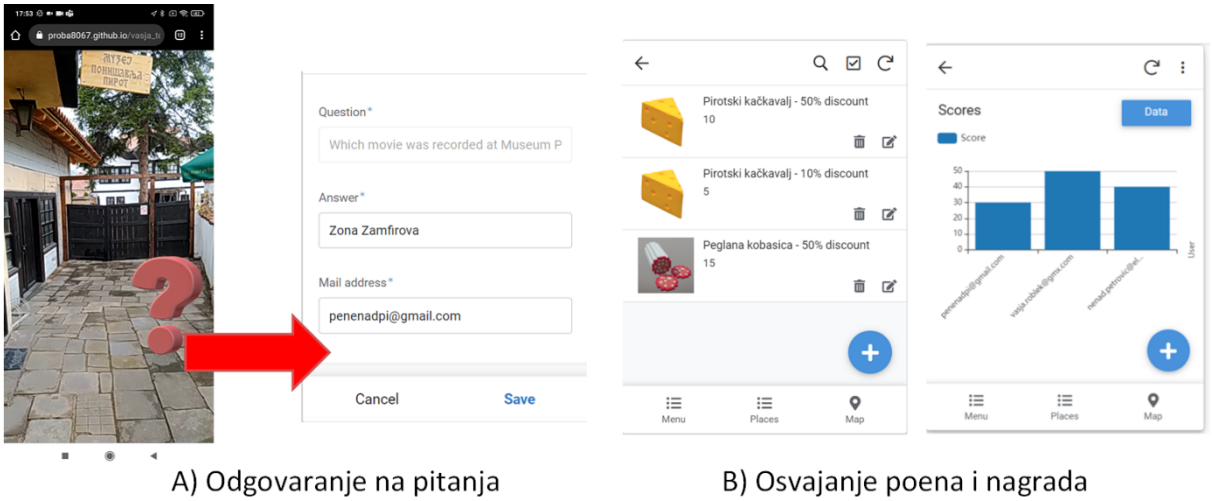
Ovaj primer, koji dopunjuje prethodne, je zasnovan na GPS lokacijama. Cilj ove mobilne web aplikacije jeste dodavanja *gamification* elemenata tokom obilaska turističkih znamenitosti, da bi se motivisali turisti po pitanju samostalnog istraživanja destinacije, uz interaktivne i takmičarske elemente.

Obilaskom neke destinacije i njene okoline, turisti otkrivaju dva tipa fiktivnih objekata dodeljenih GPS lokacijama: 1) *info* – pružaju informacije o znamenitostima 2) *upitnici* – preusmeravaju na pitanja o znamenitostima. Na ovaj način, turisti obilaskom info objekata mogu doći do informacija o okruženju, koje kasnije mogu iskoristiti za davanje odgovora na postavljena pitanja. Ideja je da se tačnim odgovorima na pitanja skupljaju bodovi, koje korisnici kasnije mogu iskoristiti za ostvarivanje popusta prilikom kupovine lokalnih zanatskih proizvoda. Nagrade se plaćaju sakupljenim poenima, a dostavljaju korisniku na e-mail adresu.

Što se dostupnih pitanja tiče, dva tipa su obuhvaćena: 1) tačno/netačno – kratka pitanja gde je odgovor da ili ne 2) višestruki izbor. Što se prikaza pitanja i informacija o objektima tiče, ovaj primer se oslanja na AppSheet[20] platformu za low-code pristup u razvoju multiplatformskih mobilnih aplikacija oslanjajući se na Google Sheets dokumente. Sa druge strane, za proveru tačnosti odgovora, obračun poena i generisanje kupona oslanjamo se na

[20] https://www.appsheet.com/

Google Apps Script radni okvir, kao što je prikazano u [135]. Prema tome, kada se prikaže fiktivni objekat na ekranu, klikom ili dodirom na ekran od strane korisnika se vrši preusmeravanje na odgovarajuću AppSheet stranicu. Snimci ekrana aplikacije su dati na **Slici 4.14**.



Slika 4.14 Primena AR.js korisničkog interfejsa namenjenog interaktivnom turističkom obilasku: a) pitanja o znamenitostima b) poeni i nagrade [135]

Po pitanju generisanja samih pitanja, i za ovu svrhu se koriste semantički-vođeni mehanizmu zasnovani na ontologijama za različite vrste pitanja, što je van opsega ove disertacije. Dodatni detalji o implementaciji po ovom pitanju se mogu naći u drugim radovima autora, kao što su [136] i [137] . Za ovu svrhu, kao ulazne podatke imamo ili tekst slobodne forme (u slučaju pitanja iz istorije i geografije) ili zapise iz eksterne baze podataka (kao što je zdravstveni informacioni sistem), a obe vrste ulaza se semantički anotiraju u skladu sa ontologijama za specifičnu vrstu pitanja (višestruki izbor, tačno/netačno, dopuniti rečenicu i slično). Kao pomoćni korak, u prvom slučaju se koristi procesiranje upotrebom tehnika obrade govornog jezika, dok u drugom slučaju imamo mapiranje relacione baze podataka na domenski-specifične ontologije, a zatim na ontologije pitanja.

Konačno, AR.js isečak koda koji je neophodno dodati HTML stranici, sa ciljem dodeljivanja info objekta nekoj GPS lokaciji, dat je na **Slici 4.15**.

```
<html>
...
<script>
    AFRAME.registerComponent('clickhandler', {
        init: function() {
            this.el.addEventListener('click', () => {
                    const url='https://www.appsheet.com/loc1_info';
                    window.location.replace(url);
            });
    }});
</script>

<a-scene
cursor='rayOrigin: mouse; fuse: true; fuseTimeout: 0;'
raycaster="objects: [clickhandler];"
vr-mode-ui="enabled: false"
embedded
arjs='sourceType: webcam; debugUIEnabled: false;'>
     ...
     <a-entity clickhandler gltf-model="./assets/models/model1.gltf"
     rotation="0 0 0" scale="0.1 0.1 0.1"
     gps-projected-entity-place="latitude: 43.1604136; longitude: 22.5915271;"
     animation-mixer/>
     <a-camera gps-projected-camera rotation-reader>
     </a-camera>
</a-scene>
...
</html>
```

Slika 4.15 Isečak AR.js koda sa primerom proširene stvarnosti zasnovane na lokacijama

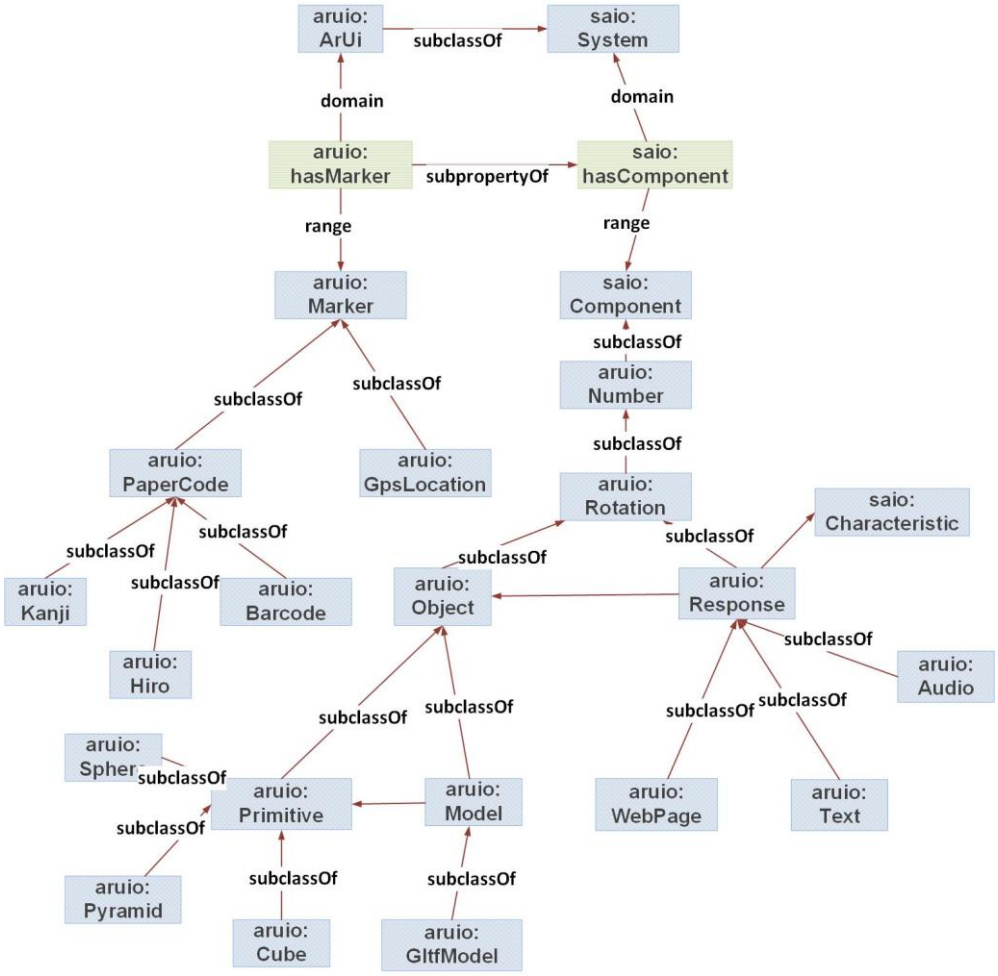
4.3.6 Primena IntisOnt semantičkog radnog okvira za realizaciju naprednih korisničkih interfejsa proširene stvarnosti

U sistemu naprednog korisničkog interfejsa proširene stvarnosti (naveden kao *aruio:ArUi*, izveden iz *saio:System*), mogu se uočiti sledeće relevantni statički aspekti (**Slika 4.16a**):

1) *Marker* – predstavlja okidač koji aktivira prikaz fiktivnog 3D objekta u aplikaciji, a dalje može biti kod odštampan na papiru (*PaperCode*) ili neka zadata GPS lokacija (*GpsLocation*);

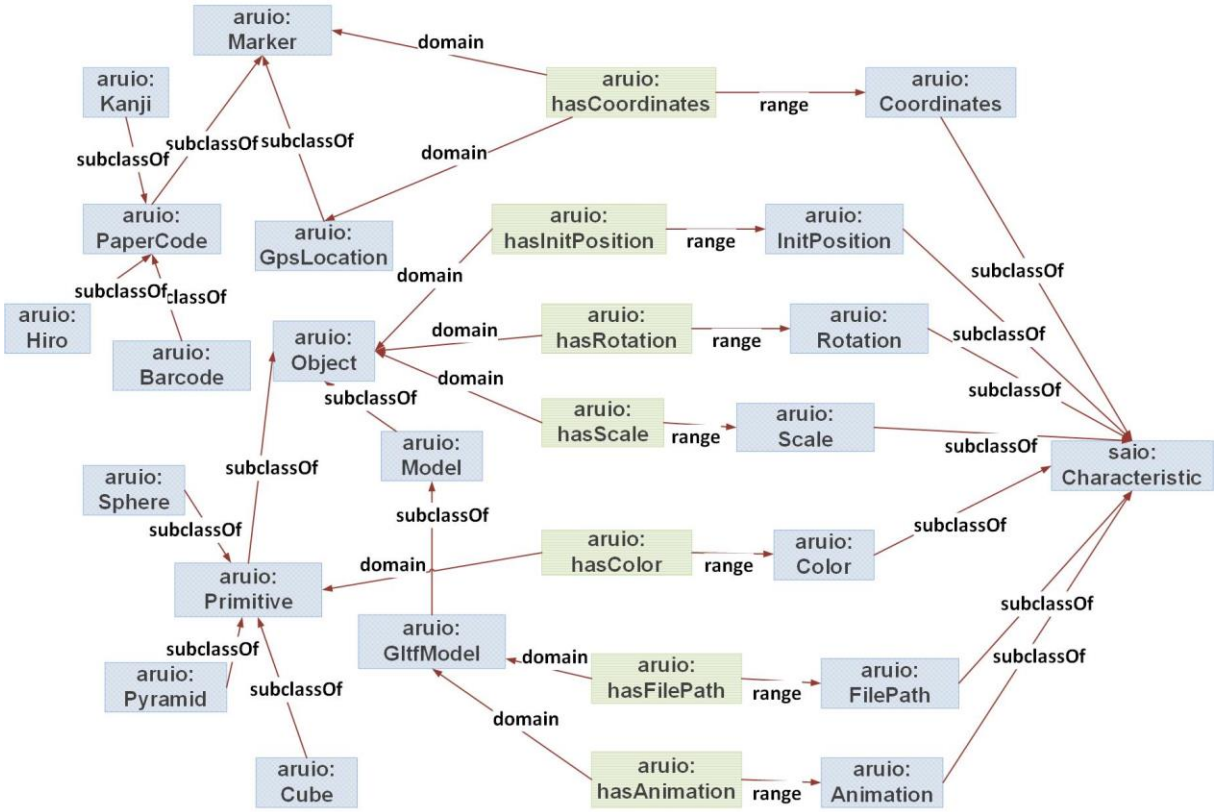
2) *Object* – predstavlja fiktivni objekat koji se pojavljuje kada se detektuje unapred definisani marker, uglavnom se odnosi na neku 3D primitivu poput kocke i lopte (*Primitive*) ili kompleksniji 3D model koji se učitava iz eksternog fajla (*Model*), a zavisno od formata možemo imati i podklase modela, recimo *GltfModel* za prethodno pomenuti standardni format, i

3) *Response* – predstavlja produkt interakcije korisnika i objekata, a može biti različite prirode, zavisno od scenarija upotrebe aplikacije – zvučni fajl (*Audio*), tekstualna poruka (*Text*), web stranica (*WebPage*).



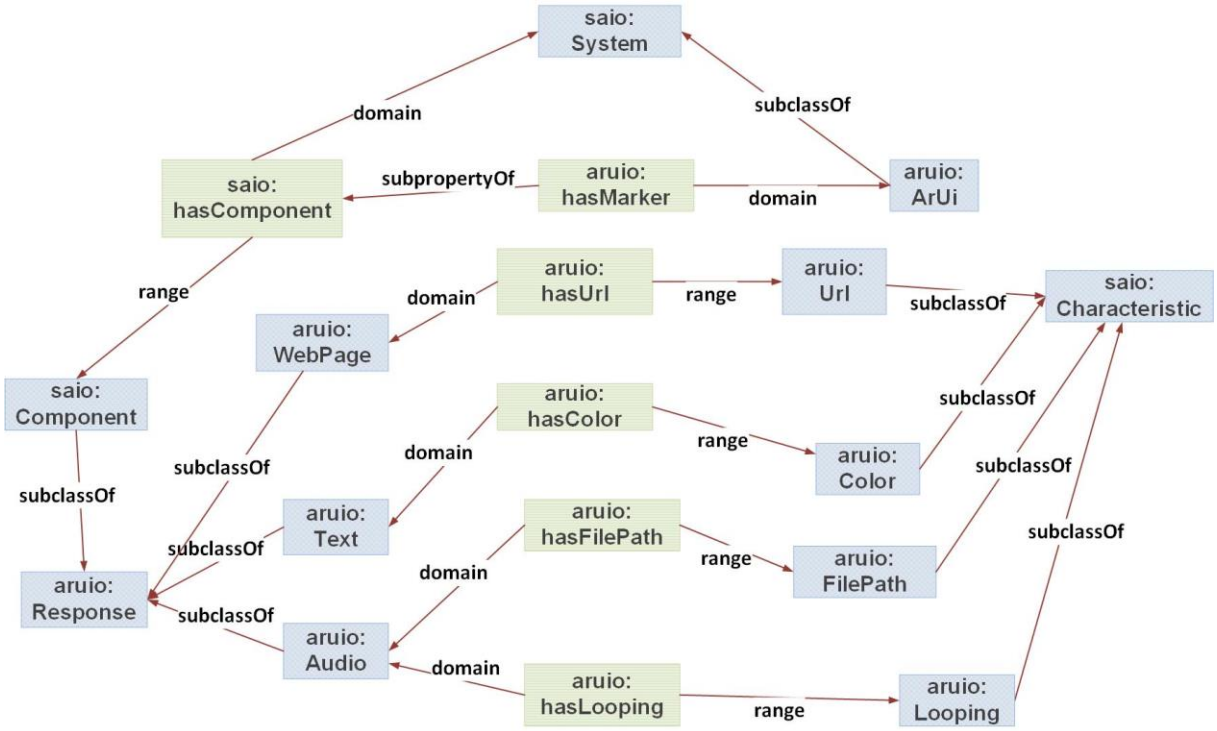
Slika 4.16a Primeno SAIO ontologije na domen naprednih korisničkih interfejsa proširene stvarnosti-Augmented Reality User Interface Ontology (ARUIO)

Što se papirnih markera tiče, oni mogu biti neki od standardnih (*hiro*, *kanji*) ili numerički barkod, tako da je vrsta markera jedna od njegovih značajnih karakteristika, jer se na taj način mogu razlikovati jedan od drugog u jednoj aplikaciji **(Slika 4.16b)**. U slučaju proširene stvarnosti zasnovanoj na lokacijama, za marker su od značaja GPS koordinate. Za svaki od markera se mapira odgovarajući objekat koji se pojavljuje detekcijom nekog konkretnog oblika ili dolaskom do neke unapred definisane lokacije. Sa druge strane, za svaki od objekata koji se pojavljuju su od značaja i njihova početna pozicija u odnosu na marker, rotacija i faktor skaliranja. Za jednostavne 3D primitive je značajan parametar i koje su boje. Zavisno od oblika primitive, imamo podklase kao što su piramida, kocka i sfera. Osim toga, za eksterne 3D modele je od važnosti i putanja na fajl. Dalje, kao opcioni parmetar jeste i to da li je uključena njegova animacija ili ne.



Slika 4.16b ARUIO ontologija: pregled karakteristika komponenti

Sa druge strane, za odgovore na interakciju imamo relevantni skup karakteristika zavisno od razmatrane podvrste **(Slika 4.16c)**. Za audio fajl, to je putanja i da li se ponavlja reprodukcija (loop) ili ne. Dalje, za tekstualni odgovor je od značaja koje je boje, koji se font koristi i sam ispisani sadržaj. Konačno, za web stranicu imamo URL, ali i skup parametara koji se može proslediti slanjem odgovarajućeg HTTP zahteva u samoj aplikaciji.



Slika 4.16c ARUIO ontologija – nastavak pregleda karakteristika komponenti

Dalje, po pitanju ponašanja sistema, razmatramo sledeće akcije u okviru predstavljenih studija slučaja:

1) *PlaySound* – reprodukcija zvučnog fajla od početka do kraja;

2) *LoopSound* – ponavljanje zvučnog fajla;

3) *HideObject* - skrivanje 3D objekta;

4) *RunAnimation* – pokretanje animacije;

5) *RobotGoForward* – kretanje robota unapred;

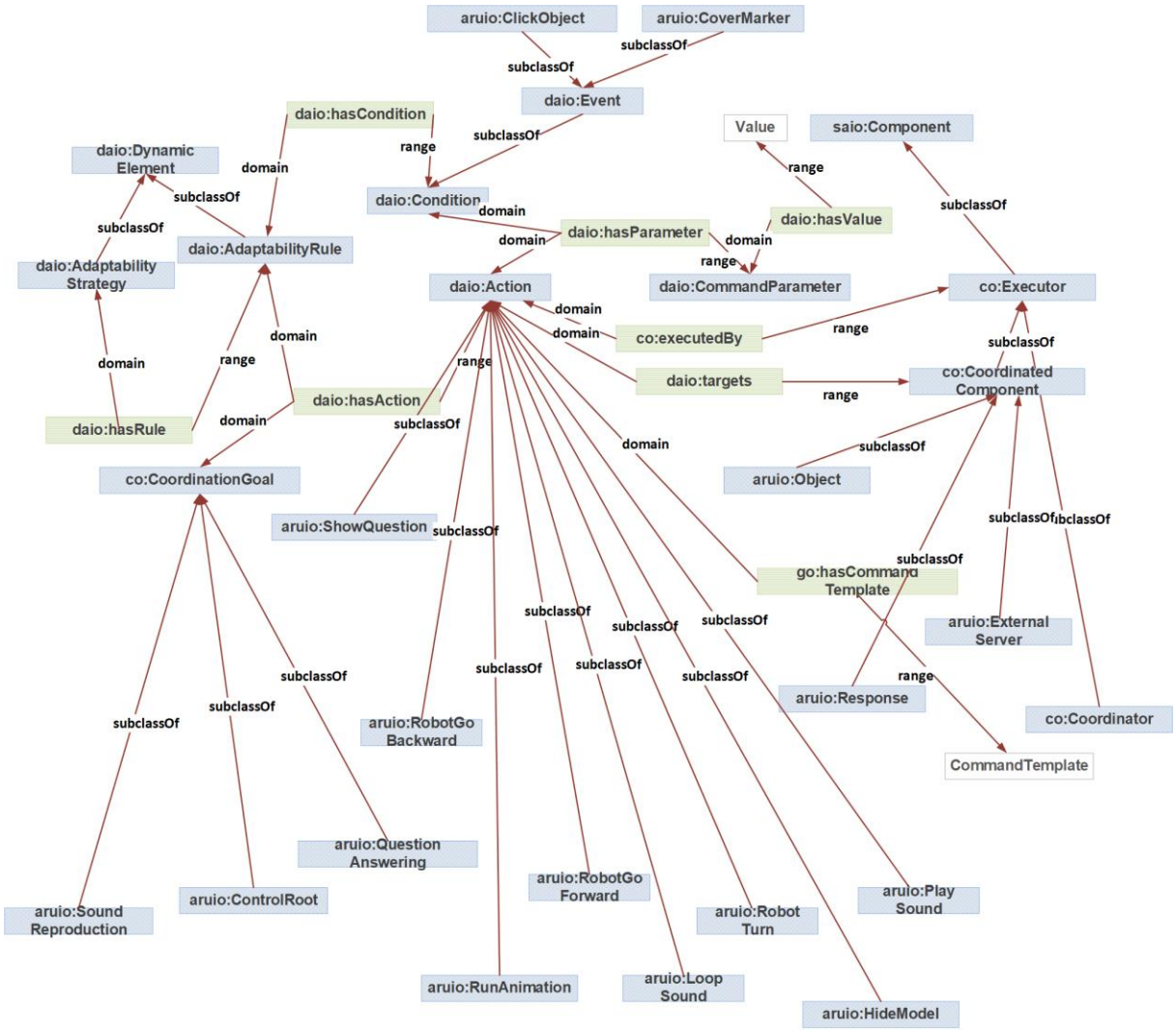
6) *RobotGoBackward* – kretanje robota unazad, i

7) *RobotTurn* – promeniti pravac kretanja robota.

Po pitanju mete ovih akcija, to može biti neki od objekata, odgovor na interakciju ili čak spoljašnji server koji treba da izvrši datu komandu. Ukoliko ne figuriše eksterni server, pretpostavka je da se sve komande izvršavaju na istom računaru koji pokreće web server sa AR.js aplikacijom proširene stvarnosti. Ovakav slučaj imamo kod upravljanja robota, gde imamo ROS gospodara koji je odgovoran za izdavanje komandi.

Dalje, ilustracija upotrebe DAIO i CO ontologija sa ciljem opisa dimačkih aspekata sistema naprednih korisničkih interfejsa u proširenoj stvarnosti, data je na **Slici 4.1**7. Za svaki od ovih scenarija je uslov zapravo prekrivanje odgovarajućeg markera (*CoverMarker*). Pored ovih akcija, ontologijom je obuhvaćen i slučaj primene proširene stvarnosti zasnovane na lokacijama. U tom slučaju, obuhvatamo i scenario prikaza web stranice sa pitanjem u vezi neke

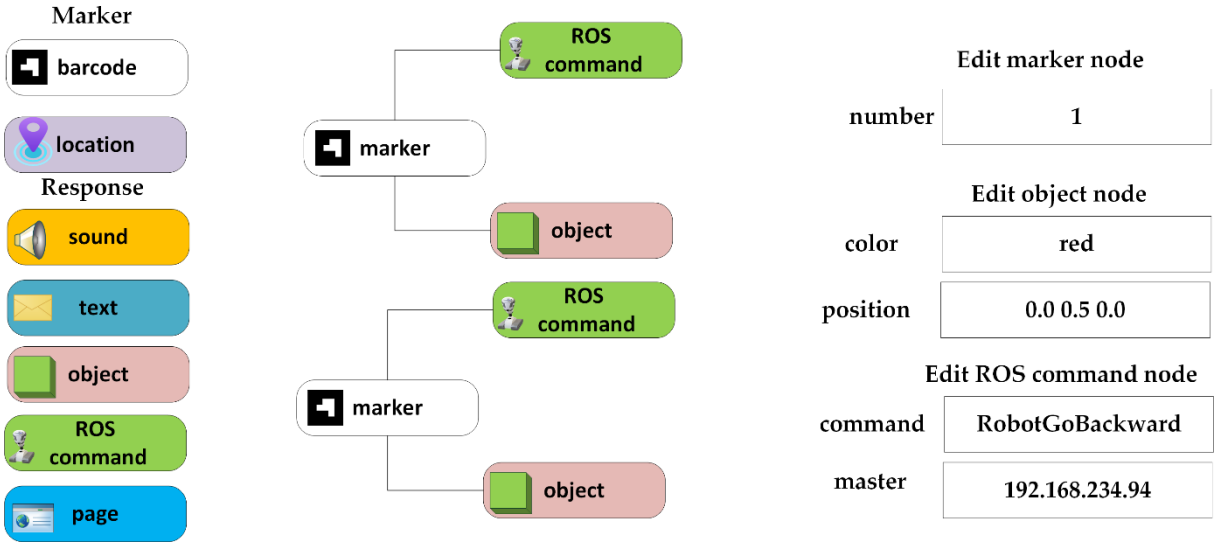
znamenitosti ili objekta od značaja na nekoj geografskoj lokaciji, što je deo aplikacija proširene stvarnosti u domenu turizma iz [135]. Za ovaj scenario je uslov aktivacije zapravo klik na 3D objekat (*ClickObject*) koji se pojavljuje kada turista dođe do određene loakcije zadate GPS koordinatama. Prethono opisani elementi se mogu iskoristiti, generalno, za realizaciju tri scenarija: reprodukcija zvučnih fajlova, upravljanje robotima i postavljanje pitanja.



Slika 4.17 Dinamički aspekti naprednih korisničkih intefejsa računarstva u magli izraženi uz pomoć DAIO i CO ontologija

Konačno, izgled Node-RED okruženja za kreiranje instanci modela sistema naprednih korisničkih interfejsa proširene stvarnosti je prikazan na **Slici 4.18**. Paleta elemenata je razdvojena na markere (marker, objekat) i odgovore (reprodukcija zvuka, komanda namenjene ROS robotima, prikaz web stranice sa pitanjem). Dati primer prikazuje Node-RED tok kojim predstavljamo korsinički interfejs proširene stvarnosit od dva barkod markera. Za svaki od njih se prikazuje kao dodeljeni objekat kocka različite boje, a kao odgovor nakon zaklanjanja

markera rukom aktivira različita komanda za upravljanje robotima, slično kao što je opisano u primeru iz prethodne podsekcije.



Slika 4.18 Node-RED okruženje za kreiranje naprednih korisničkih interfejsa proširene stvarnosti oslanjajući se na ontologije i automatsko generisanje koda

4.3.7 Korišćeni šabloni generisanja koda

U ovom delu je dat opis korišćenih šablona koda za studiju slučaja naprednih korisničkih interfejsa proširene stvarnosti. U **Tabeli 4.10** se može videti pregled najbitnijih isečaka koda koji su korišćeni za automatsko generisanje prethodno opisanih aplikacija. Na osnovu ove tabele, moguće je uočiti da se generisanje koda ovde pretežno svodi na parametrizaciju HTML i JavaScript isečaka koda koji se oslanjaju na elemente AR.js biblioteke.

Šablon iz vrste br.1 služi da HTML isečak kojim se dodaje barkod marker zadate vrednosti u aplikaciju, pri čemu se za njega vezuje odabrani objekat, koji se umeće kao ugnježđeni tag, sa svim svojim parametrima. U ovom slučaju, podrazumeva se da je uslov odgovarajućeg pravila adaptacije koji cilja barkod marker zapravo zaklanjanje samog markera.

Dalje, šablon prikazan u vrsti br. 2 definiše kako se dodaje primitiva u aplikaciju na primeru kocke, pri čemu se koriste parametri boje i početne pozicije, a zatim i njoj dodeljuje event handler, što zavisi od odgovarajuće akcije koja se nad njom primenjuje.

Što se akcija tiče, vrsta br. 3 pokazuje isečak JavaScript event handler koda, koji se dodaje u slučaju kada imamo akciju za kretanje robota unazad. Ovim handler-om, koji je dodeljen nekom od markera, definiše se da svaki put kada se zakloni taj marker, dolazi do slanja GET HTTP zahteva ROS gospodaru čija je IP adresa definisana. Zavisno od specifičnog slučaja

komande kao parametar zahteva se šalje i odgovarajući broj komande: 1-kretanje unazad 2-kretanje unapred 3-skretanje robota.

Još jedna od mogućih akcija je reprodukcija zvučnog fajla, kome odgovara takođe deo JavaScript koda handler-a, koji se dodeljuje odgovarajućem markeru, što je prikazano u vrsti br. 4. Ovaj isečak specificira da ako se promeni stanje vidljivosti 3D objekta dodeljenog nekom markeru, to utiče i na reprodukciju zvuka na sledeći način: 1) ako je objekat prethodno bio vidljiv – nakon zaklanjaja, nestaje objekat, a zatim se inicira reprodukcija odgovarajućeg zvuka 2) ako objekat nije vidljiv – nakon otkrivanja markera se objekat ponovo prikazuje, ali se zaustavlja reprodukcija zvuka.

Sa druge strane, razmatrajući audio fajl kao jedan od mogućih odgovora AR.js aplikacije, vrstom br. 5 se definiše nova <*audio*> stavka u <*a-assets*> sekciji, pri čemu njena putanja odgovara samom zvučnom fajlu koji će biti reprodukovan.

Šablon iz vrste br. 6 služi za ubacivanje drugog tipa markera, koji predstavlja GPS lokaciju. Ključni parametar jesu GPS koordinate koje odgovaraju ovom markeru. Osim toga, potrebno je dodeliti i odgovarajući objekat lokaciji koji će biti prikazan kada se korsnik nađe u njenoj neposrednoj blizini. U tom kontekstu, postoji i parametar za skaliranje, rotaciju i animaciju odgovarajućeg 3D modela objekta, a pretpostavljeni uslov odgovarajuće akcije koja cilja marker jeste klik na sam objekat.

Konačno, šablon prikazna u vrsti br. 7 služi da se u skup resursa AR.js aplikacije doda fajl koji odgovara glTF 3D modelu.

Tabela 4.10 Šabloni koda studije slučaja korisničkih interfejsa proširene stvarnosti

Br	Element	Opis	Šablon koda

1	Marker - Barkod	Ubacivanje barkod markera u aplikaciju i pridruženog objekta	`<a-marker id='marker{Marker.hasNumber.Number}' type='barcode' value='{Marker.hasNumber.Number}' cursor='rayOrigin: mouse'>`
			`{Object.hasCodeTemplate.CodeTemplate}`
			`</a-marker>`

			Ograničenja:
			`Marker.hasObject.Object`
			`AdaptabilityRule.targets.Marker`
			`AdaptabilityRule.hasCondition.CoverMaker`

2	Object - Cube	Ubacivanje 3D kocke u aplikaciju i dodeljene akcije koja se odigrava prilikom interakcije sa njom	`<a-box id='cube-{Object}'`
			`position='{Object.hasInitPosition.InitPositio'`
			`material='opacity:{Object.hasOpacity.Opacity};`
			`color:{Object.hasColor.Color};'`
			`{Action.targets.Object}>`
			`</a-box>`

3	Robot Go Backward	Komanda za kretanje robota unazad	AFRAME.registerComponent('roscommhandler', {
			tick: async function () {
			
			if (document.querySelector('#{RobotGoBackward.targets.Marker}').object3D.visible == true && zahtev==0) {
			
			zahtev=1;
			const http=new XMLHttpRequest();
			const url='{RobotGoBackward.targets.ExternalServer.hasUrl.Url}';
			sleep(1000).then(() => {
			http.open("GET",url);
			http.send();
			zahtev=0;
			})

4	Loop Sound	Reprodukcija zvučnog fajla sa ponavaljanjem	AFRAME.registerComponent('soundhandler', {tick: function () {
			var {Loopsound.targets.Marker} = document.querySelector('#{LoopSound.targets.Marker}');
			if (document.querySelector('#{LoopSound.targets.Marker}').object3D.visible == true) {
			{LoopSound.plays.Marker}.play();
			} else {
			{LoopSound.plays.Marker}.pause();
			}

5	Audio	Ubacivanje zvučnog fajla u AR.js aplikaciju	<a-assets> <audio id="{Audio}" src="{Audio.hasFilePath.FilePath}" preload="auto"></audio>
6	Marker – GPS lokacija	Dodavanje markera koji odgovara GPS lokaciji	<a-entity clickhandler{GpsLocation} look-at="[gps-camera]" animation-mixer="loop: repeat" gltf-model="#{GpsLocation.hasObject.Object}" scale="{Object.hasScale.Scale}" rotation="{Object.hasRoration.Rotation}" gps-entity-place="{GpsLocation.hasCoordinates.Coordiantes}" {Object.hasAnimation.Animation} animation-mixer {AdaptabilityRule.hasCondtion.ClickObject} clickable ></a-entity>
			Marker.hasObject.Object AdaptabilityRule.targets.Marker AdaptabilityRule.hasCondition.ClickObject
7	GltfModel	Ubacivanje a-assets stvake koja odgovara	<a-assets> <a-asset-item id="{GltfModel}"

		3D modelu u glTF format.	src="{GltfModel.hasFilePath.Filepath}" ></a-asset-item>

4.3.8 SPARQL upiti za verifikaciju sistema i proveru uslova adaptacije

U prikazanoj studiji slučaja uzimaju se u obzir sledeći aspekti verifikacije:

1) *dodeljenost objekata markeri*ma – pre generisanja koda, mora biti ispunjeno da je svakom od ubačenih markera dodeljen tačno jedan objekat;

2) *iskorišćenost resursa* – svaki od ubačenih 3D modela i zvukova u aplikaciji mora biti iskorišćen, i

3) *prisustvo ROS gospodara za scenario upravljanja robotima* – ukoliko u aplikaciji kao akcije postoji bilo koja od ROS komandi, neophodno je da se definiše server koji izvršava komponentu ROS gospodara, koja omogućava izdavanje odgovarajućih komandi.

U **Tabeli 4.11**, prikazani su odgovarajući SPARQL upiti namenjeni verifikaciji korisničkih interfejsa proširene stvarnosti.

Tabela 4.11 SPARQL upiti za verifikaciju instanci sistema i proveru ispunjenosti uslova pravila adaptacije za studiju slučaja naprednih korisničkih interfejsa proširene stvarnosti

Aspekat	Upit

Dodeljenost objekata markerima	PREFIX aruio: <http://www.example.com/aruio/>
	SELECT DISTINCT ?m
	WHERE {
	GRAPH <http://www.example.com/aruio> {
	?m aruio:hasObject ?o.
	FILTER(regex(STR(?m), "barkod1"))
	}
	}

Iskorišćenost resursa	PREFIX aruio: <http://www.example.com/aruio/>
	SELECT DISTINCT ?o
	WHERE {
	GRAPH <http://www.example.com/aruio> {
	?m aruio:hasObject ?o.
	FILTER(regex(STR(?o), "object1"))
	}
	}

	PREFIX aruio: <http://www.example.com/aruio/>
	SELECT DISTINCT ?a
	WHERE {
	GRAPH <http://www.example.com/aruio> {
	?ar aruio:targets ?a.
	FILTER(regex(STR(?a), "audio1"))
	}
	}

Prisustvo ROS gospodara	PREFIX aruio: <http://www.example.com/aruio/>
	SELECT DISTINCT ?ros
	WHERE {
	GRAPH <http://www.example.com/aruio> {
	?ar aruio:targets ?ros.

```
                    ?ar  aruio:hasAction rc.
                    ?ros rdf:type ExternalServer.
                    ?rc  rdf:type ControlRobot.
                }
}
```

4.4 Podacima-vođena prilagodljiva arhitektura električne mreže pametnih gradova

Ova studija slučaja oslanja se na prethodne radove autora iz oblasti pametnih mreža [138] [139][140], primene blokčejna i linearne optimizacije sa ciljem ostvarivanja efikasne trgovine električnom energijom [139][141], primene blokčejn tehnologija u sinergiji sa semantičkim tehnologijama sa ciljem automatskog generisanja pametnih ugovora [139] [142] kao i njihove verifikacije [6]., kao i primene nadgledanog mašinskog učenja na predikcije potrošnje i detekciju anomalija u energetici [140][143].

4.4.1 Uvod

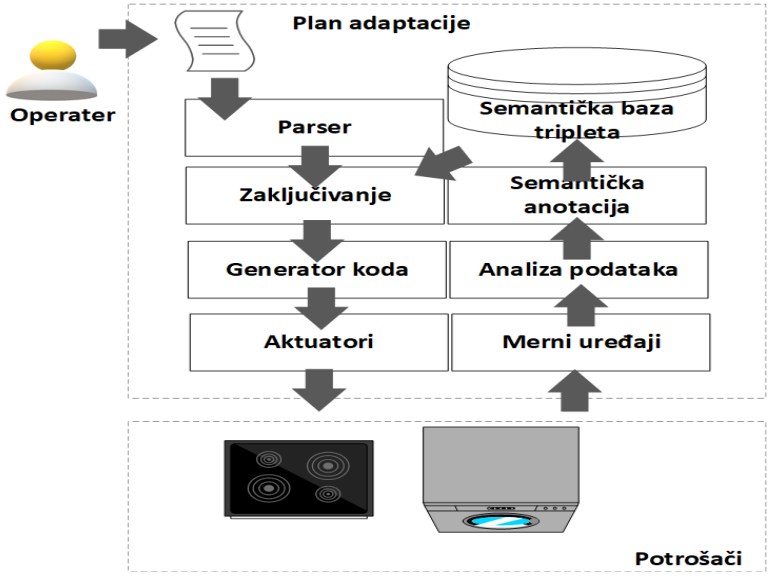
Pametni gradovi definišu se kao čvrsta sprega elektronskih uređaja, digitalnih i informacionih tehnologija u okviru domaćinstava, radnom okruženju, ali i na javnim mestima sa ciljem omogućavanja održivog razvoja zajednice [144]. Međutim, ovakav vid transformacije donosi dosta izazova u pogledu razvoja infrastrukture, projektovanja građevinskih objekata, saobraćaja, rukovanja otpadom, zakonske regulacije, zagađenja životne sredine, a pogotovu u pogledu distirbucije elektirčne energije . Ova studija slučaja se fokusira na energetske aspekte u okviru pametnih gradova.

Imajući u vidu da zahtevi za električnom energijom u pametnim gradovima neprestano rastu, dok su, sa druge strane dostupni izvori ograničeni ili neobnovljivi, upravljanje energijom u pametnim gradovima spada u jedan od najvećih izazova. Veliki pritisak na električnu mrežu za ispunjenje energetskih zahteva domaćinstava pametnih gradova predstavlja zapravo jedno od uskih grla što se tiče njihovog razvoja i širenja [145]

Prema tome, u skorije vreme se dosta truda ulaže u transformaciju postojećih sistema za distribuciju energije težeći ka takozvanim pametnim mrežama, koje se u velikoj meri oslanjaju na infromacione i komunikacione tehnologije. Pametna mreža predstavlja dvosmerni sajber-fizički sistem, koji se oslanja na informacije prikupljene i obrađene od strane računarske inteligencije sa ciljem obezbeđivanja čiste, bezbedne, pouzdane, efikasne, ekonomične i održive električne energije krajnjim korsinicima [145][146]. Iz tog razloga, primena tehnologija

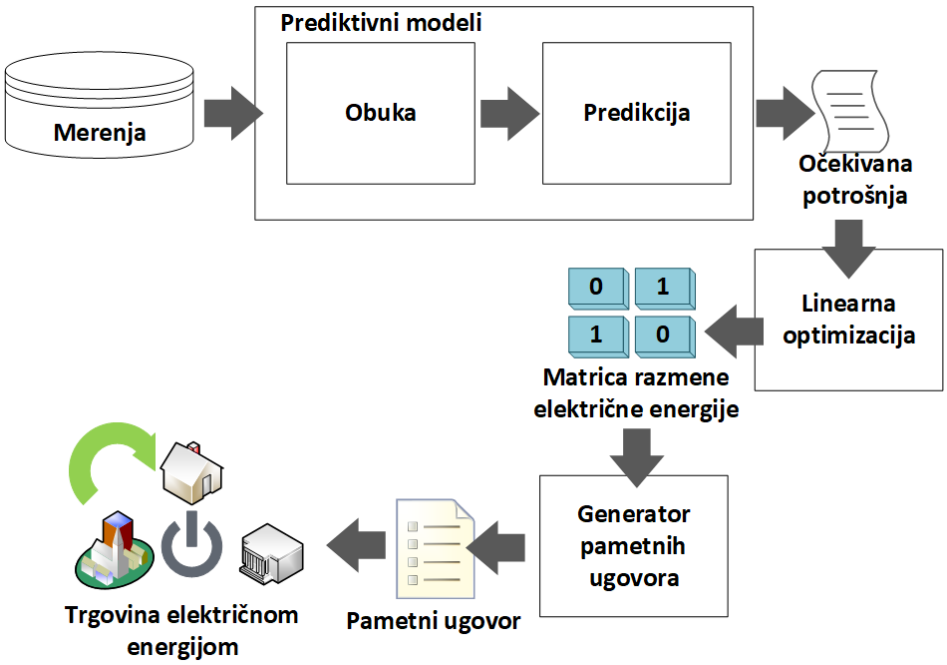
pametnih električnih mreža je od krucijalnog značaja za postizanje energetske efikasnosti u pametnim gradovima [147].

Prikazana studija slučaja prikazuje radni okvir za energetski efikasne pametne gradove, iskorišćavajući prikupljene podatke o korisnicima, njihovim potrošačkim navikama i okruženju. Za merenje vrednosti električnih veličina zaduženi su pametni uređaji za merenje. Za ovu svrhu se najčešće koriste single-board računari (poput Raspberry Pi), mikrokontroleri ili pametni telefoni. Nakon toga, prikupljeni podaci se analiziraju korišćenjem tehnika mašinskog učenja. Po pitanju analize podataka, klasifikacija se najčešće koristi za detekciju anomalija, a regresija za predviđanje opterećenja (zahteva za električnom energijom od strane domaćinstva). Ishodi analize podataka se dalje semantički anotiraju u skladu sa predloženim ontološkim radnim okvirom, tako da se mehanizmima semantičkog zaključivana i izvršavanjem SPARQL upita može doći do informacija o nastalim promenama u izvršnom okruženju. Operateri definišu akcije koje se izvršavaju u slučaju ispunjenja ciljanih uslova u okruženju, što zajedno čini plan adaptacije. Definicija plana adaptacije se realizuje upotrebom vizuelnog alata za modelovanje koji se oslanja na intuitivnu notaciju, a koji je generisan na osnovu domenske ontologije. Konačno, na osnovu zadatog plana se vrši generisanje komandi za specifične aktuatorske uređaje (poput uređaja za relejnu zaštitu zasnovanih na Arduinu), sa ciljem da se odgovori na promene nastale u okolini. Kao primer se može navesti gašenje potrošačkog uređaja upotrebom Arduino relejnog sklopa u slučaju detektovanih anomalija. Na **Slici 4.19** je data ilustracija principa rada predloženog semantički-vođenog radnog okvira prilagodljivih pametnih mreža.



Slika 4.19 Predložena arhitektura prilagodljivih pametnih mreža [138]

Osim toga, obuhvaćen je i model optimalne razmene električne energije oslanjajući se na blokčejn tehnologije i pametne ugovore za tu svrhu. Što se generisanja pametnih ugovora i njihove verifikacije tiče, takođe se koriste semantičke tehnologije. Jedan od mogućih pristupa za postizanje povećane energetske efikasnosti u pametnim mrežama jeste trgovina električnom energijom. Na ovaj način, potrošačima je omogućeno učešće u bilateralnoj, decentralizovanoj razmeni električne energije sa svrhom zadovoljenja potreba svojeg domaćinstva, pri čemu se električna mreža ne opterećuje dodatno. Princip rada mehanizama za trgovinu električnom energijom, korišćen u ovoj disertaciji, je dat na **Slici 4.20**.



Slika 4.20 Mehanizam proaktivne trgovine električnom energijom oslanjajući se na predikciju potrošnje, linarnu optimizaciju i blokčejn [141]

Prediktivni model vrši predikciju potrošnje na osnovu prethodnihh podataka o potrošnji koji dolaze sa pametnih mernih uređaja u okviru domaćinstva. Dalje, predviđena potrošnja električne energije se prosleđuje modulu za linearnu optimizaciju, koji ima za cilj da reši problem razmene električne energije između domaćinstava uz najmanje troškove, tako da zahtevi za energijom svakog od njih budu ispunjeni. Kao rezultat ovog procesa, formira se matrica razmene električne energije, koja sadrži informaciju o tome kolika se količina energija razmenjuje između svakog para domaćinstava. Konačno, u poslednjem koraku, ova matrica se koristi za generisanje pametnih ugovora namenjenih izvršenju transakcija oslanjajući se na blokčejn.

Veliki potencijal podacima-vođenog upravljanja električnom energijom je pokazan u mnogobrojnim radovima do sada [148][149], pri čemu je fokus uglavnom na smanjenju troškova, povećanju efikasnosti mreže ili oba. Međutim, prikazana studija slučaja, sa druge strane teži da smanji kompleksnost upravljanjem heterogenim uređajima i infrastrukturom na koju se pametna mreža oslanja, što obično nije slučaj u postojećim rešenjima.

4.4.2 Primena IntisOnt semantičkog radnog okvira za pametne mreže

U pogledu statičkih aspekata, i u ovom slučaj je SAIO ontologija polazna tačka, a izvodimo sledeće ključne komponente: 1) *Grid* – element najvišeg nivoa, predstavlja pametnu mrežu u celini, koja se sastoji od čvorova 2) *Prosumer* – domaćinstvo ili industrijsko postrojenje, predstavlja jedan čvor pametne mreže, učesnika u razmeni električne energije, koji troši, ali i proizvodi električnu energiju 3) *SmartDevice* - pametni uređaj koji se koristi unutar domaćinstva. Dalje, pametni uređaji mogu biti jedan od ova dva tipa: a) *aktuatori* – poput Arduino uređaja za relejnu zaštitu b) *merni uređaji* – služe da mere električne veličine i vrednosti unutar domaćinstva, na osnovu kojih se donose dalje odluke o upravljanju u okviru pametne mreže.

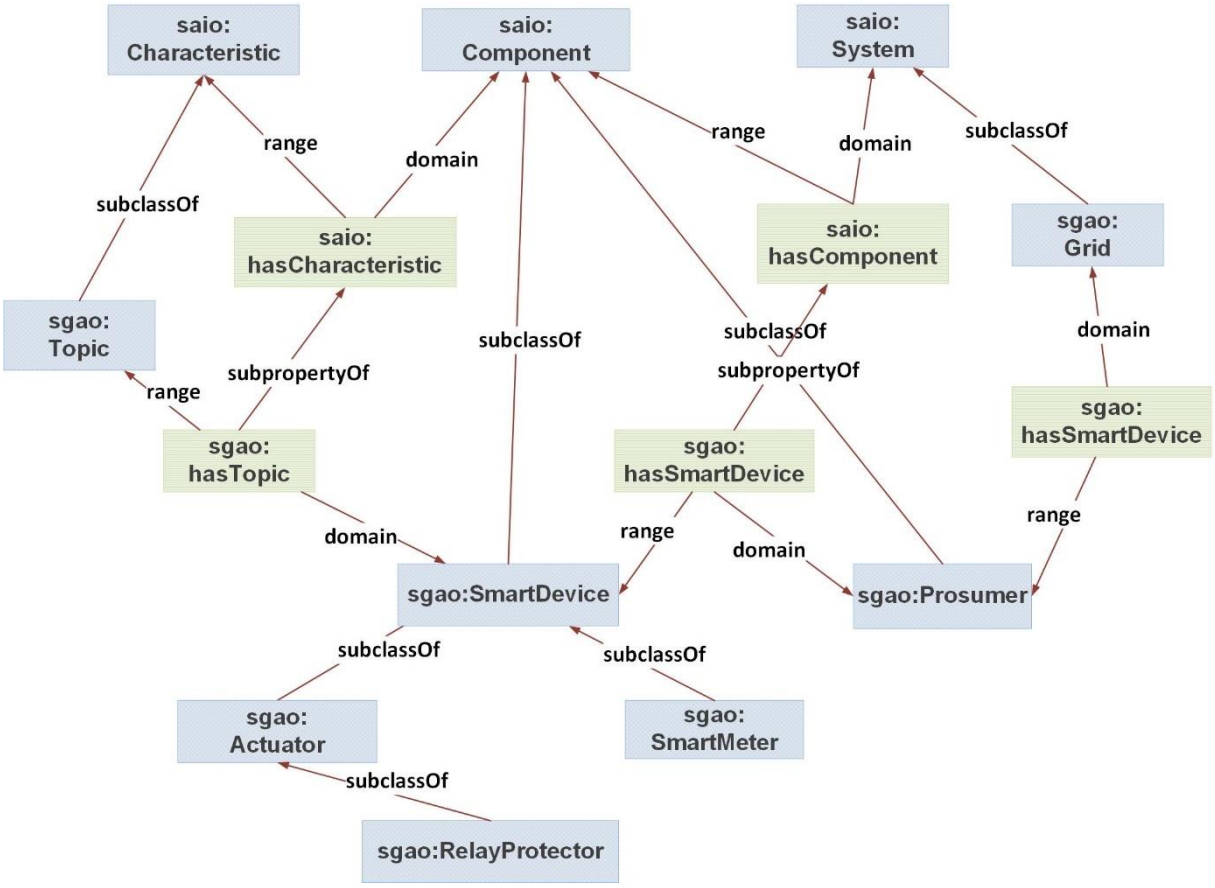
Po svojoj prirodi, oni zasnovani su najčešće zasnovani na Arduino i Raspberry Pi platformama, a sa njima se na daljinu komunicira razmenom Message Queuing Telemetry Transport (MQTT)[21] poruka za datu temu *(topic)* u ovom slučaju, kao što je opisano u [138]. MQTT predstavlja publish-subscribe ISO standardizovan protokol za razmenu poruka, skrmonih zahteva, koji radi povrh TCP/IP protokol i JSON-kodirane stringove kao format poruka. Razmena poruka kod MQTT se obavlja posredstvom brokera, koji predstavlja server zadužen za dostavljanje poruka odgovarajućih tema na koje su se pretplatnici (*subscribers*) prethodno prijavili. Pošiljaoci poruka (*publishers*) izdaju poruku na datu temu, a broker je zadužen da ih dostavi svim pretplatnicima za tu temu. Primer MQTT poruke koja se šalje pametnom aktuatorskom uređaju sa ciljem gašenja potrošača kod kog je detektovana anomalija na topic *RelayProtection* u JSON formatu *{"command":"TurnOff"}*.

Dalje, u ovom kontesktu, domaćinstvo predstavlja jednog proizvođača-potrošača (engl. *prosumer*) u pametnoj električnoj mreži, odnosno učesnika koji osim što troši električnu energiju za zadovoljavanje svojih potreba, između ostalog i proizvodi električnu energiju. Od energetskih aspekata, uzimaju se u obzir ukupna količina proizvede energije, ali, sa druge strane i zahtevi za potrošnjom u domaćinstvu. Što se proizvedene električne energije tiče, prosumer

[21] https://mqtt.org/

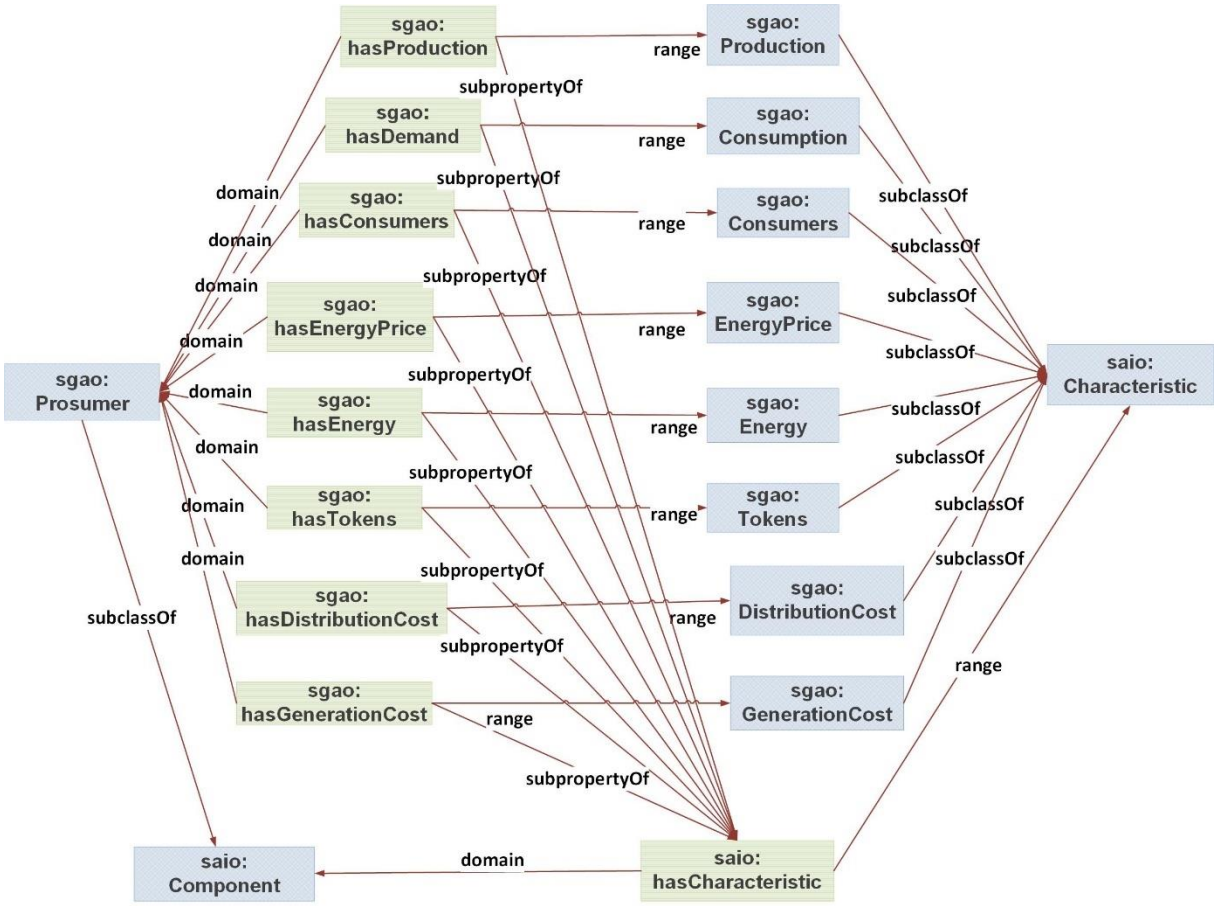
može trgovati viškom električne energije i prodavati je drugim učesnicima koji nisu u mogućnosti da proizvedu dovoljno energije za svoje potrebe. Trgovina se obavlja posredstvom blokčejna, pri čemu svaki učesnik može ponuditi jediničnu cenu po MW. Osim toga, prilikom trgovine elektirčnom energijom se i troškovi distirbucije energije uzimaju u obzir prilikom obračunavanja ukupne cene. Između ostalog, mogu se uzeti u obzir i karakteristike samog građevisnkog objekta, što oslikava njegove energetske potrebe u pogledu hlađenja i grejanja. Među karakteristikama ovog tipa, mogu se uzeti u obzir visina, odnos zapremine i površine, površina poda, površina zidova, staklene površine i orijentacija. Dalje, s obzirom da se trgovina obavlja oslanjanjem na blokčejn, svakom od domaćinstava odgovara po jedan nalog na odabranoj blokčjen platformi (u ovom slučaju Ethereum). U tom kontekstu, za svako od domaćinstava se razmatra i adresa unutar blokčejn platforme koja služ za njihovu identifikacuju. Konačno, svakom od ovih naloga odgovara neka količina blokčejn tokena, uz pomoć kojih se mogu obaviti neophodne transakcije za uspšenu razmenu električne energije u pametnoj električnoj mreži.

Slika 4.21a ilustruje prethodno opisanu primenu SAIO ontologije za svrhu reprezentacije statičkih aspekata sistema pametnih mreža.



Slika 4.21a Statički aspekti domena prilagodlivih pametnih mreža izraženi uz pomoć SAIO ontologije – SGAO ontologija

Dalje, daje detaljniji priaz relevantnih svojstava elemenata je dat na **Slici 4.21b**.



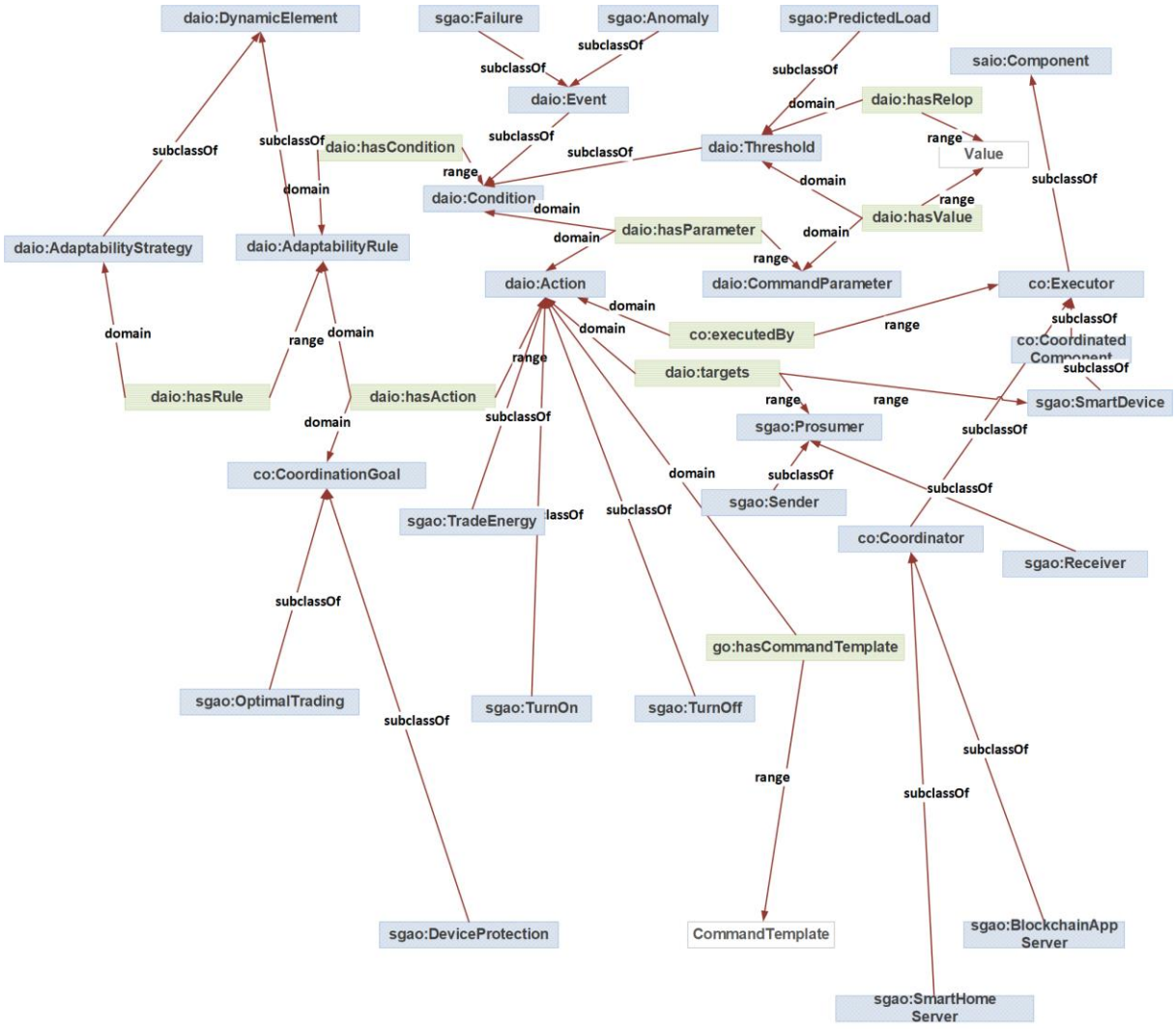
Slika 4.21b Detaljniji pregled svojstava elemenata SGAO ontologije

Sa druge strane, na **Slici 4.22** je prikazana primena DAIO i CO ontologija sa ciljem izražavanja dinamičkih aspekata, koji omogućavaju adaptivno ponašanje pametnih mreža. Kao što se na toj slici može videti, uslovi aktivacije mehanizama adaptacije mogu biti događaji kao što su detektovane smetnje u mreži (*Anomaly*) ili otkazivanje uređaja (*Failure*), ali i relacioni uslovi, kao što su granična vrednost predviđene potražnje za električnom energijom (*PredictedLoad*). Do vrednosti relevantnih za navedene uslove se dolazi primenom tehnika nadgledanog mašinskog učenja nad skupom podataka istorijskih vrednosti – regresija za predikciju potrošnje, a klasifikacija za smetnje i otkazivanje uređaja. Za realizaciju svakog od ovih uslova, poziva se odgovarajući servis koji se oslanja na prediktivni model. Detalji o primeni odgovarajućih tehnika mašinskog učenja i modela implementiranih pomoću Weka biblioteke u Javi se mogu naći u dodatku na kraju ove sekcije.

Dalje, što se tiče akcija koje se mogu preduzeti u ovim slučajevima, tu su paljenje (*TurnOn*) ili gašenje (*TurnOff*) potrošačkih uređaja po potrebi, oslanjajući se na za relejnu zaštitu, ali i

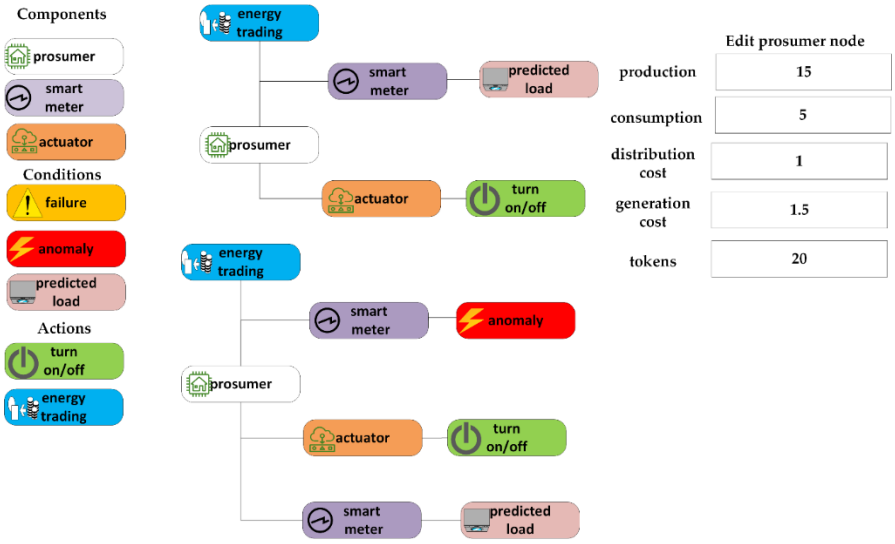
trgovina električnom energijom (*TradeEnergy*). Što se prva dva slučaja tiče, njihova implementacija se svodi na slanje poruka na odgovarajuće MQTT teme za IoT uređaje, kao što su Arduino ili Raspberry Pi. Na ovaj način se može i realizovati strategija potrošnje električne energije, tako što se potrošači gase ukoliko predikcija potrošnje prevazilazi raspoloživu količinu električne energije. Po prijemu poruke na temu, ovi uređaji izvršavaju odgovarajuće komande, zavisno od toga o kojoj se temi radi. Pretpostavka je da se na svakom od ovih uređaja povezan relej i pokrenuta skripta za kontrolu potrošačkih uređaja, kao što je prikazano u radovima [139][140].

Zadnji slučaj – trgovina električnom energijom, najčešće se realizuje kada je ili veća količina električne energije od potrebne proizvedena, pa prosumer onda teži da proda taj višak, ali i u suprotnom slučaju, kada proizvedenom količinom ne mogu da budu zadovoljene trenutne potrebe domaćinstva. U ovim situacijama se primenjuju mehanizmi za automatsko generisanje Solidity pamentih ugovora namenjenih Ethereum blokčejn platformi. Parametrizacija pametnih ugovora se vrši na osnovu ishoda procesa lienarne optimizacije za trgovinu električnom energijom uz minimalne troškove, kao što je dato u dodatku. Što se prosumera uključenih u transakciju tiče, jedan od njih ima ulogu pošiljaoca (*Sender*), a drugi primaoca (*Receiver*).



Slika 4.22 Dinamički i aspekti koordinacije prilagodljivih pametnih mreža izraženi uz pomoć DAIO i CO ontologija

Dalje, izgled Node-RED okruženja za kreiranje instanci modela koji omogućavaju primenu mehanizama pametnih mreža za trgovinu električnom energijom u domaćinstvu posredstvom blokčejna, dat je na **Slici 4.23**. Dati primer prikazuje Node-RED tok u kojem imamo dva prosumer-a (domaćinstva). Dalje, za svako od njih imamo po jedan pametni merni uređaj i aktuator, a oba učestvuju u scenario trgovine električnom energijom posredstvom blokčejna.



Slika 4.23 Node-RED okruženje za modelovanje statičkih i dinamičkih aspekata sistema prilagodljive pametne mreže zasnovane na IoT uređajima, koji se za trgovinu električnom energijom oslanja na blokčejn i pametne ugovore

4.4.3 Korišćeni šabloni generisanja koda

U **Tabeli 4.12** se može videti pregled najbitnijih isečaka koda koji su korišćeni za automatsko generisanje prethodno opisanih scenarija upotrebe u sluačaju pametnih mreža zasnovanih na IoT uređajima.

Što se pametnih mernih IoT uređaja tiče, za svrhu inicijalizacije, server pametnog doma publikuje komandu za iniciranje merenja na jedinstveni MQTT topic koji odgovara tom mernom uređaju (vrsta br. 1). Slično, po pitanju akcija aktuatora, izdaju se odgovarajuće komande na njegov MQTT topic, tako da jedna od njih odgovara paljenju potrošačkog uređaja (vrsta br. 2), a druga gašenju (vrsta br. 3). Po pitanju trgovine električnom energijom između prosumera, parametrizuje se Solidity pametni ugovor (vrsta br. 4) namenjen Ethereum blokčejn platformi, tako da čvor koji prodaje energiju predstavlja primaoca, a onaj koji kupuje pošiljaoca. Osim toga, u ovom pametnom ugovoru se radi i obračun cene, tako što se uzima u obzir količina energije koja se razmenjuje, ali i cena generisanja na strani primaoca, zajedno sa troškovima prenosa od primaoca do pošiljaoca. U okviru totalne cene, jedan deo čini drugi proizvod cene generisanja po jedinici sa prodatom količinom električne energije, a drugi jeste cena prenosa od pošiljaoca do primaoca, što se dodaje prethodnom proizvodu. Konačno, obračunata cena izražena u ETH tokenima se skida sa računa pošiljaoca, a dodaju računu primaoca. Kompletan kod ovog pametnog ugovora dat je kao primer u poglavlju o blokčejn tehnologijama.

Tabela 4.12 Šabloni koda studije slučaja korisničkih interfejsa proširene stvarnosti

Br	Element	Opis	Šablon koda	Izvršava
1	SmartMeter	Inicijalizacija pametnog mernog uređaja	`client.publish({Actuator.hasTopic.Topic}, "StartSensing");`	SmartHome Server
2	TurnOn	Paljenje potrošačkog uređaja korišćenjem Arduina za relejnu zaštitu	`client.publish({Actuator.hasTopic.Topic}, "TurnOn");`	SmartHome Server
3	TurnOff	Gašenje potrošačkog uređaja korišćenjem Arduina za relejnu zaštitu	`client.publish({Actuator.hasTopic.Topic}, "TurnOff");`	SmartHome Server

4	EnergyTrading	Definicija blokčejn transakcije koja se obavlja između dva prosumera	`pragma solidity ^0.4.21;`
			`contract TradeEnergy{`
			`event Sent(address {Sender}, address {Receiver}, uint {TradeEnergy.hasAmount.Amount}, uint {Receiver.hasDistributionCost.DistributionCost}, uint {Receiver.hasGenerationCost.GenerationCost});`
			`uint total;`
			`uint token_price;`
			`function trade() public {`
			`total=({TradeEnergy.hasAmount.Amount} * {Receiver.hasGenerationCost.GenerationCost}+ Receiver.hasDistributionCost.DistributionCost})/token_price;`
			`if (balances[{Sender}] < total) return;`
			`balances[{Sender}] -= total;`
			`balances[{Receiver}] += total;`
			`emit Sent({Sender}, {Receiver}, {TradeEnergy.hasAmount.Amount}, {Receiver.hasDistributionCost.DistributionCost}, {Receiver.hasGenerationCost.GenerationCost}););`
			`}`
			`}`
				Blockchain AppServer

4.4.4 SPARQL upiti za verifikaciju sistema i proveru uslova adaptacije

Sa ciljem provere ispunjenosti uslova adaptacije, ali i korektnosti transakcija, za prethodno opisanu studiju slučaja prilagodljivih pametnih mreža, uočavamo nekoliko aspekata koji su od značaja za verifikaciju, a **Tabela 4.13** daje pregled odgovarajućih SPARQL upita;
: 1) dovoljna količine energije na raspolaganju za obavljanje transakcije (preduslov transakcije) – primalac u transakciji (onaj koji prodaje električnu energiju) treba da ima dovoljnu količinu energije da bi mogao da izvrši transakciju (vrsta br. 1); 2) minimalna količina preostale energije za zadovoljenje potreba nakon obaljene trgovine (postuslov transakcije)– za svakog od razmatranih prosumer-a, kao krajnji ishod nakon obavljenih transakcija jeste to da je za svakog od njih ukupna preostala energija dovoljna da ispuni potrebe potrošačkih uređaja, što znači da ne bi trebalo da postoje oni koji imaju potražnju veću od trenutno dostupne energije (prikazano u vrsti br. 2); 3) uslov preopterećenja – predviđena potrošnja je veća od trenutno raspoložive energije, što dovodi do sprovođenja štednje energije (gašenje potrošača) ili kupovine električne energije od drugih prosumer-a (vrsta br. 3); 4) detektovanje anomalija– došlo je do predikcije anomalije na nekom od povezanih uređaja u domaćinstvu, što je uslov aktivacije mehanizma relejne zaštite od strane aktuatora (vrsta br. 4).

Tabela 4.13 SPARQL upiti za verifikaciju transakcija razmene električne energije između prosumera u prilagodljivoj pametnoj mreži

Br.	Aspekat	Upit

1	Dovoljna količina enrgije na raspolaganju (preduslov transakcije)	`PREFIX sgaoo: http://www.example.com/sgao/`
		`SELECT DISTINCT ?r`
		`  WHERE {`
		`  GRAPH <http://www.example.com/sgao> {`
		`  ?et sgao:hasAmount ?a.`
		`  ?et sgao:hasReceiver ?r.`
		`  ?r sgao:hasEnergy ?pre.`
		`  FILTER(?pre >= ?a)`
		`  }`
		`}`

2	Minimalna količina preostale energije (postuslov transakcije)	`PREFIX sgao: http://www.example.com/sgao/`
		`SELECT DISTINCT ?prosumer`
		`  WHERE {`
		`  GRAPH <http://www.example.com/sgao> {`
		`  ?prosumer sgao:hasDemand ?demand.`
		`  ?prosumer sgao:hasEnergy ?current.`
		`  FILTER(?current >= ?demand)`
		`  }`
		`}`

3	Uslov preopterećenja prosumer-a	`PREFIX sgao: http://www.example.com/sgao/`
		`SELECT DISTINCT ?prosumer`
		`  WHERE {`
		`  GRAPH <http://www.example.com/sgao> {`
		`  ?ar dao:Targets ?prosumer.`
		`  ?prosumer sgao:hasEnergy ?current.`

		?ar dao:hasCondition ?oc. ?oc dao:hasPredictedLoad ?predicted. FILTER(?predicted >= ?current) } }

4	Detekcija anomalija na potrošaču	PREFIX sgao: http://www.example.com/sgao/
		SELECT DISTINCT ?smeter
		WHERE {
		GRAPH <http://www.example.com/sgao> {
		?ar dao:Targets ?smeter.
		?ar dao:hasCondition ?ac.
		?ac rdf:type ?sgao:Anomaly.
		}
		}

4.4.4.1 Dodaci

4.4.4.1.1 Optimalna razmena električne energije u pametnoj električnoj mreži

Pretpostavimo da imamo mrežu međusobno povezanih domaćinstava, pri čemu se svako od njih označava kao čvor. Sa druge strane, veza između njih se označava kao poteg. Svaki $čvor_i$ ima kao karakteristiku maksimalni kapacitet električne energije koji može proizvesti, označen kao $kapacitet_i$, dok je, sa druge strane, potražnja za električnom energijom od strane potrošača tog domaćinstva označena kao $potražnja_i$, a oba su data u kilovatima na čas (kWh). U slučajevima kada je potražnja od strane potrošača veća od dostupne električne energije u nekom domaćinstvu, onda se pokreće mehanizam trgovine i razmene električne energije. Za svaki od potega, dodeljujemo promenljivu odlučivanja $razmena_{ij}$, koja zapravo predstavlja količinu energije koja će biti razmenjena između čvorova i i j, što je dato na sledeći način:

$$razmena\ [i,j] \geq 0, ako\ se\ odvija\ razmena\ između\ čvor_i\ i\ čvor_j$$
$$razmena[i,j] = 0, u\ suprotnom \tag{4.9}$$

Pri tome, za svaki od potega, cena prenosa električne energije između čvorova i i j se označava sa pc_{ij}. Sa druge strane, cena generisanja za svaki $čvor_i$ se označava kao gc_i. Glavno ograničenje po pitanju razmene energije između čvorova jeste u tome da je ukupna energija čvora jednaka ili veća od potražnje za taj konkretan čvor. Pri tome, uzimamo u obzir sumu kapaciteta proizvedene električne energije i primljene energije od strane drugih čvorova u mreži, dok se sa negativnim znakom ubraja prodata energija drugim čvorovima. Ovo ograničenje može biti izraženo na sledeći način:

$$kapacitet[i] + \sum j \in Čvorovi\ razmena[i,j] -$$
$$razmena[j,i]\ potražnja[i], gde\ je\ i\ \in\ Čvorovi \tag{4.10}$$

Konačno, što se tiče funkcije cilja, jedna od mogućnosti jeste minimizacija suma troškova, uzimajući u obzir generisanje, ali i razmenu električne energije između čvorova, sa ciljem da se optimizuje distribucija:

$$minimize \sum i, j \in \text{Čvorovi } razmena[i,j]pc[i,j]gc[i], gde\ je\ i \in \text{Čvorovi} \quad (4.11)$$

Odgovarajući kod AMPL implementacije može se naći u radu [122].

4.4.4.1.2 Predikcija potrošnje električne energije u domaćinstvu

Predikcija potreba za elelktričnom energijom domaćinstava za različite sezone u toku godine je od izuzetne vežnosti što se tiče postizanja energetske efikasnosti distribucije u pametnim električnim mrežama. Ovaj prediktivni model, zasnovan na [97], koristi se u četvrtoj studiji slučaja, a njegov cilj je da omogući predviđanje koliko energije troši neko domaćinstvo tokom zime za potrebe grejanja, a leti za potrebe rashlade, zavisno od karakteristika same prostorije. Prediktivni problem se u ovom slučaju razmatra kao regresija, s obzirom da je ciljana promenljiva za predikciju kontinualna numerička vrednost. Za obuku ovog prediktivnog modela, upotrebljen je javno dostupan skup podataka [151], koji poseduje podatke za 768 domaćinstava, a njegova strukutra je ilustrovana u **Tabeli 4.14**.

Tabela 4.14 Korišćeni skup podataka za predikciju potrošnje električne energije domaćinstava

Promenljiva	Opis
Relativna kompaktnost	Odnos zapremine i površine prostorije
Veličina stana	Površina data u m^2
Veličina zidova	
Veličina krova	
Zastakljene površine	
Visina kuće	Koliko je kuća visoka, dato u metrima
Orijentacija	Strana sveta ka kojoj je građevina orijentisana
Distribucija staklenih površina	Koliko staklenih površina ima građevina
Opterećenje grejanja	Količina energije koju domaćinstvo zimi troši na grejanje
Opterećenje hlađenja	Količina energije koju domaćinstvo leti troši na hlađenje

Što se tiče predikcija, korišćena je linearna regresija iz Weka biblioteke, a ostvareni su sledeći rezultati: srednja relativna greška (MRE) od 22.86% i srednjekvadratna (RMSE) greška 29.282%. Za obuku metoda bilo je potrebno 1.087 s za ovaj skup podataka, dok za pojedinačnu predikciju vrednosti ne prelazi 0.1 s. U poređenju sa pristupom koji se oslanja na duboke neuronske mreže [139], vreme pojedinačne predikcije je u ovom slučaju je više od 5 puta brže.

4.4.4.1.3 Predikcija stabilnosti električne mreže

Stabilnost je od krucijalnog značaja za pametne mreže i infrastrukutru u pozadini. Veće varijacije vrednosti parametara mogu dovesti do pogrešnih odluka, što u slučaju elektirčnih mreža može dovesti do fatalnih posledica. Prema tome, još jedan od krucijalnih aspekata pametnih mreža jeste predikcija anomalija i nestabilnosti, a zatim i proaktivno preduzimanje odgovarajućih akcija, pre nego što dođe do ozbiljnih posledica. U ovom radu, predikcija stailnosti električne mreže se tretira kao problem binarne klasifikacije.

Za realizaciju ovog prediktivnog modela oslanjamo se na javno dostupni skup podataka [152] koji sadrži 60 000 uzoraka izvezenih iz simulatora pametne mreže. Odgovarajući matematički model zasnovan na sistemu diferencijalnih jednčanina za mrežu od 4 čvora prikazan je u radu [143]. Ovakva arhitektura pametnih mreža ima pretpostavku da je jedan od njih proizvođač, koji snabdeva energijom ostala 3 potrošačka čvora. Opis razmatranih ulaznih i izlaznih promenljivih je dat u **Tabeli 4.15**.

Tabela 4.15 Korišćeni skup podataka za predikciju potrošnje električne energije domaćinstava

Promenljiva	Tip	Opis
τ_1-τ_4	Realan broj [0.5, 10]	Vreme reakcije za svaki od čvorova-učesnika u razmeni električne energije. Prva vrednost (τ_1) se odnosi na proizvođača električne energije, dok ostale vrednosti predstavljaju potrošače.
p_1-p_4	Realan broj [-2.0, 0.5]	Nominalna snaga koja se tiče potrošnje (negativna vrednost) ili proizvodnje (pozitivna) od strane čvorova učesnika. Ovdes se p1 odnosi na proizvođača, dok su ostalo potrošači a važi sledeća relacija između ovih vrednosti: *p1 = - (p2 + p3 + p4)*
γ_1- γ_4	Realan broj [0.05, 1.00]	Koeficijenti elastičnosti cene odgovarajućih prosumer čvorova.
stab	Realan broj	Maksimalni realni deo korena karakteristične diferencijalne jednačine
stabf	Kategorička vrednost	Ako je (stab>0) onda: stabf=1 U suprotnom: stabf=0

Rezime ostvarenih rezultata za različite Weka metode klasifikacije, dat je u **Tabeli 4.17**. Kao što se može videti, najbolje performanse predikcije pokazuje J48, sa prilično brzim vremenom obuke u odnosu na ostale upoređivane metode. Sa druge strane, *DecisionStump* je bio ubedljivo najbrži po pitanju vremena izvršenja, ali ne može da parira ostalim metodama po pitanju performansi. Što se tiče pojedinačnog poziva klasifikacije J48 kao metode sa najboljim performansama predikcije tiče, predloženi pristup u ovoj disertaciji ima više od 3 puta kraće vreme izvršenja nego kada se oslanjamo na duboke neuronske mreže, kao što je prikazano u [139]. Međutim, tačnost predikcija je oko 5% niža u poređenju sa dubokim neuronskim mrežama.

Tabela 4.17 Rezultati predikcije stabilnosti električne mreže upotrebom Weka biblioteke u Javi

Algoritam	Vreme obuke [s]	Pojedinačna predikcija vrednosti [s]	Accuracy [%]
IBk	169.71	0.13	86.69
J48	28.50	0.09	91.07
DecisionTable	98.43	0.11	76.08
DecisionStump	3.22	0.07	68.33

5 EKSPERIMENTI I EVALUACIJA

U ovom poglavlju se razmatra evaluacija predloženog semantički-vođenog radnog okvira, na osnovu rezultata ostvarenih za prikazane studije slučaja i njihove varijante. Razmatrani su kvantitativni i kvalitativni aspekti evaluacije, uzimajući u obzir poređenje sa alternativnim metodama i postojećim sličnim rešenjima, vođeno smernicama iz rada [11].

Što se kvantitativnih rezultata u prikazanim eksperimentima tiče, razmatrani su sledeći aspekti od značaja:

1) vreme izvršenja (u sekundama) - koliko traje neki određeni korak prilikom automatskog izvršenja od strane algoritma, poput generisanja koda, izvršenja SPARQL upita, izvršenja procedure linearne optimizacije, obuke modela mašinskog učenja ili pojedinačne predikcije upotrebom utreniranog modela;

2) kvalitet predikcija (u procentima) – kvalitet dobijenih predikcija od strnae modela mašinskog učenja, izražen kao tačnost za klasifikaciju, a srednja relativna ili srednjekvadratna greška za regresiju;

3) trajanje manuelnih procedura (u minutima i sekundama) – koliko neki od razmatranih koraka ili zadataka traje kada se izvršava ručno, od strane korisnika;

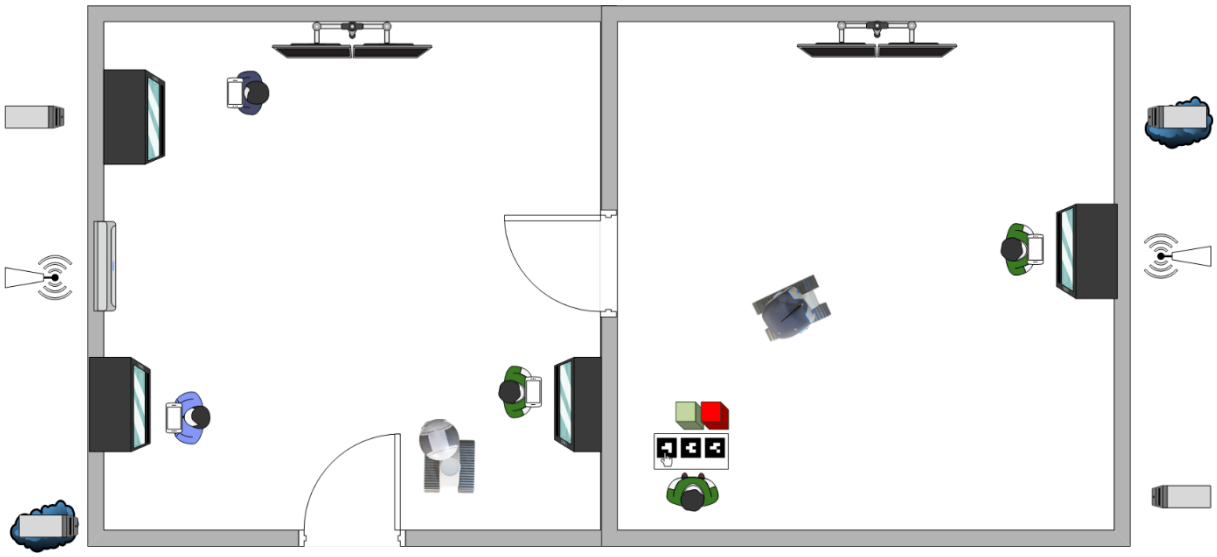
4) benefiti (u procentima) – koliko je poboljšanje perforamnsi u slučaju korišćenog plana adaptacije, i

5) ubrzanje – odnos vremena, koji izražava koliko puta se brže izvršavaju automatske procedure od manuelnih koraka, sa ciljem izražavanja efektivnosti pristupa.

Što se tiče vremena izvršenja, kao referentno izvršno okruženje, korišćen je Acer Nitro 5 laptop pod Windows 11 operativnim sistemom, sa sledećim karakteristikama: procesor - Intel i5-10300H, 2.5 GHz, quad-core; RAM meorija – 24 GB DDR4; skladištenje podataka – 1TB M1 SSD. Merenja vremena izvršenja pojedinih koraka su data kao prosečna vrednost za 10 pokretanja. Za simulaciju blokčejn transakcija, koristi se Remix[22] IDE namenjen izvršenju

[22] https://remix.ethereum.org/

Solidity pametnih ugovora za Ethereum blokčejn platformu.Evalucija je sprovedena kombinovanjem elemenata prethodno prikazanih studija slučaja, proširenjem postavke eksperimenta iz prethodnog rada autora sa SCI liste [9]. Ilustracija objedinjenog scenarija data je na **Slici 5.1.**



Slika 5.1 Ilustracija objedinjenog scenarija evaluacije: računarstvo u magli, koordinacija robota, proširena stvarnosti i verifikacija blokčejn pametnih ugovora

Postavka je sledeća – razmatramo izložbeni zatvoreni prostor sa eksponatima. Mobilni roboti se kreću unutar prostorije sa ciljem nadzora i to da detektuju osobe koje ne nose masku, ali i nedozvoljeno prisustvo osoba u prostorijama gde je ograničen pristup za vreme redovnih izložbi. U scenariju imamo 3 robota – prvi detektuje osobe bez maske, a nakon detekcije, drugi robot dolazi sa kutijom u kojoj su maske do te prostorije, kao slučaj koordinacije. Treći robot detektuje prisustvo osoba i broji ih. Osim automatskog patroliranja, robotima je, po potrebi, moguće upravljati ručno upotrebom aplikacije proširene stvarnosti sa markerima, kao što je opisano u trećoj studiji slučaja. Na sličan način, pored svakog od eksponata postoji barkod marker na koji, kada se usmeri kamera pametnog telefona, prikazuje odgovarajući 3D model. Klikom na 3D model od strane korisnika, dolazi na preusmeravanje do web stranice koja sadrži informacije o ovom objektu, ali i mogućnost rotacije tog modela i zumiranje dodirom od strane korisnika.

Što se servisa u pozadini tiče, imamo ROS gospodara, zajedno sa servisima računrskog vida za detekciju maske na osobama i brojanje osoba. Tu su i web aplikcije proširene stvarnosti – za kontroler robota, ali i informacije o eksponatima. Ukoliko dođe do sumnjivog saobraćaja (detekcija metodom mašinskog učenja iz dodatka), broj njegovih replika se povećava na 2, što

je primer adaptivnog ponašanja u slučaju anomalija. Sa druge strane, ukoliko je broj detektovanih posetilaca u prostoriji veći od 10, dolazi do kreiranja još jedne replike kontejnera web servera sa aplikacijom proširene stvarnosti namenjene posetiocima. Dalje, ako je više od 5 robota aktivno, dodaje se još jedna replika ROS gospodara i u ovom slučaju. ROS gospodar je raspoređen na lokalnom serveru, takođe i servisi računarskom vida, s obzirom na osetljivost prikupljenih podataka (slike sa osobama), tako da ne izlaze izvan granica organizacije u okviru koje se koriste. Osim toga, još jedan razlog zašto je ROS gospodar na lokalnom serveru - s obzirom da bi njegova javna izloženost mogla da dovede u pitanje privatnost prikupljenih senzorskih podataka, ali i mogućnost napadačima da preuzmu kontrolu nad robotima, što može imati katastrofalne posledice. Međutim, sa druge strane, imamo da se predikcija anomalija, broja korisnika i potrošnje električne energije odvijaju na serverima u oblaku. Što se dostupnih servera tiče, na raspolaganju je njih četiri – od toga dva lokalna (fizički unutar organizacije), a preostala dva u oblaku.

Po pitanju primene elemenata softverski-definisanih mreža, ukoliko je detektovan pad broja obrađenih slika sa kamere u sekundi od strane robota po pitanju detekcije maski i brojanja osoba, aktivira se mehanizam za smanjenje mrežnog kašnjenja, oslanjajući se na uređaje sa podrškom za softverki-definisane mrežne funkcije. Kao odgovor na pad performansi, može se primeniti prioritizacija saobraćaja koji dolazi sa robota (slike za analizu). Konačno, u slučaju otkaza mrežnih uređaja ili dodavanja novih, može se primeniti metoda za optimalno rutiranje primenom linearne optimizacije, kao što je opisano prethodno u dodatku. Dalje, izlaz optimizacije jeste matrica alokacije, koja se dalje primenjuje za generisanje SDN pravila oblikovanja saobraćaja u mrežama zasnovanim na OpenFlow[23] protokolu.

Dodatno, što se aspekata primene pametnih mreža tiče, u prostoriji imamo pet razmatranih potrošača – četiri kutije sa eksponatima, koje obuhvataju osvetljenje, senzore za dim i alarm; jedan klima uređaj za regulaciju temperature. Osim toga, imamo dva pametna merna instrumenta – jedan za eksponate, a drugi za rashladni uređaj, pri čemu su prisutna i dva uređaja za relejnu zaštitu, koji služe da ugase potrošače ukoliko dođe do anomalija po pitanju snabdevanja električnom energijom. Konačno, sa ciljem evaluacije scenarija trgovine električnom energijom dodajemo još 7 prosumer-a, sa kojima muzej može razmenjivati električnu energiju.

Tabela 5.1 prikazuje rezime ostvarenih kvantitativnih rezultata, što se različitih komponenti i aspekata studije slučaja tiče. Prva kolona predstavlja naziv razmatrane studije

[23] https://info.support.huawei.com/info-finder/encyclopedia/en/OpenFlow.html

slučaja. U drugoj koloni se nalazi opis postavke eksperimenta za tu studiju slučaja u pogledu pokrivenih elemenata, pravila verifikacije, veličine skupa podataka za predikcije i ostalih specifičnosti. Dalje, treća kolona označava naziv konkretnog koraka ili komponente u okviru studije slučaja koji se razmatra za evaluaciju. Sa druge strane, četvrta kolona prikazuje primenjivana pravila adaptacije u razmtranoj studiji slučaja. Peta kolona specificira aspekat evaluacije koji se za tu komponentu ili korak razmatra, zajedno sa jedinicom mere kojom se izmerena vrednost izražava. U nastavku, šesta kolona daje konkretnu izmerenu vrednost za prethodno razmatrani aspekat od interesa. Konačno, poslednja kolona daje izmereno vreme za manuelno kreiranje ekvivalentnog sistema bez upotrebe prikazanog pristupa i pomoćnih alata. Ova vrednost je data u minutima, sa pretpostavkom da osoba ima već prethodno iskustvo u radu sa korišćenim tehnologijama (računarstvo u magli – Docker, Kuberetes; koordinacija robota – ROS; proširena stvarnost – AR.js; pametne mreže – MQTT, IoT uređaji; blokčejn – Solidity pametni ugovori i Ethereum).

Tabela 5.1 Eksperimenti i evaluacija objedinjene studije slučaja – kvantitativni rezultati

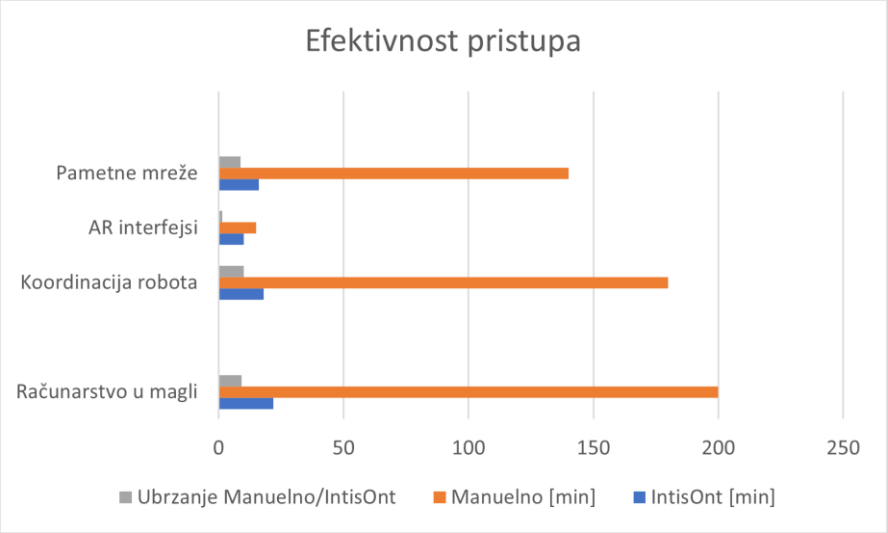
Studija slučaja	Opis postavke	Korak	Pravila adaptacije	Aspekat [mera]	Merenje	Manuel. proc. [min]
Računarstvo u magli	6 servera (2 edge, 2 cloud) 4 mrežna uređaja 6 servisa: -ROS gospodar, -detektovanje maski, -brojanje osoba -AR aplikacija za eksponate -AR aplikacija za upravljanje robota	Kreiranje domenske ontologije (delimično manuelno)	1) Ako je detktovan sumnjiv saobraćaj onda: replicirati ROS gospodara 2) Ako broj posetilaca>10 onda: skalirati AR aplikaciju 3) Ako je broj aktivnih robota>5 onda: skalirati ROS gospodara 4) Ako je fps detekcije osoba<10 onda: SDN prioritizacija i oblikovanje saobraćaja	Vreme izvršenja [s]	1100	230
		Generisanje GUI			2.13	
		Kreiranje instance (manuelno)			200	
		Raspoređivanje kontejnera – generisanje koda			2.21	
		Verifikacija – SPARQL upiti			1.15	
		Linearna optimizacija - optimalan raspored			0.25	
		Linearna optimizacija - minimizacija kašnjenja			0.23	
		Adaptacija			0.16	
		Predikcije – detekcija malicioznog saobraćaja na serveru klasifikacijom (IBk)		Tačnost [%]	99.22	
				Vreme obuke [s]	1.81	
Koordinacija robota	3 ROS robota	Kreiranje domenske ontologije (delimično manuelno)	1) Ako je detektovana osoba bez maske u nekoj prostoriji onda: poslati robota sa	Vreme izvršenja [s]	900	180
		Generisanje GUI			2.04	
		Kreiranje instance			150	

		Kreiranje izvršnog okruženja – generisanje koda	stalkom u tu prostoriju 2) Ako je broj osoba u prostoriji>5 onda: zatvoriti vrata		1.99	
Proširena stvarnost	1 marker po eksponatu 3 markera za kontrolu robota	Kreiranje domenske ontologije (delimično manuelno)	1) Ako je detektovan marker onda: prikaži 3D objekat 2) Ako je zaklonjen marker sa prikazanim 3D objektom onda: izvrši ROS komandu 3) Ako korisnik klikne na 3D objekat onda: prikaži odgovarajuću stranicu sa informacijama		500	20
		Generisanje GUI		Vreme izvršenja [s]	1.83	
		Kreiranje instanci			120	
		Verifikacija – SPARQL upiti			2.19	
		Informacije o eksponatima – generisanje aplikacije			1.16	
		Kontroler robota – generisanje aplikacije			1.74	
Pametne mreže	2 pametna merna uređaja 2 aktuatora	Kreiranje domenske ontologije (delimično manuelno)	1) Ako je detektovana anomalija na potrošaču onda: ugasiti potrošača relejnim sklopom 2) Ako je PredikcijaPotrošnje > RaspoloživaEn onda: obaviti trgovinu energije		720	120
		Generisanje GUI		Vreme izvršenja [s]	1.51	
		Generisanje izvršnog okruženja			1.78	
		Kreiranje instance (manuelno)			250	
		Verifikacija – 3 pravila			0.92	
		Predikcija potrošnje – Linearna regresija		MRE [%]	22.86	
				RMSE [%]	29.28	
				Vreme obuke [s]	1.087	
		Predikcija stabilnosti - klasifikacija upotrebom J48		Tačnost [%]	91.08	
				Vreme obuke [s]	28.50	
Pametni ugovori	8 prosumer-a učestvuje u razmeni	Linearna optimizacija – trgovina energijom		Vreme izvršenja [s]	0.144	20
		Generisanje ugovora			3.21	
		Verifikacija pametnih ugovora – 1 pravilo			0.74	

Sa druge strane, sa ciljem ilustracije efektivnosti prikazanog pristupa, dat je uporedni prikaz vremena za manuelno kreiranje ekvivalentnih sistema naspravm automatizovanog uz prikazane alate. Ovde je sumarno vreme za predloženi pristup dato u minutima i upoređeno sa vremenom potrebom u slučaju manuelne procedure. Osim toga, odnos vremena manuelnog pristupa naspram predloženog je takođe razmatran kao ubrzanje koje se postiže korišćenjem predloženog pristupa, što se može videti u **Tabeli 5.2**. Dalje, **Slika 5.2** prikazuje ostvarenu uštedu vremena i ubrzanje korišćenjem predloženog pristupa. Kao što se može videti, postignute ubrzanje varira zavisno od studije slučaja, pri čemu ide i do preko 10 puta. Najizraženiji benefiti su kod studija slučaja koordinacije robota i računarstva u magli. To se može objasniti kompleksnošću infrastrukture koja čini izvršno okruženje (ROS i Kubernetes klasteri heterogenih uređaja), tako da automatsko generisanje koda značajno smanjuje potreban trud, ali i čini krivu učenja u ovim oblastima manje strmom. Slična je i situacija sa studijom slučaja pametnih mreža koja sadrži i blokčejn elemente. Sa druge strane, i kod aplikacija proširene stvarnosti postoji uočljivo ubrzanje, ali je manje izraženo, s obzirom na jednostavnost samog AR.js radnog okvira koji se zasniva na jednostavnim HTML elementima uz kraće isečke JavaScript koda.

Tabela 5.2 Pregled ostvarenog ubrzanja u poređenju sa ekvivalentnim postupcima ukoliko se ne koristi predloženi pristup

Studija slučaja	IntisOnt [min]	Manuelno [min]	Ubrzanje Manuelno/IntisOnt
Računarstvo u magli	22	230	10.45
Koordinacija robota	18	180	10
AR interfejsi	10	15	1.5
Pametne mreže	16	140	8.75



Slika 5.2 Efektivnost pristupa u prikazanim studijama slučaja

U nastvku su razmatrani kvalitativni aspekti evaluacije predloženog rešenja. Za svaku od studija slučaja, u odgovarajućim tabelama je dat pregled ostvarenih benefita u odnosu na druga slična rešenja, uzimajući u obzir relevantne aspekte iz odgovarajućih domena.

Prvo, dato je upoređivanje prikazane studija slučaja računarstva u magli sa relevantnim sličnim rešenjima, što se može videti u okviru **Tabele 5.3** . Simbol „X“ u odgovarajućoj koloni označava da je taj aspekat pokriven rešenjem. Što se tiče razmatranih aspekata imamo sledeće: 1) cloud – podrška za izvršenje na infrastrukturi u oblaku 2) edge – izvršenje po ivici mreže 3) verifikacija modela – da li poseduje mehanize kojima se potvrđuje da je model sistema ispravan 4) automatizovano raspoređivanje – da li rešenje ima podršku za automatsko generisanje koda i raspoređivanje servisa na osnovu tog koda 5) run-time analiza – da li postoji podrška za uzimanje u obzir aspekata tokom izvršenja, poput QoS i anomalija 6) adaptivnost – da li to rešenje obuhvata aspekte adaptacije infrastrukture kao odgovor na promene u izvršnom okruženju (poput oporavka od napada ili otkaza sistema). Može se videti da, iako većina njih ima podršku za automatizovanje i raspoređivanje koda, samo jedno od razmatranih rešenja zapravo pokriva aspekte adaptacije i to samo za slučaj repliciranja servisa. U predstavljenoj studiji slučaja iz ove disertacije, pokriveni su svi navedeni aspekti, zajedno sa adaptacijom.

Što se izdvojenih relevantnih rešenja tiče, DICE [155] pruža jezik za modelovanje koji omogućava automatizovanu konfiguraciju i raspoređivanje aplikacija koje obrađuju veliku količinu podataka u oblaku. Predstavlja jednu od ranijih platformi ovog tipa, ali ne obuhvata podršku za izvršenje po ivici (*edge*), na lokalnim serverima, niti podržava IoT uređaje. Sa druge strane, DITAS [157] popunjava ovu prazninu i donosi podršku za edge i IoT uređaje, ali njegova notacija za modelovanje je prilično pojednostavljena i pruža ograničenu fleksibilnost po pitanju konfiguracije parametara, a i sama verifikacija kreiranog modela nije obuhvaćena. Po pitanju adaptacije servisa, ova dva rešenja ne pružaju nikakve mogućnosti. Međutim, kasniji progres PaaSage [156] projekta uvodi jednostavna pravila skaliranja servisa kao vid adaptacije. Dalje, među izdvojenim relevantnim sličnim rešenjima se nalazi i qCon [158] koji ima podršku za računarstvo u magli (i cloud i edge izvršenje), pri čemu se fokusira na analizu prikupljenih podataka u toku izvršenja, dok izostaje podrška za kompletno automatizovano raspoređivanje. Prema tome, možemo zaključiti da je prikazana studija slučaja računarstva u magli kompletnija od sličnih rešenja po pitanju pokrivenih aspekata, pogotovu što se tiče adaptivnosti, podrške za heterogene edge uređaje, ali i koordinacije, koja nije prisutna kod izdvojenih projekata.

Tabela 5.3 Upoređivanje sa sličnim rešenjima u oblasti računarstva u magli

Naziv	Cloud	Edge	Verifikacija modela	Automatizovano raspoređivanje	Run-time analiza	Adaptivnost
DICE [155]	X	-	X	X	X	-
DITAS [157]	X	X	-	X	X	-
PaaSage [156]	X	-	X	X	X	X
qCon [158]	X	X	-	-	X	-

Dalje, u **Tabeli 5.4** prikazano je upoređivanje sa relevantnim rešenjima u oblasti pametnih mreža. Od relevantnih elemenata se uzimaju u obzir: 1) modelovanje – obeleženo u slučaju da razmatrano rešenje obuhvata jezik za modelovanje pametnih mreža 2) verifikacija modela – da li rešenje obuhvata mehanizme za proveru ispunjenosti ograničenja i uslova instance modela sa ciljem generisanja ispravnog izvršivog koda ili simulacije 3) izvršenje – ako razmatrano rešenje uzima u obzir aspekte relevantne za događaje do kojih može doći u pametnim mrežama 4) Generisanje koda za uređaje – da li se na osnovu definisanog modela može generisati izvršivi kod za uređaje (poput Raspberry Pi) 5) Analiza podataka i predikcije– da li rešenje obuhvata primenu predikcija i tehnika za analizu podataka 6) adaptivnost – da li rešenje obuhvata primenu adaptivnih pravila ili sličnih mehanizama sa ciljem prilagođavanja promenama do kojih dolazi u spoljašnjem okruženju 7) koordinacija – ukoliko omogućava usklađivanje akcija više različitih uređaja u sistemu. Kao što se može videti, u nekima od postojećih rešenja jeste obuhvaćena adaptivnost, ali postoji praznina po pitanju aspekata koordinacije, što je jedna od glavnih prednosti primene predloženog pristupa u domenu pametnih mreža.

Tabela 5.4 Upoređivanje sa sličnim rešenjima u oblasti pametnih mreža

Naziv	Modelovanje	Verifikacija modela	Izvršenje	Generisanje koda za uređaje	Analiza podataka i predikcije	Adaptivnost	Koordinacija
CosiML [159]	X	X	X	-	X	-	-
Smartlog [161]	X	-	X	X	-	X	-
VDM-SL [162]	X	X	X	X	-	-	-

Slično, **Tabela 5.5** daje uporedni prikaz studije slučaja u oblasti robotike domenski-specifičnim jezicima i notacijama u ovoj oblasti. Prvo od sličnih rešenja [163] je zasnovano na jeziku za modelovanje aplikacija virtuelne stvarnosti, pri čemu je primena ove notacije

prikazana na primeru implementacije 3D simulatora eksperimenata sa robotima i to u web pretraživaču. Drugo rešenje [164] razmatra primenu jezika za modelovanje na aspekte adaptacije u robotskom sistemu zasnovanom na skupu dronova. Treće rešenje [165] predstavlja modelima-vođeni pristup zasnovan na vizuelnoj notaciji, koji cilja ROS platformu i obuhvata aspekte međusobne komunikacije robota, što se može iskoristiti za realizaciju scenarija koordinacije.

Tabela 5.5 Upoređivanje sa sličnim rešenjima u oblasti robotike

Naziv	Modelovanje	Verifikacija modela	Izvršenje	Generisanje koda za uređaje	Analiza podataka i predikcije	Adaptivnost	Koordinacija
VRML [163]	X	X	X	-	-	-	-
DRESS-ML [164]	X	X	X	X	-	X	-
EARL [165]	X	-	X	X	-	-	X

Konačno, u **Tabeli 5.6** je dat rezime funkcionalnosti sličnih rešenja kao prikazana studija slučaja iz oblasti proširene stvarnosti. Za ovaj domen se može videti da koordinacija uglavnom nije razmatrana u postojećim rešenjima, za razliku od prikazanog primera u ovoj disertaciji. SSIML/AR predstavlja vizuelnu notaciju i jezik za modelovanje namenjen apstraktnoj specifikaciji AR aplikacija (generalno) i korisničkih interfejsa, obuhvatajući tri aspekta – struktura interfejsa, prezentacija odgovarajućih informacija i integracija korisničkog interfejsa sa drugim sistemskim komponentama. Dalje, Systems Modeling Language (SysML) je notacija za modelovanje AR aplikacija iz dve persepektive: struktura i ponašnje. Multimodal Interaction Framework (MINT) je modelima-vođen radni okvir visokog nivoa namenjen kreiranju AR aplikacija iz perspektive dizajnera, pri čemu je akcenat na upotrebi različtih načina zadavanja komandi i korisničkih unosa.

Tabela 5.6 Upoređivanje sa sličnim rešenjima u oblasti korisničkih inerfejsa zasnovanih na proširenoj stvarnosti

Naziv	Modelovanje	Verifikacija modela	Statički aspekti	Generisanje koda	Ponašanje	Koordinacija
SSIML/ AR [166]	X	X	X	X	-	-
SysML [167]	X	X	X	X	X	-
MINT [168]	X	-	X	X	X	-

Konačno, biće razmotreni preduslovi, pretpostavke i ograničenja realizovanih eksperimenata i to za svaku od prikazanih studija slučaja. Što se tiče realizacije studije slučaja računarstva u magli, pretpostavka je da svi uređaji imaju instaliranu odgvoarajuću varijantu Docker Engine i Kuberentes klijent za pridruživanje klasteru. Osim toga, na serveru koji ima ulogu gospodara je potrebno instalirati i upravljačke komponente Kubernetesa za kreiranje i upravljanje klasterom. U tom slučaju, radi pojednostavljenja studije slučaja, poželjno je pridruživanje svih servera na raspolaganju klasteru koji je kreirao izabrani Kubernetes ili Docker Swarm gospodar. Sa druge strane, u varijanti prve studije slučaja koja se ne oslanja na rešenja za orkestraciju kontejnera (Docker Swarm ili Kubernetes), dodatno je potrebna i dozvola za SSH pristup svim serverskim uređajima od strane mašine na kojoj se izvršavaju IntisOnt komponente za generisanje koda. Konačno, potrebna je i aktivna internet konekcija na svim uređajima zbog upotrebe javnog online Docker Hub registra Docker slika. Dalje, za studiju slučaja koordinacije robota, neophodno je instalirati komponente ROS gospodara na serveru, a svaki od računara koji upravlja odgovarajućim robotom treba da ima odgovarajuću instancu ROS-a. Osim toga, pretpostavka je da su svi roboti i ROS gospodar unutar iste mreže i međusobno vidljivi. Za treću studiju slučaja, po pitanju aplikacije za upravljanje robotima, neophodno je da uređaj koji pokreće AR korisnički interfejs za upravljanje robotima bude ili povezan na internet ili unutar iste mreže kao i ROS gospodar koji upravlja robotima. Konačno, za poslednju studiju slučaja je pretpostavka da IoT uređaji poseduju odgovarajuće dodatne module (kao što je relej), ali i povezani na mrežu, pri čemu je za ovu svrhu Arduino uređajima neophodan WiFi modul ili Ethernet shield dodatak.

6 DISKUSIJA

Što se alternativnih pristupa na temu automatizovanog razvoja domenski-specifičnih sistema tiče, ističu se trenutno aktuelni veliki jezički modeli (Large Language Models - LLMs)[169], a pogotovu među njima ChatGPT[24], razvijen od strane OpenAI, javno dostupan od kraja 2022. Iako su ovi modeli prvobitno namenjeni sumarizaciji teksta ili vođenju konverzacije sa ljudima, usled velike znatiželje korisnika, pokazano je da se lako mogu prilagoditi za najrazličitije scenarije upotrebe, među njima generisanje, analiza, dopuna i otkrivanje grešaka u programskom kodu (engl. *debugging*) [170]. Uglavnom, generativni LLM

[24] https://chat.openai.com/

imaju mogućnost da na osnovu jednostavne, teksutalne specifikacije u formi upita govornog jezika kao odgovor generišu računarski kod različite namene. U tom kontekstu, cilj budućih istraživanja bi bio da se servis koji se oslanja na LLM integriše kao deo predloženog pristupa, sa svrhom da automatizuje i olakša pojedine korake od značaja: 1) automatsko generisanje SPARQ upita predstavljenog pristupa – ideja je da se na osnovu izbora i odluka od strane korisnika kroz grafički interfejs formiraju pitanja namenjena LLM servisu, na osnovu kojih LLM kao odgovor obavlja generisanje potrebnih SPARQL upita, kao što su upiti za verifikaciju i proveru ispunjenosti uslova adaptacije; 2) automatsko kreiranje instanci sistema na osnovu date ontologije – da LLM upotrebom ontologije kao ulaza i pojednostavljene specifikacije krajnjeg korisnika konstruiše RDF graf, i 3) generisanje koda – da se na osnovu RDF grafa generiše ciljani, domenski-specifični kod. Sličan pristup se pokazao kao efektivan u modelima-vođenom softverskom inženjerstvu, na primeru automatizovanog kreiranja i manipulacije instanci Ecore modela [171], ali i dalje primene sinergije LLM i instanci modela za generisanje pametnih ugovora za blokčejn platforme [172].

Iako na prvi pogled deluje da bi pristup zasnovan na LLM mogao da služi kao kompletna alternativa predloženom radnom okviru, veliki broj do sada poznatih nedostataka još uvek onemogućava primenu LLMa u praksi: 1) LLM teže „halucinacijama“ [169][170] - što dovodi u pitanje kvalitet generisanog koda; 2) nemogućnost pouzdane verifikacije – kod semantičkih grafova, baza znanja je čvrsto utemeljena na definicijama datim ontologijama, što omogućava prilično jednostavan mehanizam verifikacije uz pomoć SPARQL upita nad semantičkom bazom znanja, ali i pravila formalne logike primenjenih nad ovako predstavljenim činjenicama; 3) količina podataka neophodna za obučavanje – LLM zahtevaju ogromnu količinu tekstualnih podataka, što zahteva dodatno vreme i pripremu, dok se u predloženom pristupu dovoljne domenski-specifične definicije uz pomoć ontologija; 4) proširljivost i prilagodljivost – predloženi pristup zasnovan na ontologijama i šablonima koda se brže može prilagoditi za rešavanje nekog novog domenski-specifičnog problema (hipotetički, čak i „u hodu“), što kod LLM zahteva specijalizaciju modela i obuku nad velikom količinom podataka, ali i vrlo moćan računarski hardver za taj proces; 5) vreme izvršenja – metode zasnovane na LLM uglavnom zahtevaju aktivnu internet konekciju, traju dosta duže s obzirom na milijarde parametara po modelu (čak i preko 20 sekundi u nekim slučajevima), pa je njihova upotrena i ovim aspektom ograničena, jer još uvek nisu pogodni za rad u (makar približno) realnom vremenu; 6) visoki hardverski zahtevi – pokretanje servisa zasnovanih na velikim jezičkom modelima zahteva vrlo moćan hardver, najčešće klastere sa velikim brojem servera visokih perforamansi, pri čemu je svaki od njih pojedinačno značajno moćniji od standardnih poslovnih i kućnih računara; 7)

visoka cena – pored skupog i moćnog hardvera, klasteri namenjeni izvršenju servisa koji se oslanjaju na velike jezičke modele troše i ogromne količine električne energije; 8) objašnjivost rezultata – semantički mehanizmi zaključivanja su slični rezonavanju kod ljudi i zasnovani na eksplicitnim reprezentacijama znanja, dok se LLM i slični modeli zasnovani na neuronskim mrežama ponašaju kao crne kutije, tako što da se na osnovu dobijenog izlaza malo toga može pretpostaviti o ulazu koji ga je prouzrokovao.

Sa druge strane, modelima-vođeno softversko inženjerstvo se takođe može koristiti za istu svrhu automatizovanje razvoja domenski-specifičnih sistema i aplikacija, a spada u zrelije pristupe [173]. Iako metamodeli pružaju definiciju koncepata i veza između njih poput ontologija, ontologije i semantičke baze znanja kao glavne prednosti imaju proširljivost i skalabilnost. Semantičke grafovi omogućavaju ugrađivanje novog znanja kroz triplete, pri čemu nema potrebe da se menja algoritam koji služi za generisanje koda na osnovu tog znanja. Na taj način, nove činjenice reprezentovane u obliku tripleta omogućavaju da se promene iz okruženja uzmu u obzir, dok je modelima-vođeni pristup zahtevniji za ovakva proširenja.

Kod modelima-vođenog pristupa, ukoliko dođe do potrebe da se metamodel proširi novim konceptima i vezama, u većini slučajeva je neophodno ponovo generisati i sve dodatne alate (kao primer Ecore iz EMF [29]), kao što su parser i vizuelni alat za modelovanje. Sa druge strane, proširenje ontologija je fleksibilnije, tako da se ažuriranja najčešće integrišu u algoritme za generisanje koda dodavanjem novih upita. Osim toga, upotreba SPARQL upita može značajno pojednostaviti kod neophodan za pribavljanje podataka na osnovu kompleksnih veza između koncepata. Kod modelima-vođenog pristupa u ovom slučaju je potrebno nadograditi sam parser i rukovati referencama objekata sve dok se ne pribavi željena vrednost, što zahteva dodatne napore. Konačno, semantički grafovi imaju kao prednosti ponovnu upotrebljivost i prenosivost, što može olakšati razvoj u slučaju kada imamo više različitih aplikacija koje obrađuju podatke u istom domenu.Prema tome, može se zaključiti da za statičke i topološke aspekte modelima-vođeno softversko inženjerstvo pruža slične mogućnosti, ali po pitanju aspekata prilagođavanja (kao što je scenario adaptacije ili koordinacije) – ontologije i semantičke tehnologije predstavljaju preferirano rešenje zbog svoje proširljivosti i fleksibilnosti [109].

7 ZAKLJUČAK

U ovoj disertaciji je predstavljen semantički-vođen radni okvir zasnovan na ontologijama i algoritmima za automatsko generisanje koda, čiji je cilj olakšavanje projektovanja i implementacije kompleksnih sistema u različitim domenima upotrebe. Efektivnost ovog pristupa je ilustrovana kroz četiri studije slučaja iz različitih oblasti primene: računarstvo u magli, robotika, proširena stvarnost i pametne energetske mreže. Na osnovu dobijenih rezultata, može se tvditi da ovakav pristup značajano skraćuje vreme potrebno za razvoj kompleksnih domenski-specifičnih sistema u odnosu na proces koji se oslanja na manuelne korake. Ubrzanje je najviše izraženo u prvoj studiji slučaja – iznosi čak preko 10 puta, što se može objasniti kompleksnošću infrastrukutre računarstva u magli, čija konfiguracija zahteva ogromne napore, pogotovu ako se koristi orkestracija (kao što je slučaj Kubernetes-a). Sa druge strane, u slučaju aplikacija naprednih korisničkih interfejsa proširene stvarnosti se dobija manje ubrzanje, s obzirom da je sama struktura AR.js aplikacija prilično jednostavna. Međutim, primena predloženog radnog okvira smanjuje strmost krive učenja i doprinosi uštedi vremena potrebnog za upoznavanje sa novim tehnologijama, s obzirom da grafički alat pruža mogućnost intuitivnog kreiranja instance sistema, bez poznavanja specifičnosti implementacije niskog nivoa (Kubernetes, Docker i ROS komande, AR.js elementi, sintaksa i strukutra Solidity pametnih ugovora).

Kvalitativno gledano, upoređivanjem realizovanih studija slučaja korišćenjem predloženog pristupa u odnosu na slična rešenja, možemo zaključiti da predstavljeno rešenje popunjava praznine po pitanju aspekata koordinacije i adaptacije. Dok je adaptacija u nekim postojećim rešenjima ipak zastupljena, koordinacija nije pokreivena u većini slučajeva.

Između ostalog, predloženi pristup pomaže ostvarivanju i drugih unapredjenja koja donose dodatnu vrednost u odgovarajućim domenima: 1) omogućava jednostavnu verifikaciju upotrebom SPARQL upita - s obzirom na reprezentaciju znanja u vidu semantičkog grafa; 2) osim statičkih aspekata, uzima u obzir ponašanje sistema, ali i kooridnaciju elemenata sistema, što nije slučaj u velikom broju postojećih sličnih rešenja; 3) pruža intuitivnu vizuelnu notaciju na osnovu ontologija, koja se dalje koristi za jednostavno kreiranje instanci sistema od strane krajnjih korisnika; 4) ponovna upotrebljivost – za razvoj sistema iz novog domena je dovoljno samo kreirati nove ontologije, dok se i dalje koriste isti algoritmi za generisanje koda; 5) lako prilagodljivo i za nove domene – upotrebom analogija i kompetentnih pitanja, domenski eksperti mogu u kratkom roku kreirati domenski-specifične ontologije na osnovu predloženih

viših ontologija; 6) omogućava „low-code“ razvoj kompleksnih sistema, čak i od strane domenskih eksperata – uptrebom kombinacije intuitivne vizuelne notacije i mehanizama za kreiranje novih ontologija upotrebom analogija, domenskim eksertima je omogućeno i da kreiraju nove ontologije, a zatim i da generišu izvršiv kod, bez znanja o programiranju i algoritmima.

Sa druge strane, kao pomoćna sredstva, u ovoj disertaciji su implementirani model linearnog programiranja, kao i prediktivni model nadgledanog mašinskog učenja u različitim domenima, što predstavlja dodatnu vrednost celokupnog doprinosa ove disertacije. Iako zaključivanje i upiti nad semantičkim grafom pružaju dosta mogućnosti po pitanju izvođenja vrednosti neophodnih pri generisanju koda, zbog same prirode domene i slučajeve korišćenja, u nkeim slučajevima potrebno je primeniti kompleksnije (kao što je optimizacija) i robustnije (mašinsko učenje) matematičke mehanizme da bi se dobile neophodne vrednosti i ostvarili benefiti. Prema tome, u kontekstu ove disertacije, navedene metode služe imaju komplementarnu ulogu i koriste se u sinergiji sa mehanizmima zaključivanja. Dok mašinsko učenje i metode opimizacije daju konkretne numeričke rezultate, semantičke tehnologije daju značenje ovim rezulatima i omogućavaju donošenje odluka na osnovu njih.

Što se linearne optimizacije tiče, koristi se za nalaženje optimalnog rasporeda kontejnera po serverima kod računarstva u magli, ali i za trgovinu električnom energijom između prosumera (oslanjajući se na blokčjen i pametne ugovore) u pametnim električnim mrežama. Na osnovu dobijenih rezultata, može se zaključiti da primena tehnika lienarne optimizacije dovodi do povećanja efikasnosti u tim domenima: 1) računarstvo u magli - postizanje minimalne potrošnje energije ili maksimalne brzine izvršenja, zavisno od izbora korisnika i dinamičko oblikovanje saobraćaja sa ciljem smanjenja kašnjenja u softverski definisanim mrežama; 2) pametne mreže – trgovina električnom energijom uz minimalne troškove i opterećivanje same mreže. Sa druge strane, primenom tehnika mašinskog učenja u ovim domenima omogućena je proaktivna adaptacija odgovarajućih infrastrukutra: 1) računarstvo u magli – predviđanjem anomalija i napada servis se blagovremeno može replicirati, tako da ne dođe do pada kvaliteta korisničke usluge; 2) pametne mreže – predikcijom anomalija na potrošačkim uređajima mogu se sprečiti njihova oštećenja tako što se blagovremeno aktiviraju mehanizmi relejne zaštite; predikcijom potrošnje električne energije se proaktivno može izvršiti snabdevanje dodatnom količinom energije uz minimalne troškove, oslanjajući se na trgovinu upotrebom sinergije linearne optimizacije i blokčejna. Između ostalog, implementirane tehnike nadgledanog mašinskog učenja pokazuju i sličan kvalitet predikcija za odgovarajuće studije slučaja u odnosu na postojeću literaturu: : 1) predikcija anomalija u serverskom logu – 99.22%

tačnosti za kNN metodu 2) predikcija stabilnosti energetske mreže - 91.07% tačnosti za J48 algoritam 3) predviđanje potrošnje električne energije u domaćintvu - 22.86% relativna greška, 29.282% srednjekvadratna greška uz pomoć linearane regresije. Međutim, jedna od glavnih prednosti prikazanih prediktivnih modela jeste značajno kraće vreme izvršenja od alternativnih pristupa kao što su neuronske mreže (bilo da se radi o obuci ili pojedinačnoj predikciji željenog aspekta), što ih, uz prihvatljivu preciznost predikcija čini pogodnim za scenarije koji obuhvataju koordinaciju i dinamičko prilagođavanje sistema sa strogim vremenskim ograničenjima. Za pojedinačnu predikciju vrednosti pokazuje 3 brže vreme izvršenja za klasifikaciju, a više od 5 puta za regresiju u odnosu na neuronske mreže implementirane u [139]. Osim toga, vreme za pojedinačnu predikciju u svim slučajevima je oko jedne desetine sekunde, što uz solidnu tačnost predikcija omogućva uspešnu realizaciju prikazanih scenarija u kojima se ostvaruje proaktivna adaptacija na osnovu ishoda mašinskog učenja.

Konačno, kao rezultati eksperimenata u prikazanim studijama, dobijene su i korisničke aplikacije, koje se dalje mogu koristiti od strane krajnjih korisnika: 1) računarstvo u magli – alat za automatizovano raspoređivanje Docker kontejnera kod hibridnih infrastruktura (cloud/edge, x86/ARM) i to sa podrškom za upravljanje klasterom oslanjajući se na Kubernetes; 2) robotika – radni okvir za koordinaciju Turtlebot robota zasnovanih na ROS sistemu upotrebom jednostavne vizuelne notacije; 3) proširena stvarnost – napredni korisnički interefejsi za upravljanje robotima pokretom ruku, pri čemu se njegova modifikacija može prilagoditi i za muzički scenski nastup; 4) pametne mreže – vizuelna notacija za upravljanje IoT uređajima pametnog doma – pametnim mernim instrumentima i senzorima, ali i aktuatorima.

Sa druge strane, što se dodatnih napora za realizaciju novih studija slučaja tiče, glavna prednost predloženog pristupa je u tome što implementacija novih algoritama za generisanje koda ili njihovo proširenje nisu neophodni, već samo definicija domenskih ontologija od strane eksperata i dodavanje šablona koda odgovarajućim elementima iz domena koji se umeću u konačan rezultat tokom generisanja koda. I u jednoj i u drugoj oblasti se očekuje ubrzani razvoj novih alata koji će u budućnosti još više olakšati ovaj posao.

Po pitanju budućih istraživanja, planirano je razmotriti sinergiju velikih jezičkih modela i semantičkih tehnologija. Iako deluje da pristupi zasnovani na velikim jezičkim modelima nadmašuju sve druge po pitanju generisanja koda, usled nedostataka navedenih u prethodnoj sekciji, njihova upotrebljivost u mnogim poslovnim scenarijima postaje upitna. Po pitanju budžeta, semantički-zasnovana rešenja su značajno ekonomičnija, jer se bez ikakvih poteškoća mogu izvršavati i na konvencionalnim kućnim računarima, dok LLM servisi zahtevaju

infrastrukutru čije intenzivno korišćenje čak i na dnevnoj bazi može prouzrokovati mnogo veće troškove. Osim toga, proširljivost LLM-zasnovanih rešenja je takođe problematična, s obzirom da je za njihovu obuku neophodna velika količina teksutalnih podataka, što, ponekad, zavisno od problema i konkretnog domena, može biti ključni problem usled nedostatka potrebnih podataka. Prema tome, ova sinergija bi se oslanjala na LLM samo po pitanju zadataka sumarizacije teksta (što je jedna od najjačih strana ovih modela) i eventualno drugih izvora podataka, ali ne i na generisanje konačnog koda. Sumarizacija teksta bi bila integrisana sa metodama semantičke anotacije, što bi omogućilo da se značajno olakša kreiranje semantičkih baza znanja polazeći od teksta slobodne forme i date ontologije. Osim toga, primena LLM ima puno potencijala po pitanju učenja ontologija na osnovu velike količine teksta, s obzirom na njihove solidne performanse u izdvajanju ključnih koncepata iz zadatog teksta i njihovih veza. Konačno, po pitanju generisanja izvršivog koda, LLM i dalje ne predstavlja prvi izbor, pogotovu u osetljivim domenima (zdravstvo, automobilska industrija), ali u kombinaciji sa tradicionalnim metodama verifikacije i validacije ipak može dostići prihvatljiv nivo upotrebljivosti.

LITERATURA

[1] R. J. Sternberg, “A Theory of Adaptive Intelligence and Its Relation to General Intelligence”, Journal of Intelligence, 7, 23, pp. 1-17, 2019. https://doi.org/10.3390/jintelligence7040023

[2] R. J. Sternberg, *Adaptive Intelligence: Surviving and Thriving in Times of Uncertainty*, Cambridge University Press, 2021. https://doi.org/10.1017/9781316650554

[3] P. Bateson, “Adaptability and evolution”, Interface Focus 7: 20160126, pp. 1-8, 2017. http://dx.doi.org/10.1098/rsfs.2016.0126

[4] A. E. Williams, “The Architecture of Cognition as a Generalization of Adaptive Problem-Solving in Biological Systems”, Biologically Inspired Cognitive Architectures 2021. BICA 2021. Studies in Computational Intelligence, vol 1032. Springer, Cham, pp 590–595, 2022. https://doi.org/10.1007/978-3-030-96993-6_66

[5] C. T. Yu and C. H. Chen, “Adaptive information system design: one query at a time”, 1985 ACM SIGMOD international conference on Management of data (SIGMOD '85). Association for Computing Machinery, New York, NY, USA, pp. 280–290, 1985. https://doi.org/10.1145/318898.318924

[6] N. Petrović, M. Tošić, "Semantic Approach to Smart Contract Verification", Facta Universitatis Series Automatic Control and Robotics 19(1), pp. 21-37, 2020. https://doi.org/10.22190/FUACR2001021P

[7] A. E. Williams, “Are wicked problems a lack of general collective intelligence?”, AI & Society, 2021. https://doi.org/10.1007/s00146-021-01297-8

[8] V. Nejkovic, N. Petrovic, M. Tosic, N. Milosevic, “Semantic approach to RIoT autonomous robots mission coordination”, Robotics and Autonomous Systems 126:103438, pp.1-19,2020. https://doi.org/10.1016/j.robot.2020.103438

[9] N. Petrovic, M. Tosic, “SMADA-Fog: Semantic model driven approach to deployment and adaptivity in Fog Computing”, Simulation Modelling Practice and Theory, Volume 101, May 2020, 102033, ISSN: 1569-190X, pp. 1-25, 2019. doi: https://doi.org/10.1016/j.simpat.2019.102033

[10] A. E. Williams, N. Petrović, “The Role of Cognitive Computing and Collective Cognitive Computing in the Future Internet”, preprint, pp. 1-14, 2021. https://doi.org/10.31730/osf.io/qb7sm

[11] J. Venable, J. P. Heje, R. Baskerville, “FEDS: a Framework for Evaluation in Design Science Research”, European Journal of Information Systems, 25:1, pp. 77-89, 2016. doi: https://doi.org/10.1057/ejis.2014.36

[12] J. Sander, A. Kuwertz, “Supplementing Machine Learning with Knowledge Models Towards Semantic Explainable AI”, Human Interaction, Emerging Technologies and Future Applications IV, IHIET-AI 2021, Advances in Intelligent Systems and Computing, vol. 1378. Springer, Cham, pp. 3-11, 2021. https://doi.org/10.1007/978-3-030-74009-2_1

[13] N. Petrović, M. Tošić, “Explainable Artificial Intelligence and Reasoning in Smart Cities”, YuInfo 2020, pp. 1-6, 2020.

[14] N. Petrović, “Tackling the COVID-19 Conspiracies: The Data-Driven Approach” 2020 55th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST), pp. 27-30, 2020. https://doi.org/10.1109/ICEST49890.2020.9232760

[15] M. Fowler and R. Parsons, Domain-Specific Languages, Addison-Wesley Professional, 2010.

[16] L. Bettini, Implementing Domain-Specific Languages with Xtext and Xtend, Packt Publishing, 2013.

[17] I. Čeh, M. Črepinšek, T. Kosar, M. Mernik, "Ontology Driven Development of Domain-Specific Languages”, Computer Science and Information Systems vol. 8, no. 2, pp. 317-342, 2011. https://eudml.org/doc/253052

[18] I. Čeh, M. Črepinšek, T. Kosar, M. Mernik, “Using ontology in the development of domain-specific languages”, INForumm 2010, pp. 185-196. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.383.8276&rep=rep1&type=pdf

[19] R. Tairas, M. Mernik, J. Gray, “Using Ontologies in the Domain Analysis of Domain-Specific Languages”, Lecture Notes in Computer Science, pp. 332–342, 2009. https://doi.org/10.1007/978-3-642-01648-6_35

[20] R. Girardi, C. G. de Faira, "An ontology-based technique for the specification of domain and user models in multi-agent domain", CLEI Electronic Journal, Vol. 7 No. 1 (2004): Special Issue of Best Papers presented at JIISIC'2003, Valdivia, Chile, pp. 1-11, 2004. https://doi.org/10.19153/cleiej.7.1.7

[21] K. Kolomvatsos, M. Tsiroukis, S. Hadjiefthymiades, “An experiment description language for supporting mobile IoT applications”, FIRE Book, European Commission, River Publishers, pp. 461-486, 2016.

[22] N. Ferry, F. Chauvel, A. Rossini, B. Morin, A. Solberg, “Managing multi-cloud systems with CloudMF”, Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud '13). Association for Computing Machinery, New York, NY, USA, pp. 38–45, 2013. https://doi.org/10.1145/2513534.2513542

[23] A. Kalnins, L. Lace, E. Kalnina, A. Sostaks, “DSL Based Platform for Business Process Management”, SOFSEM 2014: Theory and Practice of Computer Science. SOFSEM 2014. Lecture Notes in Computer Science, vol 8327. Springer, Cham, pp. 351-362. https://doi.org/10.1007/978-3-319-04298-5_31

[24] J. Jacobs, J. Nicolay, C. De Troyer, W. De Meuter, “PrintTalk: a constraint-based imperative DSL for 3D printing”, Proceedings of the 18th ACM SIGPLAN International Workshop on Domain-Specific Modeling (DSM 2021), Association for Computing Machinery, New York, NY, USA, pp. 11–20, 2021. https://doi.org/10.1145/3486603.3486778

[25] S. Ceri, P. Fraternali, A. Bongio, “Web Modeling Language (WebML): a modeling language for designing Web sites”, Computer Networks, Volume 33, Issues 1–6, pp. 137-157, 2000. https://doi.org/10.1016/S1389-1286(00)00040-2

[26] N. Petrović, M. Radenković, V. Roblek, M. Khokhobaia, I. Gagnidze, “Model-driven Approach to Interactive 3D Web Presentation Generation”, 15th International Online Conference on Applied Electromagnetics - ПЕС 2021, pp. 137-140.

[27] R. Morgan, G. Grossmann, M. Schrefl, M. Stumptner, T. Payne, “VizDSL: A Visual DSL for Interactive Information Visualization”, 30th International Conference on Advanced Information Systems Engineering, pp. 440-455, 2018. https://doi.org/10.1007/978-3-319-91563-0_27

[28] M. Skotnica, R. Pergl, R, “Das Contract - A Visual Domain Specific Language for Modeling Blockchain Smart Contracts”, Advances in Enterprise Engineering XIII. EEWC 2019. Lecture Notes in Business Information Processing, vol 374. Springer, Cham, pp. 149-166, 2020. https://doi.org/10.1007/978-3-030-37933-9_10

[29] Eclipse Modeling Framework (EMF), dostupno na: https://www.eclipse.org/modeling/emf/ , poslednji pristup: 15/03/2022.

[30] Node-RED [online], dostupno na: https://nodered.org/ , poslednji pristup: 13/03/2022.

[31] F. Jelenkovic, M. Tosic, V. Nejkovic, “Semantic driven code generation for networking testbed experimentation”, Enterprise Information Systems, pp. 1–17, 2018. https://doi.org/10.1080/17517575.2018.1509135

[32] T. Gruber, Toward Principles for the Design of Ontologies Used for Knowledge Sharing, International Journal Human-Computer Studies 43 (5-6), pp. 907-928, 1995.

[33] J. Davies, R. Studer and P. Warren, Semantic Web Technologies: Trends and Research in Ontology-based Systems, John Wiley & Sons, 2006.

[34] V. Raghvani, “An introduction to semantic technology and semantic reasoning” [online], dostupn na: https://www.legislate.tech/post/an-introduction-to-semantic-technology-and-semantic-reasoning , poslednji pristup 20/05/2022.

[35] T. S. Perdih, N. Lavrač and B. Škrlj, “Semantic Reasoning from Model-Agnostic Explanations”, 2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI), pp. 105-110, 2021. https://doi.org/10.1109/SAMI50585.2021.9378668

[36] S. Dornelas Costa et al., “A core ontology on the Human-Computer Interaction phenomenon”, Data & Knowledge Engineering vol. 138. 101977, 2022. http://doi.org/10.1016/j.datak.2021.101977

[37] M. Compton et al., The SSN ontology of the W3C semantic sensor network incubator group, Journal of Web Semantics, Volume 17, pp. 25-32, 2012. https://doi.org/10.1016/j.websem.2012.05.003

[38] M. Ashburner et al., "Gene ontology: tool for the unification of biology", The Gene Ontology Consortium. Nat Genet. 2000 May;25(1):25-9. https://doi.org/10.1038/75556

[39] S. El-Sappagh, F. Franda, F. Ali, K. S. Kwak, "SNOMED CT standard ontology based on the ontology for general medical science", BMC Med Informatics and Decision Making, 18(1):76, pp. 1-19, 2018. https://doi.org/10.1186/s12911-018-0651-5

[40] Y. He et al., "CIDO, a community-based ontology for coronavirus disease knowledge and data integration, sharing, and analysis", Sci Data 7, 181, pp. 1-5, 2020. https://doi.org/10.1038/s41597-020-0523-6

[41] A. Ouatiq, K. El Guemmat, K. Mansouri, M. Qbadou, "A design of a multi-agent recommendation system using ontologies and rule-based reasoning: pandemic context", International Journal of Electrical and Computer Engineering (IJECE), Vol.12, No.1, February 2022, pp. 515-523. https://doi.org/10.11591/ijece.v12i1.pp515-523

[42] C. C. Insaurralde, E. P. Blasch, P. C. G. Costa, K. Sampigethaya, "Uncertainty-Driven Ontology for Decision Support System in Air Transport", Electronics 2022, 11, 362, pp. 1-25. https://doi.org/10.3390/electronics11030362

[43] S. Tessier, Y. Wang, "Ontology-based feature mapping and verification between CAD systems", Advanced Engineering Informatics. 27, pp. 76–92, 2013. https://doi.org/10.1016/j.aei.2012.11.008

[44] A. Ławrynowicz et al., "Food Recipe Ingredient Substitution Ontology Design Pattern", Sensors. 22. 1095, pp. 1-14, 2022. https://doi.org/10.3390/s22031095

[45] A. Mulyarto et al., "Ontology-Based Model for Food Transformation Processes - Application to Winemaking", Metadata and Semantics Research (MTSR) 2014, Communications in Computer and Information Science, vol 478, Springer, Cham, pp. 329-343, 2014. https://doi.org/10.1007/978-3-319-13674-5_30

[46] A. Pinto et al., "Towards an ontology for urban tourism", SAC '21, pp. 1887-1890, 2021. https://doi.org/10.1145/3412841.3442142

[47] F. Noardo, "A Spatial Ontology for Architectural Heritage Information", International Conference on Geographical Information Systems Theory, Applications and Management, pp. 143-163, 2017. https://doi.org/10.1007/978-3-319-62618-5_9

[48] M. H. Karray et al., "ROMAIN: Towards a BFO Compliant Reference Ontology for Industrial Maintenance", Applied Ontology, vol. 14, no. 2, pp. 155-177, 2019. https://doi.org/10.3233/AO-190208

[49] H. Ugarte-Rojas, B. Chullo-Llave, BLONDiE: Blockchain Ontology with Dynamic Extensibility", preprint, pp. 1-6, 2020. https://arxiv.org/abs/2008.09518

[50] C. Islam, M. A. Babar and S. Nepal, "An Ontology-Driven Approach to Automating the Process of Integrating Security Software Systems," 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP), pp. 54-63, 2019. https://doi.org/10.1109/ICSSP.2019.00017

[51] X. Wang, N. Guarino, G. Guizzardi, J. Mylopoulos, "Towards an Ontology of Software: a Requirements Engineering Perspective", Ebook: Formal Ontology in Information Systems, Frontiers in Artificial Intelligence and Applications. 267, pp. 317-329, 2014. https://doi.org/10.3233/978-1-61499-438-1-317

[52] H.-J. Happel, S. Seedorf, "Applications of ontologies in software engineering" , ISWC 2006, pp. 1-14, 2006.

[53] N. Mavetera, J. Kroeze, "An ontology-driven software development framework", Business Transformation through Innovation and Knowledge Management: An Academic Perspective, pp. 1713-1724, 2010.

[54] D. Gašević, N. Kaviani, M. Milanović, "Ontologies and Software Engineering", in Handbook on Ontologies, Springer, pp. 593-615, 2009. https://doi.org/10.1007/978-3-540-92673-3_27

[55] M. Bravo, L. Hoyos-Reyes, J. Reyes, "Methodology for ontology design and construction", Contaduría y Administración 64 (4), pp. 1-24, 2019. https://10.22201/fca.24488410e.2020.2368

[56] M. Tosic, I. Seskar, "Resource specification and intelligent user interaction for federated testbeds using Semantic Web technologies", Computer Networks, 63, pp. 84–100, 2014. https://doi.org/10.1016/j.comnet.2014.01.005

[57] Apache, Apache Jena [online], dostupno na: https://jena.apache.org/ , poslednji pristup 26/04/2022.

[58] S. Malik, S. Jain, "Sup_Ont: An Upper Ontology", International Journal of Web-Based Learning and Teaching Technologies 16, pp. 79-99, 2021. https://doi.org/10.4018/IJWLTT.20210501.oa6

[59] V. Mascaradi, V. Cordi, P. Rosso, "A Comparison of Upper Ontologies (Technical Report DISI-TR-06-21)", pp. 1-15, 2014. https://www.researchgate.net/publication/228674060_A_Comparison_of_Upper_Ontologies_Technical_Report_DISI-TR-06-21

[60] R. Arp, B. Smith and A. Spear, *Building Ontologies With Basic Formal Ontology*, MIT Press, 2015.

[61] L. Temal, A. Rosier, O. Dameron and A. Burgun, "Mapping BFO and DOLCE", Studies in Health Technology and Informatics, vol. 160, pp. 1065–1069, 2010.

[62] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, L. Schneider, "Sweetening Ontologies with DOLCE", In: Gómez-Pérez, A., Benjamins, V.R. (eds) Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web. EKAW 2002, Lecture Notes in Computer Science, vol 2473. Springer, Berlin, Heidelberg, pp. 166-181, 2002. https://doi.org/10.1007/3-540-45810-7_18

[63] G. Guizzardi et al., “UFO: Unified Foundational Ontology”, Applied Ontology, vol. 17, no. 1, pp. 167-210, 2022. https://doi.org/10.3233/AO-210256

[64] P. Cassidy, “Toward an Open-Source Foundation Ontology Representing the Longman’s Defining Vocabulary: The COSMO Ontology OWL Version”, Proceedings of the Third International Ontology for the Intelligence Community Conference, pp. 1-4, 2008. http://ceur-ws.org/Vol-440/paper11.pdf

[65] I. Niles and A. Pease, “Towards a standard upper ontology”, Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001 (FOIS '01), Association for Computing Machinery, New York, NY, USA, pp. 2–9, 2001. https://doi.org/10.1145/505168.505170

[66] J. Grabarske and D. Heutelbeck, “An Upper Ontology for the Social Web”, 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, pp. 1128-1131, 2012. https://doi.org/10.1109/asonam.2012.194

[67] M. Maroun, “A Survey on Ontology Operations Techniques”, Mathematical and Software Engineering, Vol. 7, No. 1-2 (2021), pp. 7-28, 2021. https://doi.org/10.5281/zenodo.5716095

[68] N. Choi, I.-Y. Song, and H. Han, “A survey on ontology mapping”, SIGMOD Rec. 35, 3, pp. 34–41, 2006. https://doi.org/10.1145/1168092.1168097

[69] M. A. Abbas and G. Berio, “Creating Ontologies Using Ontology Mappings: Compatible and Incompatible Ontology Mappings” 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), pp. 143-146, 2013. https://doi.org/10.1109/WI-IAT.2013.169

[70] D. Gentner, L. Smith, “Analogical reasoning”, In: V. S. Ramachandran Encyclopedia of Human Behavior (2nd Ed.)., Oxford, UK: Elsevier, pp. 130-136, 2012.

[71] V. Nejkovic and M. Tosic, "Using analogy computing for ontology development," 2016 IEEE 32nd International Conference on Data Engineering Workshops (ICDEW), pp. 115-120, 2016. https://doi.org/10.1109/ICDEW.2016.7495628

[72] J. I. Toledo-Alvarado, A. Guzman-Arenas, G. L. Martinez-Luna, “Automatic Building of an Ontology from a Corpus of Text Documents Using Data Mining Tools”, Journal of Applied Research and Technology Vol. 10 No. 3 June 2012, pp. 398-404, 2012.

[73] A. Maedche and S. Staab, “Ontology Learning for the Semantic Web”, IEEE Intelligent Systems vol. 16 (2), pp. 72-79, 2001.

[74] W. Wilson, W. Liu and M. Bennamoun, “Ontology Learning from Text: A Look Back and into the Future”, ACM Computing Surveys Vol. 44 No. 4 August 2012, Article 20, pp. 1-36, 2012.

[75] C. Manning et al., “The Stanford CoreNLP Natural Language Processing Toolkit”, pp. 1-6, 2014.

[76] N. Petrovic, M. Tosic, “Never-Ending Ontology Learning Approach Applied to Biomolecular Function Prediction”, IcETRAN 2019, pp. 762-767, 2019.

[77] E. Raad, J. Evermann, “The role of analogy in ontology alignment: A study on LISA, Cognitive Systems Research”, vol. 33, pp 1-16, 2015. https://doi.org/10.1016/j.cogsys.2014.09.001

[78] V. Presutti, E. Blomqvist, E. Daga and A. Gangemi, “Pattern-based ontology design”, In Ontology Engineering in a Networked World, pp. 35–64. Springer, 2012.

[79] A. Tahani, B. Parsia and U. Sattler, “Mining Ontologies for Analogy Questions: A Sim-ilarity-based Approach”, Online: http://ceur-ws.org/Vol-849/paper_32.pdf

[80] J. Reynolds, A. Pease, and J. Li, “Analogy and Deduction for Knowledge Discovery”, In IKE, pp. 39-48, 2004.

[81] M. Stojkovic, M. Trifunovic, D Misic, and M. Manic, “Towards analogy-based rea-soning in semantic network”, Computer Science and Information Systems 12(3): pp. 979–1008, 2015.

[82] V. Nejković and N. Petrovic, "Ontology Development Approach Adopting Analogy and Competency Questions", ICIST 2023, pp. 288-297. https://doi.org/ 10.1007/978-3-031-50755-7_27

[83] D. Strmečki, I. Magdalenić, D. Radosević, “A Systematic Literature Review on the Application of Ontologies in Automatic Programming”, International Journal of Software Engineering and Knowledge Engineering, 28(05), pp. 559–591, 2018. https://doi.org/10.1142/s0218194018300014

[84] A. Alokla et al, “Ontological Engineering For Source Code Generation”, Future Computing and Informatics Journal vol. 4, pp. 52-66, 2019. https://doi.org/10.54623/fue.fcij.4.2.1

[85] S. Pileggi, A. A. Lopez-Lorca, G. Beydoun, “Ontologies in Software Engineering”, 29th Australasian Conference on Information Systems (ACIS2018), pp. 1-8, 2018.

[86] W. Hesse, “Ontologies in the Software Engineering Process”, Proceedings of the Workshop on Enterprise Application Integration (EAI 2005), pp. 1-13, 2005.

[87] P. M. Schäfer, “OnToCode: Template-based code-generation from ontologies”, Journal of Open Source Software, 4(40), 1513, pp. 1-2, 2019. https://doi.org/10.21105/joss.01513

[88] C. Steinmetz et al., "Ontology-driven IoT code generation for FIWARE," 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), pp. 38-43, 2017. https://doi.org/10.1109/INDIN.2017.8104743

[89] Y. Gheraibia, A. Bourouis, “Ontology and automatic code generation on modeling and simulation”, 2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT). https://doi.org/10.1109/setit.2012.6481892

[90] A. Braham, M. Khemaja, F. Buendía, F. Gargouri, “User Interface Adaptation through Ontology Models and Code Generation”, In: Ruiz, P.H., Agredo-Delgado, V., Kawamoto, A.L.S. (eds) Human-Computer Interaction. HCI-COLLAB 2021. Communications in Computer and Information Science, vol 1478. Springer, Cham, pp. 225-236. https://doi.org/10.1007/978-3-030-92325-9_17

[91] L. Shan, “Code Generation of Ontology-based abstract Algorithms” (master thesis), University of Dublin, Trinity College, 2008. https://www.scss.tcd.ie/publications/tech-reports/reports.08/TCD-CS-2008-54.pdf

[92] G. B. Dantzig and M. N. Thapa, Linear Programming 1 - Introduction, Springer, 1997.

[93] R. Fourer, D. Gay, B. Kernighan, AMPL: A Modeling Language for Mathematical Programming, Duxbury Press, 2003.

[94] G. B. Dantzig, A. Orden and P. Wolfe, *The generalized simplex method for minimizing a linear form under linear inequality restraints*, Pacific Journal of Mathematics 5(2) (1955) 183-195.

[95] I. Al-Azzoni, J. Blank, N. Petrović, “A Model-Driven Approach for Solving the Software Component Allocation Problem”, MDPI Algorithms, 14(12), pp. 1-19, 2021. https://doi.org/10.3390/a14120354

[96] N. N. Petrović, “Leveraging Linear Programming for Deployment of Container-based Applications within Softwarized Networks in Fog Computing”, 2019 14th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS), pp. 54-57, 2019. https://doi.org/10.1109/TELSIKS46999.2019.9002338

[97] N. Petrović, “Mašinsko učenje u okviru kursa Informacioni sistemi korišćenjem Weka API za Javu”, IEEESTEC – 15th Student Projects Conference, pp. 305-309, 2022.

[98] M. Gates, Blockchain: Ultimate guide to understanding blockchain, bitcoin, cryptocurrencies, smart contracts and the future of money, CreateSpace Independent Publishing Platform, 2017.

[99] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System” [online]. Dostupno na: https://bitcoin.org/bitcoin.pdf , poslednji pristup: 16/05/2022.

[100] V. Buterin, “Ethereum Whitepaper” [online]. Dostupno na: https://ethereum.org/en/whitepaper/, poslednji pristup: 16/05/2022.

[101] U. Bodkhe, “Blockchain for Industry 4.0: A Comprehensive Review”, IEEE Access, vol. 8, pp. 79764 – 79800, 2020. https://doi.org/10.1109/ACCESS.2020.2988579

[102] G. Zheng, L. Gao, L. Huang, J. Guan, Ethereum Smart Contract Development in Solidity, Springer, 2021.

[103] M. Mukhopadhyay, Ethereum Smart Contract Development: Build blockchain-based decentralized applications using Solidity, 1st edition Packt Publishing, 2018.

[104] C. Di Ciccio, G. Meroni, P. Plebani, “On the adoption of blockchain for business process monitoring”, Softw Syst Model 21, pp. 915–937, 2022. https://doi.org/10.1007/s10270-021-00959-x

[105] L. Ricci, D. D. F. Maesa, A. Favenza and E. Ferro, “Blockchains for COVID-19 Contact Tracing and Vaccine Support: A Systematic Review”, IEEE Access, vol. 9, pp. 37936-37950, 2021. https://doi.org/10.1109/ACCESS.2021.3063152

[106] N. Petrović, Đ. Kocić, "Smart technologies for COVID-19 indoor monitoring", Viruses, Bacteria and Fungi in the Built Environment, ISBN: 9780323852067, Woodhead Publishing, pp. 251-272, 2021. https://doi.org/10.1016/B978-0-323-85206-7.00012-5

[107] P. Patel, P. Trikha, "Interpreting Inference Engine for Semantic Web", International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 2, Issue 2, February 2013 pp. 676-678.

[108] N. Petrović, "Model-driven Approach for Deployment of Container-based Applications in Fog Computing", Proceedings of 5th International Conference on Electrical, Electronic and Computing Engineering (IcETRAN 2018), Palics, Serbia, 2018, pp. 1084-1089.

[109] N. Petrović, "Model-based Approach for Semantic-driven Deployment of Containerized Applications to Support Future Internet Services and Architectures", Serbian Journal of Electrical Engineering 16 (1), pp. 21-44, Feb. 2019. https://doi.org/10.2298/SJEE1901021P

[110] N. Petrovic, "Enabling Flexibility of Data-Intensive Applications on Container-Based Systems with Node-RED in Fog Environments", master thesis, Milan, Country: Italy. of Politecnico di Milano, 2017. https://www.politesi.polimi.it/bitstream/10589/135075/1/2017_07_Petrovic.pdf

[111] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero and D. A. Tamburri, "DevOps: Introducing Infrastructure-as-Code," 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSEC), pp. 497-498, 2017. https://doi.org/10.1109/ICSE-C.2017.162

[112] A. A. Laghari, A. K. Jumani, R. A. Laghari, "Review and State of Art of Fog Computing", Arch Computat Methods Eng 28, pp. 3631–3643, 2021. https://doi.org/10.1007/s11831-020-09517-y

[113] Raspberry Pi [online], dostupno na: https://www.raspberrypi.org/, poslednji pristup: 08/09/2023.

[114] Docker [online], dostupno na: https://www.docker.com/, poslednji pristup: 08/09/2023.

[115] Kubernetes [online], dostupno na: https://kubernetes.io/docs/home/, poslednji pristup: 08/09/2023.

[116] N. Petrović: "Machine Learning-Based Run-Time DevSecOps: ChatGPT Against Traditional Approach", IcETRAN 2023, pp. 1-5, 2023. https://doi.org/10.1109/IcETRAN59631.2023.10192161

[117] SCOR [online], dostupno na: http://infosys1.elfak.ni.ac.rs/scor/ , poslednji pristup: 12/09/2023.

[118] V. Nejkovic, N. Petrovic, N. Milosevic, M. Tosic, "The SCOR ontologies framework for robotics testbed", 2018 26th Telecommunication Forum (TELFOR), pp. 1-4, 2018. https://doi.org/10.1109/telfor.2018.8611841

[119] N. Petrovic, V. Nejkovic, N. Milosevic, M. Tosic, "A Semantic Framework for Design-Time RIoT Device Mission Coordination", 2018 26th Telecommunication Forum (TELFOR), Belgrade, pp. 1-4, 2018. https://doi.org/10.1109/telfor.2018.8611845

[120] N. Petrovic, M. Tosic, V. Nejkovic, N. Milosevic, "Formalizing Device Coordination in IoT Systems: The SCOR Case Study", YuInfo 2019, pp. 1-6, 2019.

[121] C. Fairchild, T. L. Harman, *ROS Robotics By Example*, 2nd edition, Packt Publishing, 2017.

[122] Y. S. Pyo, H. C. Cho, R. W. Jung, T. H. Lim, *ROS Robot Programming*, ROBOTIS Co. LTD, 2017.

[123] Turtlebot [online], dostupno na: https://www.turtlebot.com/

[124] OpenCR 1.0 ROBOTIS e-manual [online], dostupno na: https://emanual.robotis.com/docs/en/parts/controller/opencr10

[125] N. Petrović, Đ. Kocić, "IoT-based System for COVID-19 Indoor Safety Monitoring", IcETRAN 2020, pp. 603-608, 2020.

[126] N. Petrovic, "Surveillance System Based on Semantic Video and Audio Annotation Leveraging the Computing Power within the Edge", XIV International SAUM 2018, pp. 281-284, 2018.

[127] X. Qiao et al., Web AR: A Promising Future for Mobile Augmented Reality - State of the Art, Challenges, and Insights, Proceedings of the IEEE 107(4):1-16, 2019. https://doi.org/10.1109/JPROC.2019.2895105

[128] L. Đorđević, N. Petrović, M. Tošić, "An Ontology-based Framework for Automated Code Generation of web AR Applications", Telfor Journal, vol. 12, no. 1, pp. 67-72, 2020. https://doi.org/10.5937/telfor2001067Q

[129] N. Petrović, M. Tošić, V. Nejković, "AR-Enabled Mobile Apps for Robot Coordination", The 7th Conference with International Participation Knowledge Management and Informatics, pp. 1-9, June 2021.

[130] AR.js - Augmented Reality on the Web [online], dostupno na: https://github.com/AR-js-org/AR.js, poslednji pristup: 21/09/2023.

[131] I. Poupyrev, R. Berry, M. Billinghurst, H. Kato, K. Nakao, L. Baldwin, and J. Kurumisawa, "Augmented Reality Interface for Electronic Music Performance", 9th International Conference on Human-Computer Interaction, pp. 805-808, 2001.

[132] A. G. D. Correa, G. A. de Assis, M. Do Nascimento, I. Ficheman, R. de D. Lopes, "GenVirtual: An Augmented Reality Musical Game for Cognitive and Motor Rehabilitation", 2007 Virtual Rehabilitation, pp. 1-6, 2007.

[133] K. Nishida, A. Yuguchi, K. Jo, P. Modler, M. Noisternig, "Border: A Live Performance Based on Web AR and a Gesture-Controlled Virtual Instrument", Proceedings of the International Conference on New Interfaces for Musical Expression, pp. 43-46, 2019.

[134] N. Petrović, "Augmented and Virtual Reality Web Applications for Music Stage Performance" 2020 55th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST), Niš, Serbia, 2020, pp. 33-36, 2020. https://doi.org/10.1109/ICEST49890.2020.9232713

[135] N. Petrović, V. Roblek, M. Khokhobaia and I. Gagnidze, "AR-Enabled Mobile Apps to Support Post COVID-19 Tourism," 2021 15th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS), pp. 253-256, 2021. https://doi.org/10.1109/TELSIKS52058.2021.9606335

[136] M. Radovic, N. Petrovic, M. Tosic, “An Ontology-Driven Learning Assessment Using the Script Concordance Test”, Applied Sciences 12(3), 1472, pp. 1-16, 2022. https://doi.org/10.3390/app12031472

[137] M. Radović, N. Petrović, M. Tošić, “Ontology-based Generation of Multilingual Questions for Assessment in Medical Education”, Journal of Teaching English for Specific and Academic Purposes, vol. 8, no 1, Special Issue, pp. 1-15, May 2020. https://doi.org/10.22190/JTESAP2001001R

[138] N. Petrović, Đ. Kocić, “Podacima-vođena arhitektura za prilagodljive energetske mreže zasnovana na IoT uređajima”, ETRAN 2019, pp. 880-885, 2019.

[139] N. Petrović and Đ. Kocić, “Data-driven framework for energy-efficient smart cities”, Serbian Journal of Electrical Engineering, vol. 17, no. 1, pp. 41-63, Feb. 2020. https://doi.org/10.2298/SJEE2001041P

[140] N. Petrović, Đ. Kocić, "Sistem za relejnu zaštitu zasnovan na Arduino Uno u okviru podacima-vođene arhitekture pametne mreže", IEEESTEC – 12th Student Projects Conference, pp. 331-334, 2019.

[141] N. Petrovic and D. Kocic, "Framework for efficient energy trading in smart grids," 2019 27th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2019, pp. 1-4, 2019. https://doi.org/10.1109/TELFOR48224.2019.8971149

[142] N. Petrovic, “Adopting Semantic-Driven Blockchain Technology to Support Newcomers in Music Industry”, CIIT 2019, pp. 2-7, 2019.

[143] N. Petrović, I. Al-Azzoni, and A. Alqahtani “Model-driven approach to smart grid stability prediction in Neo4j”, eNergetics 2021, 2021. pp. 299-305.

[144] M. Stone, J. Knapper, G. Evans, E. Aravopoulou, “Information management in the smart city, The Bottom Line”, Vol. 31 No. 3/4, 2018, pp. 234-249. https://doi.org/10.1108/BL-07-2018-0033

[145] A. H. Bagdadee, L. Zhang, “Smart Grid: A Brief Assessment of the Smart Grid Technologies for Modern Power System”, Journal of Engineering Technology, vol. 8, no. 1, pp. 122- 142, 2019.

[146] H. Gharavi, R. Ghafurian, “Smart Grid: The Electric Energy System of the Future [Scanning the Issue]”, Proceedings of the IEEE, 99(6), pp. 917–921, 2011. https://doi.org/10.1109/jproc.2011.2124210

[147] J. J. Garcia et al., “Smart City Energy Management via Monitoring of Key Performance Indicators”, CIRED Workshop 2014, pp. 1-5, 2014.

[148] I. S. Bayram, M. Z. Shakir, M. Abdallah, K. Qaraqe, “A survey on energy trading in smart grid”, 2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP), pp. 258-262, 2014. https://doi:10.1109/globalsip.2014.7032118

[149] R. Deng, Z. Yang, M.-Y. Chow, J. Chen, “A Survey on Demand Response in Smart Grids: Mathematical Models and Approaches, IEEE Transactions on Industrial Informatics”, 11(3), pp. 570–582, 201 5. https://doi.org/10.1109/tii.2015.2414719

[150] N. Petrovic and D. Kocic, "Framework for efficient energy trading in smart grids," 2019 27th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2019, pp. 1-4, doi: 10.1109/TELFOR48224.2019.8971149.

[151] Energy efficiency Data Set [online], dostupno na: https://archive.ics.uci.edu/ml/datasets/Energy+efficiency, poslednji pristup: 06/10/2023.

[152] Predicting Smart Grid Stability with Deep Learning [online], dostupno na: https://www.kaggle.com/pcbreviglieri/predictingsmart-grid-stability-with-deep-learning, poslednji pristup: 30/11/2021

[153] B. Schäfer et al., "Taming instabilities in power grid networks by decentralized control", Eur. Phys. J. Spec. Top. 225, 569–582, 2016. https://doi.org/10.1140/epjst/e2015-50136-y

[154] V. Arzamasov, K, Böhm, P. Jochem, "Towards Concise Models of Grid Stability", 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), pp.1-6, 2018. https://doi.org/10.1109/SmartGridComm.2018.8587498

[155] D. Perez-Palacin, J. Merseguer, J. I. Requeno, M. Guerriero, E. Di Nitto and D. A. Tamburri, "A UML Profile for the Design, Quality Assessment and Deployment of Data-intensive Applications", Software & Systems Modeling. pp. 1-38, 2019. https://doi.org/10.1007/s10270-019-00730-3

[156] N. Ferry et al., "CloudMF: Applying MDE to Tame the Complexity of Managing Multi-cloud Applications", 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, 2014, pp. 269-277.

[157] P. Plebani et al., "DITAS: Unleashing the Potential of Fog Computing to Improve Data-Intensive Applications", Advances in Service-Oriented and Cloud Computing, 2018, pp. 154–158. https://doi.org/10.1007/978-3-319-79090-9_11

[158] C. Hong, K. Lee, M. Kang, C. Yoo, "qCon: QoS-Aware Network Resource Management for Fog Computing", Sensors 2018, 18(10), 2018, pp. 1-21. https://doi.org/10.3390/s18103444

[159] R. Singh, N. S. Gill, P. Gulia, "A Comparative Performance Analysis of Modeling and Simulation Tools for Smart Grid", International Journal of Engineering Trends and Technology (70) , pp. 332-342, 2022. https://doi.org/10.14445/22315381/IJETT-V70I4P229

[160] D. Oudart, J. Cantenot, F. Boulanger, S. Chabridon, "The smart grid simulation framework: model-driven engineering applied to cyber-physical systems", MODELSWARD 2020, pp. 3-25, 2020.

[161] T.-T.-Q. Nguyen et al., "Smartlog – A declarative language for distributed programming in smart grids", Computers & Electrical Engineering 80, pp. 1-22, 2019. https://doi.org/10.1016/j.compeleceng.2019.106499

[162] S. Kousar et al., "Formal Modeling of IoT-Based Distribution Management System for Smart Grids", Sustainability, 14, 4499, pp. 1-25, 2022. https://doi.org/10.3390/su14084499

[163] M. Rohrmeier, “Web based robot simulation using VRML”, IEEE WSC 2000, 1525–1528, 2000. http://doi.org/10.1109/wsc.2000.899135

[164] L. Alves et al., “DRESS-ML: a domain-specific language for modelling exceptional scenarios and self-adaptive behaviours for drone-based applications”, ICSE-SEIS '22, pp. 56–66, 2022. https://doi.org/10.1145/3510458.3513012

[165] T. Winiarski, M. Węgierek, D. Seredyński, W. Dudek, K. Banachowicz, and C. Zieliński, “EARL—Embodied Agent-Based Robot Control Systems Modelling Language,” Electronics, vol. 9, no. 2, pp. 1-27, 2020. http://doi.org/10.3390/electronics9020379

[166] A. Vitzthum & H. Heinrich, “Modeling augmented reality user interfaces with SSIML/AR”, Journal of Multimedia 1(3), pp. 13-22, 2006.

[167] E. Aoki et al., “Representing Augmented Reality Applications in Systems Modeling Language” 2023 IEEE International Systems Conference (SysCon), pp. 1-8, 2023. https://doi.org/10.1109/SysCon53073.2023.10131060

[168] S. Feuerstack et al., “Model-based design of multimodal interaction for augmented reality web applications”, Web3D '15: Proceedings of the 20th International Conference on 3D Web Technology, pp. 259–267, 2015. https://doi.org/10.1145/2775292.2775293

[169] S. Pahune, M. Chandrasekharan, “Several Categories of Large Language Models (LLMs): A Short Survey”, International Journal for Research in Applied Science and Engineering Technology 11, pp. 615-633, 2023. https://doi.org/10.22214/ijraset.2023.54677

[170] A. Janković, D. Minčić, N. Petrović, M. Tošić, “ChatGPT Assisted Development of Laravel Applications”, TELSIKS 2023, pp. 1-4, 2023. https://doi.org/10.1109/TELSIKS57806.2023.10316094

[171] N. Petrović, I. Al-Azzoni, “Automated Approach to Model-Driven Engineering Leveraging ChatGPT and Ecore”, 16th International Conference on Applied Electromagnetics – ПЕС 2023, pp. 166-168, 2023. https://www.researchgate.net/publication/373439135_AUTOMATED_APPROACH_TO_MODEL-DRIVEN_ENGINEERING_LEVERAGING_CHATGPT_AND_ECORE

[172] N. Petrović, I. Al-Azzoni, “Model-Driven Smart Contract Generation Leveraging ChatGPT”, In: Selvaraj, H., Chmaj, G., Zydek, D. (eds) Advances in Systems Engineering. ICSEng 2023, Lecture Notes in Networks and Systems, vol 761, Springer, Cham, pp. 387-396, 2023. https://doi.org/10.1007/978-3-031-40579-2_37

[173] M. Brambilla, J. Cabot, M. Wimmer, ”Model-Driven Software Engineering in Practice”, 2017. https://doi.org/10.2200/S00441ED1V01Y201208SWE001

SPISAK SKRAĆENICA

AI – Artificial Intelligence,
AMPL – A Mathematical Programming Language,
API – Application Programming Interface,
AR – Augmented Reality,
ARM – Advanced RISC Machine,
CAD – Computer-Aided Design,
CIDDB - Coburg Intrusion Detection Data Sets,
CO – Coordination Ontology,
CPU – Central Processing Unit,
CSV – Comma-Separated Values,
DAIO – Dynamic Aspects Intelligent System Ontology,
DB–DataBase,
EDL - Experiment Description Language,
ETH – Ethereum, ether (kao merna jedinica tokena)
FP – False Positive,
FN – False Negative,
GDPR– General Data Protection Regulation,
GL – Graphics Library,
glTF - GL Transmission Format,
GO – Generator Ontology,
GPS – Global Positioning System,
GPU – Graphics Processing Unit,
GUI–Graphical User Interface,
HTML- HyperText Markup Language,
HTTP – HyperText Transfer Protocol,
MQTT - Message Queuing Telemetry Transport,
IRI - Internationalized Resource Identifier,
IoT–Internet of Things,
IBk – Instance Based kNN,
IP – Internet Protocol,
ISO – International Organization for Standardization,
IntisOnt - Intelligent System Ontology,
JS–JavaScript,

JSON–JavaScript Object Notation,

JSON-LD – JavaScript Object Notation for Linked Data,

LiDAR – Light Detection and Ranging,

LXC – Linux Containers,

LLM – Large Language Model,

ML–Machine Learning,

MRE – Mean Relative Error,

OS - Operating System,

PC – Personal Computer,

PoS – Proof of Stake,

PoW – Proof of Work,

RAM – Random Access Memory,

RDF–Resource Description Framework,

REST - Representational State Transfer,

RMSE – Root Mean Square Error,

ROS – Robot Operating System,

RPi – Raspberry Pi,

SAIO – Static Aspects Intelligent System Ontology,

SHACL - Shapes Constraint Language,

SLAM - Simultaneous Localization and Mapping,

SPARQL–Standard query language and protocol for Linked Open Data on the web or for RDF triplestores,

SSD – Solid-State Drive,

SSH – Secure Shell,

SQL – Structured Query Language,

TCP – Transmission Control Protocol,

TN – True Negative,

TP – True Positive,

Turtle – Terse RDF Triple Language,

QoS – Quality of Service,

UI – User Interface,

URI –Uniform Resource Identifier,

URL-Uniform Resource Locator,

W3C-World Wide Web Consortium,

Weka - Waikato Environment for Knowledge Analysis

XAI – eXplainable Artificial Intelligence,

XML –Extensible Markup Language,

XSLT–eXtensible Stylesheet Language,

YAML- Yet Another Markup Language

SPISAK SLIKA

SPISAK TABELA

BIOGRAFIJA AUTORA

Nenad N. Petrović je rođen 14. oktobra 1992. godine u Pirotu, Republika Srbija. Osnovnu školu i Gimnaziju je završio u Pirotu. Elektronski fakultet u Nišu je upisao 2011. godine, a diplomirao 2015. sa poveljom *Najbolji diplomirani student osnovnih akademskih studija generacije 2011.*, na odseku Računarstvo i informatika, sa prosekom 10.00. Master studije računarskog inženjerstva upisao je 2015. godine na Univerzitetu Politecnico di Milano u Milanu, Italija, a master rad odbranio jula 2017. godine. Na završnoj godini osnovnih studija i tokom master studija, bio je nosilac stipendije fonda „Dositeja“. Tokom studija u Italiji je radio kao student-istraživač, a kasnije i kao informatički konsultant i poslovni analitičar u firmi „Tech Rain“, Milano, Italija.

Doktorske studije iz oblasti računarstva i informatike upisuje na Elektronskom fakultetu u Nišu 2017. godine. Po povratku u Srbiju, počinje da radi i kao asistent na Katedri za računarstvo i informatiku, od aprila 2018. Iza sebe ima bogato iskustvo u nastavi, učestvujući u aktivnostima za realizaciju više predmeta na osnovnim akademskim studijama: Algoritmi i programiranje, Logičko projektovaje, Mikroračunarski sistemi, Internet stvari, Informacioni sistemi, Informacione tehnologije i sistemi, Programski prevodioci, Metodi i sistemi za obradu signala.

Iza sebe ima preko 150 naučnih publikacija, štampanih u zbornicima nacionalnih i internacionalnih konferencija, ali i časopisima od međunarodnog značaju, pri čemu trenutno ima 6 objavljenih radova u časopisima sa SCI liste i 2 poglavlja u monografijama. Osim toga, prilično je aktivan kao recenzent – više puta je bio član programskog odbora i vodio sesije na međunarodnim konferencijama, ali i recenzirao preko 30 radova za časopise, pri čemu je više od pola sa SCI liste. Učestvovao je u uspešnoj realizaciji nekoliko Horizon 2020 projekata od 2018. pa nadalje.

ИЗЈАВА О АУТОРСТВУ

Изјављујем да је докторска дисертација, под насловом

Развој интелигентних система вођен доменским онтологијама

која је одбрањена на Електронском факултету Универзитета у Нишу:

- резултат сопственог истраживачког рада;

- да ову дисертацију, ни у целини, нити у деловима, нисам пријављивао на другим факултетима, нити универзитетима;

- да нисам повредио ауторска права, нити злоупотребио интелектуалну својину других лица.

Дозвољавам да се објаве моји лични подаци, који су у вези са ауторством и добијањем академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада, и то у каталогу Библиотеке, Дигиталном репозиторијуму Универзитета у Нишу, као и у публикацијама Универзитета у Нишу.

У Нишу, 20.06.2024

Потпис аутора дисертације:

Ненад Петровић

Ненад Н. Петровић

ИЗЈАВА О ИСТОВЕТНОСТИ ЕЛЕКТРОНСКОГ И ШТАМПАНОГ ОБЛИКА ДОКТОРСКЕ ДИСЕРТАЦИЈЕ

Наслов дисертације:

Развој интелигентних система вођен доменским онтологијама

Изјављујем да је електронски облик моје докторске дисертације, коју сам предао за уношење у **Дигитални репозиторијум Универзитета у Нишу**, истоветан штампаном облику.

У Нишу, 20.06.2024

Потпис аутора дисертације:

Ненад Петровић

Ненад Н. Петровић

ИЗЈАВА О КОРИШЋЕЊУ

Овлашћујем Универзитетску библиотеку „Никола Тесла“ да у Дигитални репозиторијум Универзитета у Нишу унесе моју докторску дисертацију, под насловом:

Развој интелигентних система вођен доменским онтологијама

Дисертацију са свим прилозима предао сам у електронском облику, погодном за трајно архивирање.

Моју докторску дисертацију, унету у Дигитални репозиторијум Универзитета у Нишу, могу користити сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons), за коју сам се одлучио.

1. Ауторство **(CC BY)**
2. Ауторство – некомерцијално **(CC BY-NC)**
3. **Ауторство – некомерцијално – без прераде (CC BY-NC-ND)**
4. Ауторство – некомерцијално – делити под истим условима **(CC BY-NC-SA)**
5. Ауторство – без прераде **(CC BY-ND)**
6. Ауторство – делити под истим условима **(CC BY-SA)**

У Нишу, 20.06.2024

Потпис аутора дисертације:

Ненад Петровић

Ненад Н. Петровић