

**SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN**

Goran Filinić

**WISEAGENTNI SUSTAV ZA INTERAKCIJU S
OGLASIMA PREUZETIH OD WEB
STRANICA**

PROJEKT

VIŠEAGENTNI SUSTAVI

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Goran Filinić

Matični broj: 0016124457

Studij: Baze podataka i baze znanja

**WISEAGENTNI SUSTAV ZA INTERAKCIJU S OGLASIMA PREUZETIH
OD WEB STRANICA**

PROJEKT

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, siječanj 2024.

Izjava o izvornosti

Izjavljujem da je ovaj projekt izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvatanjem odredbi u sustavu FOI Radovi

Sažetak

Ovaj rad fokusira se na implementaciju višeagentnog sustava za interakciju s oglasima automobila s web stranice *Index Oglasi*. Sustav je razvijen korištenjem Spade, platforme za izradu multiagentnih sustava u Pythonu. Glavna svrha sustava je informiranje korisnika putem Telegrama o novopristiglim oglasima za automobile koji bi ga mogli zanimati. Rad detaljno opisuje arhitekturu svakog agenta, metode komunikacije među agentima, te način donošenja odluka. Kroz evaluaciju, provjerena je funkcionalnost sustava u odnosu na postavljene ciljeve projekta, dok su izazovi i lekcije naučene tijekom implementacije raspravljani. Razvijeni sustav pruža dinamično i fleksibilno rješenje za praćenje oglasa i olakšava korisnicima pregovaranje u svijetu online automobilskih oglasa.

Ključne riječi: analiza cijena; automobili; Python; Spade; Telegram; višeagentni sustav; web scraping

Sadržaj

1. Uvod	1
2. Metode i tehnike rada	2
3. Višeagentni sustav	3
3.1. Komunikacija agenata	3
4. Opis i prikaz problema	5
5. Implementacija	7
5.1. Arhitektura rješenja	7
5.2. Implementacija pojedinih agenta	9
5.2.1. TopLevelAgent	10
5.2.2. DatabaseAgent	10
5.2.3. PriceAgent	12
5.2.3.1. Buyer	16
5.2.3.2. Seller	20
5.2.4. TelegramAgent	22
6. Zaključak	25
Popis literature	26
Popis slika	27
Popis isječaka koda	28

1. Uvod

Ovaj projektni rad ima za cilj implementirati višeagentni sustav za interakciju s oglasima preuzetim s web stranice *Index Oglasi*. Višeagentni pristup pruža model u kojem autonomne jedinice, nazvane agentima, sudjeluju u dinamičkom procesu pregovaranja i analize oglasa. Razvoj sustava temelji se na upotrebi Spade, open-source platforme za izradu multiagentnih sustava u programskom jeziku Python.

Sustav se sastoji od ključnih agenata, a svaki agent obavlja specifičnu ulogu u procesu interakcije s oglasima. *Web-Scraping Agent* zadužen je za prikupljanje informacija o oglasima putem web scrapinga, *Database Agent* upravlja lokalnom bazom podataka, *Price Agent* analizira cijene prethodnih oglasa, *Buyer Agent* predstavlja potencijalnog kupca, *Seller Agent* predstavlja prodavatelja, dok *Telegram Agent* šalje obavijesti korisnicima putem Telegrama o novim oglasima s povoljnim cijenama.

Rad će detaljno opisati arhitekturu i implementaciju svakog pojedinog agenta, naglašavajući ulogu svakog u sustavu. Kroz projekt će se istražiti mogućnosti komunikacije između agenata, donošenje odluka te analiza prikupljenih podataka u svrhu optimizacije pregovaračkog procesa.

2. Metode i tehnike rada

U radu je korištena deskriptivna metoda za prikupljanje informacija o višeagentnim sustavima i konceptima kako bi se mogao izraditi projekt sa željenim ciljem. Također, korištena je metoda analize za analizu alata i dosadašnjih istraživanja višeagentnih sustava.

Tehnika "scraping" je korištena kako bi se projekt realizirao, omogućujući agentima da automatski prikupe informacije s ciljane web stranice - Index Oglasa. Ovom tehnikom, agenti simuliraju ljudsko ponašanje pretraživanja weba kako bi ekstrahirali željene podatke, u slučaju ovog projekta podatke o automobilima.

Kako bi se naposljetku izradio projektni zadatak korišteni su sljedeći alati: python, biblioteke pythona SPADE, json, re, BeautifulSoup, requests; Telegram aplikacija, sustav Git za verzioniranje koda i XAMPP prosody poslužitelj na kojem su agenti aktivni

3. Višeagentni sustav

Agenti su autonomni entiteti koji se mogu organizirati u jedinice i surađivati kako bi rješavali probleme različite kompleksnosti [1]. Stoga, sustavi s više agenata MAS (*eng. Multi-Agent System*) su sustavi sastavljeni od više agenata koji međusobno djeluju u jednom okruženju. Također, zbog njihove inherentne sposobnosti interaktivnosti, agenti se mogu koristiti kao paradigma pri dizajniranju kompleksnih distribuiranih sustava. Pomoću njihove interoperabilnosti, svaki agent može djelovati kako bi maksimizirao očekivani izlaz sustava i prilagodio se neočekivanim okolnostima [2].

Ovaj projekt mogao se izvesti i bez implementacije višeagentnog sustava, ali distribucija zadaća, paralelnost i modularnost su razlozi za eleganciju implementiranog sustava. Uvođenjem više agenta, omogućava se efikasnija raspodjela resursa te bolja skalabilnost i prilagodljivost sustava promjenama uvjeta. Sposobnost agenata da surađuju i međusobno komuniciraju doprinosi agilnosti i efikasnosti u rješavanju problema, čineći sustav otpornijim na promjene i dinamičnim izazovima.

3.1. Komunikacija agenata

Jezik komunikacije agenata (*eng. Agent Communication Language - ACL*) uspostavlja strukturu za razmjenu poruka između agenata, kako po vrsti tako i po značenju. Razgovor se može smatrati unaprijed dogovorenim protokolom ili obrascem razmjene poruka usmjerenim prema određenom zadatku ili cilju.

Dva učestala jezika za komunikaciju agenta su FIPA-ACL i KQML. Oba jezika temelje se na teoriji govornih činova i sastoje se od različitih performativa [3]. Mogu se shvatiti kao rečenice koje istovremeno opisuju i utječu na okolinu. Poruke razmijenjene između agenata mogu predstavljati radnje ili komunikacijske aktivnosti. Unatoč svojoj relativnoj starosti, ovi standardi se i dalje koriste do danas [2].

FIPA protokol 2 predstavlja smjernice kako bi se platforme učinile interoperabilnima. FIPA-ACL sastoji se od 22 komunikacijska performativa. Najčešći performativi su [2]:

- *inform*: Pošiljatelj obavještava primatelja da je određena propozicija istinita;
- *request*: Pošiljatelj traži od primatelja izvršenje određene radnje;
- *agree*: Pošiljatelj se slaže izvršiti određenu radnju, možda u budućnosti;
- *not understood*: Pošiljatelj (npr., i) obavještava primatelja (npr., j) da je primijetio radnju izvršenu od strane j, ali je nije razumio;
- *refuse*: Pošiljatelj odbija izvršiti određenu radnju, objašnjavajući razlog odbijanja.

I dok FIPA-ACL samo uspostavlja protokole interakcije između agenata, postoje i FIPA specifikacije za upravljanje agentima koje se koriste u kombinaciji s komunikacijskim standar-

dima. Osnovni dijelovi tih specifikacija su (i) Platforma za Agente, gdje svaki agent ima jedinstveni identifikator nazvan AID (FIPA Agent Identifier); (ii) Sustav za Upravljanje Agentima (AMS), odgovoran za upravljanje Platformom za Agente; i (iii) Facilitator Direktorija (DF), koji omogućava agentima objavljivanje usluga na otkrivan način.

Slično tome, jezik KQML također se sastoji od komunikacijskih performativa. Posjeduje tri različita sloja:

1. *Sloj sadržaja*: gdje se stvarni sadržaj poruke nalazi, u jeziku vlastitog računala;
2. *Komunikacijski sloj*: koristi se za kodiranje nižih komunikacijskih parametara, kao što su identitet pošiljatelja i primatelja poruke;
3. *Sloj poruke*: pruža performativske elemente korištene u procesu komunikacije, pronalazeći sve moguće interakcije s agentima koji podržavaju KQML.

KQML također opisuje posebnu klasu agenata nazvanu facilitatori. Facilitator je odgovoran za obavljanje različitih komunikacijskih usluga, poput održavanja registra usluga, rutiranja poruka prema tim uslugama na temelju sadržaja i posredovanja između dijelova komunikacije. Za razliku od FIPA-ACL, ove specifikacije sadržane su u definiciji ACL-a i ne ovise o dodatnim standardima [2].

Iako su slični u više aspekata, postoje neke karakteristike specifične za jedan ili drugi ACL koje čine njihovu primjenu ovisnom o domeni. Ove razlike mogu biti ključne pri odabiru jednog ili drugog ACL-a prema određenom skupu zahtjeva [2].

4. Opis i prikaz problema

U sklopu ovog projekta, naglasak je stavljen na implementaciju višeagentnog sustava koji ima zadatak preuzimanja oglasa s popularnih web stranica te njihovo pohranjivanje. Korisniku se zatim šalju obavijesti ako su zadovoljeni određeni parametri. Specifična web stranica koja se koristi za dohvaćanje oglasa je "Index oglasi", kako je prikazano na Slici 1. Fokus dohvaćanja oglasa usmjeren je na kategoriju automobila, a ta kategorija posjeduje kompleksnu strukturu podataka koja uključuje različite attribute poput godine proizvodnje, marke, kilometraže, i drugih relevantnih informacija. Važno je napomenuti da je struktura skripte za dohvaćanje podataka prilagođena od strane autora projekta, uzimajući kao polazište originalnu skriptu autora koji je pružio primjer *scrapanja* podataka za aute s navedene web stranice [4]. U poboljšanoj skripti, atribut marke vozila se automatski popunjava na temelju naziva oglasa ukoliko nije eksplicitno definiran, a struktura je dodatno modificirana radi bolje čitljivosti i efikasnije obrade podataka.

Temeljna ideja obaviještavanja korisnika je jednostavno usmjerena na analizu marki automobila i prosječnih cijena, koristeći prikupljene podatke iz prošlosti. Konkretno, sustav uspoređuje cijene novo prikupljenih oglasa za određenu marku automobila s prosječnom cijenom zabilježenom u bazi podataka. U slučaju da je cijena novopristiglih oglasa niža od prosječne cijene, korisnik će primiti obavijest o tom oglasu. Ovaj pristup omogućuje korisnicima da budu obaviješteni o potencijalno povoljnim ponudama, prateći trendove u cijenama automobila na temelju prethodno prikupljenih podataka.

Stoga u projektu se traži implementacija slijedećih funkcionalnosti:

- **Web-Scraping agent:** Razviti agenta koji će obavljati web scraping na "Index oglasa" kako bi preuzeo relevantne informacije o oglasima za automobile. Agent mora biti sposoban razumjeti kompleksnu strukturu podataka na web stranici.
- **Agent za spremanje Podataka:** Implementirati agenta koji ima mehanizam za pohranu prikupljenih podataka o oglasima. Ovi podaci trebaju biti organizirani na način koji olakšava analizu i usporedbu, sortirani od najmlađih do najstarijih.
- **Agent za analizu cijena:** Razviti agenta koji će analizirati cijene prijašnjih oglasa i prosječnih cijena marki automobila. Također agent treba generirati nekakvu preporučenu cijenu pregovaranja za korisnika kako bi imao polazišnu točku.
- **Agent za slanje obavijesti:** Ukoliko cijena novopristiglih oglasa za određenu marku bude niža od prosjeka, ovaj agent šalje obavijest korisniku preko telegram sučelja.


INDEXOGLASI
Upišite pojam...
Sve Županije
Predaj oglas besplatno
Registracija
Prijava

Index Oglasi / Auto-moto / Osobni automobili


Osobni automobili
pronađeno 34.739 oglasa

Napredni oglasi
100 rezultata


PONUĐA (34.739)
POTRAŽNJA (173)
NA POKLON (4)
MARKA
Sve marke
MODEL VOZILA
Svi modeli
LOKACIJA VOZILA
Sve Županije
GODINA PROJEVOENJE
DO
KILOMETRAŽA
DO



Renault Thalia 1.4 RT
Uređni servisirani auto, 1 vlasnik, reg. do 26.06.2024.
GODIŠTE: 2001 | KM: 221.000 | STAROST: RABljENO | KW: 55
Medimurska | Objavljeno prije 2 min
1.550 € ~ 11.678 kn



Citroen C4 1.6 Benzin
GODIŠTE: 2007 | KM: 75.000 | STAROST: RABljENO | KW: 80
Splitko-darmatinska | Objavljeno prije 8 min
2.400 € ~ 18.083 kn



Renault Twingo 1.2 8v
GODIŠTE: 2005 | KM: 190.400 | STAROST: RABljENO | KW: 43

Slika 1: Stranica s koje se preuzimaju oglasi

5. Implementacija

U razvoju ovog projekta, korištena je SPADE Python biblioteka u kombinaciji s Prosody XMPP serverom kako bi se omogućio lokalni servis za rad agenata. Ova kombinacija alata pruža efikasno okruženje za implementaciju višeagentnog sustava. Za interakciju s korisnicima, integriran je Telegram chatbot, pružajući korisnicima jednostavan način za primanje obavijesti.

Svi prikupljeni podaci, uključujući informacije o oglasima i prosječne cijene, pohranjeni su u JSON formatu. Odluka za uporabu JSON formata proizlazi iz jednostavnosti i efikasnosti čitanja i pisanja podataka. S obzirom na relativno jednostavnu strukturu podataka koja ne zahtijeva složenu međusobnu povezanost, JSON pruža pogodno rješenje za pohranu informacija unutar projekta. Ova prilagodljiva i jednostavna struktura podataka olakšava manipulaciju informacijama, što je ključno za učinkovito funkcioniranje sustava.

Važno je napomenuti da ovaj projekt ne posjeduje tradicionalno korisničko sučelje. Umjesto toga, implementiran je kao konzolna aplikacija, a jedina interakcija s korisnikom odvija se putem Telegram obavijesti ili poruka. Ova jednostavna i čista komunikacija omogućava korisnicima jednostavan pristup informacijama o oglasima, bez potrebe za složenim korisničkim sučeljem.

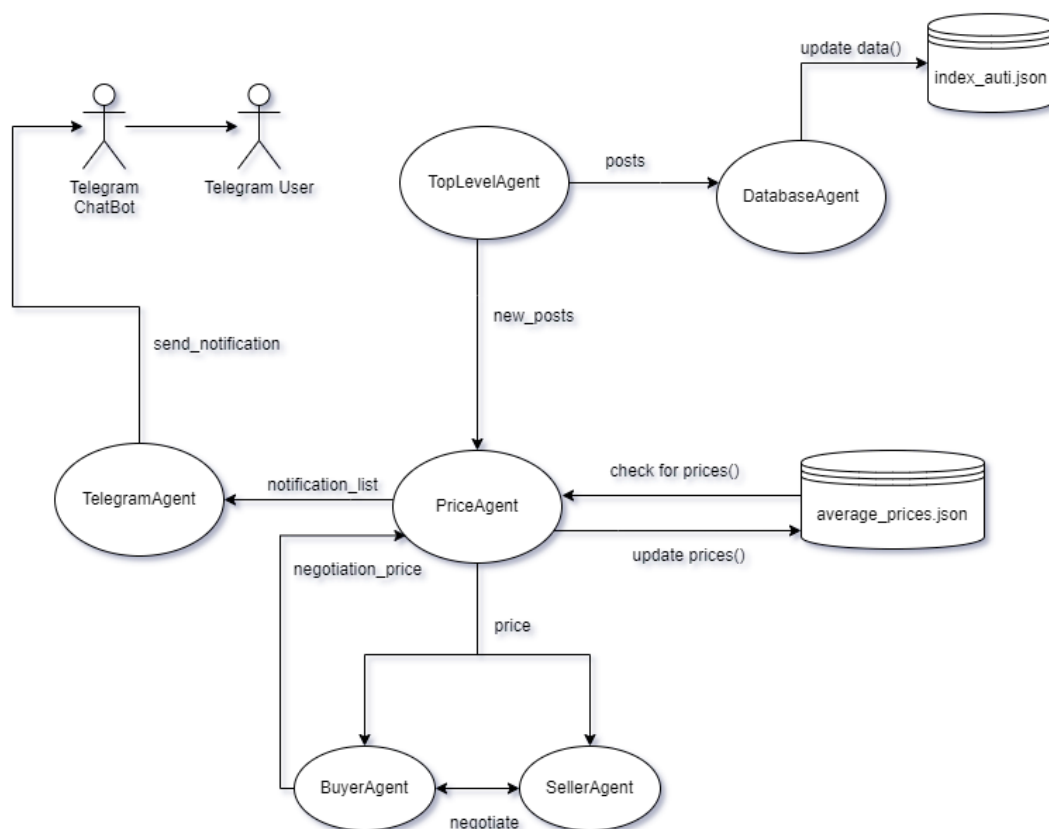
5.1. Arhitektura rješenja

Kako bi se prikazala struktura implementiranog rješenja prikazati će se dva dijagrama:

- **Dijagram arhitekture rješenja:** Ovaj dijagram pružit će široki pregled ključnih objekata unutar sustava, uključujući agente, interakciju s korisnikom, tehnologije korištene za web scraping, pohranu podataka i komunikaciju.
- **UML dijagram slijeda:** UML dijagram slijeda pružit će detaljan prikaz međusobne interakcije između različitih komponenti sustava tijekom izvršavanja ključnih funkcionalnosti, poput web scrapinga, analize podataka i slanja obavijesti korisnicima.

Ovi dijagrami dodatno pojašnjavaju arhitekturu i funkcioniranje sustava te olakšava razumijevanje kako pojedine komponente surađuju kako bi postigle funkcionalnosti definirane u prethodnom poglavlju. Slika 2 prikazuje dijagram arhitekture rješenja. Ova arhitektura ilustrira organizaciju svih agenata koji su spomenuti u prethodnom poglavlju. Agenti i njihove uloge su slijedeći:

- **TopLevelAgent:** Glavni agent definiran s cikličkim ponašanjem svakih 10 minuta. Njegova uloga je prikupljanje podataka putem web scrapinga i slanje tih podataka agentu za bazu podataka.
- **DatabaseAgent:** Ovaj agent provjerava stanje podataka u bazi *index_auti.json*. Na temelju najnovijeg oglasa ažurira stanje baze podataka.

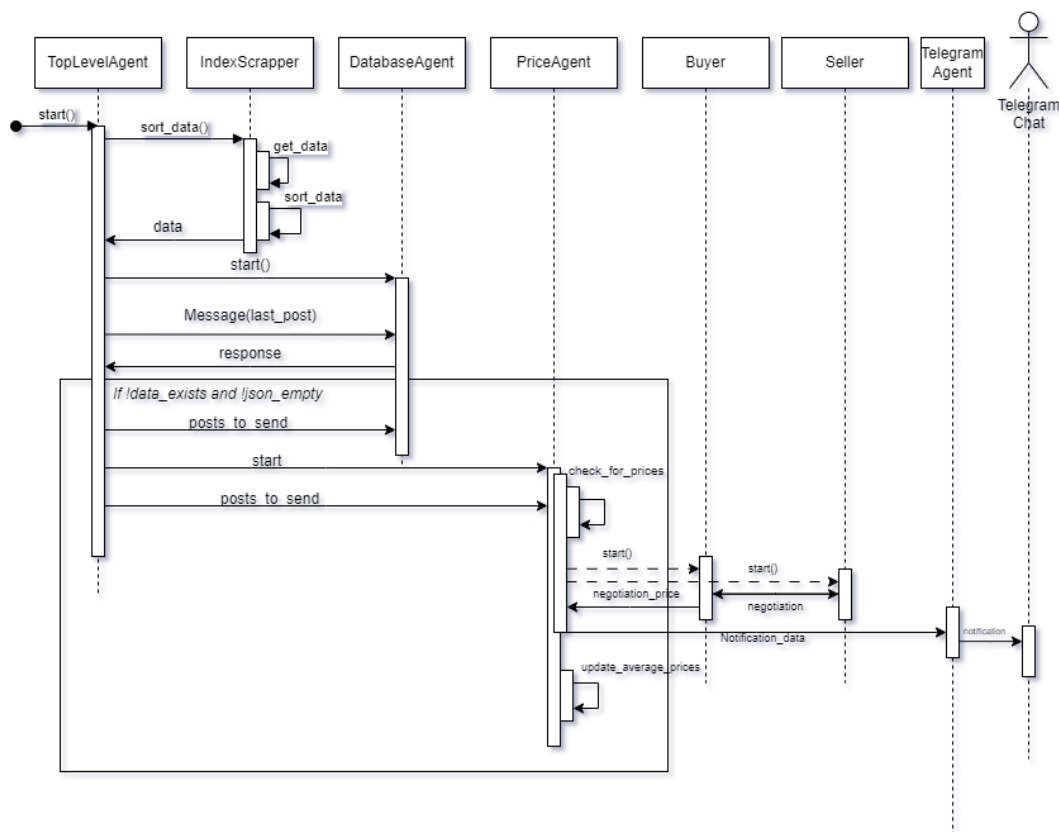


Slika 2: Dijagram arhitekture rješenja

- **PriceAgent:** Agent provjerava stanje podataka u bazi *average_prices.json*. Na temelju tih podataka uspoređuje ih s novim cijenama u oglasima te sastavlja listu oglasa koje je potrebno proslijediti korisnicima. Osim toga, ovaj agent kreira dva dodatna agenta, *Buyer* i *Seller*, koji pokušavaju postići dogovor. Ovo jednostavno implementirano rješenje simulira pregovaranje i dobiva rezultate koje potom predlaže korisnicima
- **TelegramAgent:** Jednostavni agent koji prima skup obavijesti te ih prosljeđuje Telegram chatbotu kako bi korisnici primili obavijesti.

Da bi se pružio bolji prikaz u tijek aktivnosti tijekom izvođenja, Slika 3 prikazuje dijagram slijeda. Na ovom dijagramu primjetan je posebni objekt *IndexScrapper*, koji u stvarnosti poziva *TopAgent* kako bi dohvaćao podatke. Implementacija ovog objekta dodana je radi poboljšane čitljivosti i preglednosti programskog koda. Također, uočljivo je kako je implementirana grana uvjeta. Ako *DatabaseAgent* ne registrira nove oglase, cijeli proces uspoređivanja cijena, pregovaranja i slanja obavijesti neće se izvršiti.

Ovaj dijagram je izuzetno koristan jer olakšava praćenje tijeka procesa, posebno prilikom izvršavanja konzolne aplikacije. S obzirom na različite ispisane poruke i načine stvaranja i gašenja agenata, tijekom procesa može biti konfuzan. Važno je naglasiti da je ovaj dijagram pojednostavljeni model s ključnim elementima. Detalji pojedinih agenata bit će detaljnije razrađeni u slijedećem poglavlju.



Slika 3: Dijagram slijeda

5.2. Implementacija pojedinih agenta

Ovaj odjeljak projektne dokumentacije detaljno opisuje implementaciju pojedinih agenta u višeagentnom sustavu. Cilj je pružiti jasan uvid u uloge, funkcionalnosti i međusobnu interakciju svakog agenta unutar sustava. Svaki agent obavlja specifične zadatke s namjerom doprinosa ostvarenju funkcionalnosti sustava.

Pojedinačni agenti, kao što su *TopLevelAgent*, *DatabaseAgent*, *PriceAgent* i *TelegramAgent*, imaju precizno definirane zadatke unutar višeagentnog okvira. Ovaj odjeljak pruža objektivne opise njihovih ponašanja i funkcionalnosti, omogućavajući čitatelju bolje razumijevanje načina njihove suradnje i doprinosa konačnom cilju projekta.

Uz opise agenata, analizirane su i njihove međusobne interakcije, naglašavajući ključne trenutke razmjene informacija, donošenja odluka ili izvođenja specifičnih ponašanja. Ovaj odjeljak također pruža pregled važnih aspekata implementacije uz isječke kôda koji čine temelj uspješnog funkcioniranja sustava.

Kroz dubinsku analizu svakog agenta, cilj je steći bolje razumijevanje o načinu djelovanja višeagentnog sustava kao cjeline, pridonoseći postizanju ciljeva projekta.

```

1 class TopLevelAgent (Agent):
2     class TopLevelBehaviour (CyclicBehaviour):
3         async def run(self):
4             print ("-----")
5             await self.initialize_scraper()

6             data = self.scraper.sort_data()
7             comparison_task = asyncio.create_task(self.start_agents(data))
8             await asyncio.gather(comparison_task)
9             await asyncio.sleep(600)

10    async def setup(self):
11        print(f"Top-level agent {self.jid} is starting...")
12        b = self.PeriodicTaskBehaviour()
13        self.add_behaviour(b)

```

Isječak koda 1: TopLevelAgent, inicijalizacija i run petlja

5.2.1. TopLevelAgent

TopLevelAgent predstavlja glavnog agenta u sustavu. Ovaj agent pokreće proces prikupljanja podataka o oglasima s web stranice "Index oglasi" putem *IndexScraper* objekta. Isječci kôda 1 i 2 prikazuju inicijalizaciju agenta i glavnu petlju:

Isječak kôda 1 prikazuje definiciju klase *TopLevelAgent* koja nasljeđuje *Agent* iz SPADE biblioteke. Unutar klase, definirana je unutarnja klasa ponašanja *TopLevelBehaviour*, koja nasljeđuje *CyclicBehaviour*. Glavna petlja agenta (*async def run(self):*) pokreće se svakih 10 minuta, pri čemu se prvo inicijalizira *IndexScraper* objekt (metoda *initialize_scraper()*). Zatim se podaci sortiraju, pokreće se zadatak usporedbe (*start_agents*), i agent čeka 10 minuta prije nego što ponovno pokrene proces. Metoda *setup* dodaje cikličko ponašanje agentu pri njegovom pokretanju.

Isječak kôda 2 prikazuje zadatak usporedbe unutar *TopLevelAgent* klase. Prvo se kreira *DatabaseAgent* kako bi se provjerilo stanje podataka u lokalnoj bazi. Zatim se dohvaća zadnji oglas iz trenutno prikupljenih dostupnih podataka. Oglas se šalje *DatabaseAgentu*, čeka se odgovor, te se na temelju odgovora donose daljnje odluke.

Ukoliko odgovor sadrži metapodatke koji označavaju da je taj oglas prisutan u lokalnoj bazi i ažuran je, sustav ne poduzima daljnje korake, već se čeka slijedeći ciklus prikupljanja podataka. Ukoliko lokalna datoteka s bazom oglasa ne postoji (*json_empty*), svi trenutni podaci se šalju *DatabaseAgentu* kako bi se pohranili (2, linije 16-21). Ako postoje podaci, šalju se svi oglasi od prvog do posljednjeg unesenog oglasa u bazi (2, linije 26-36).

Ovaj postupak osigurava ažurnost podataka u lokalnoj bazi i omogućava prijenos novih oglasa od zadnjeg unesenog oglasa u bazi do trenutno dostupnih podataka.

5.2.2. DatabaseAgent

DatabaseAgent ima ključnu ulogu u provjeri stanja podataka u lokalnoj bazi (*Index_auti.json*). Agent ima dvije klase ponašanja, *CheckIfEmptyBehavior* i *ProcessDataBehavior*, koji se izvr-

```

1 class TopLevelAgent (Agent):
2     class TopLevelBehaviour (CyclicBehaviour):
3         async def start_agents (self, data):
4             database_agent = DatabaseAgent ("database@localhost", "db")
5             await database_agent.start ()
6             last_post = data[0] if data else None
7             message = spade.message.Message (to=str (database_agent.jid))
8             message.body = json.dumps (last_post)
9             response = await send_and_wait_for_response (self, message)
10
11             % Process the response based on its metadata
12             if response:
13                 print ("got response!", response)
14                 if response.metadata.get ("data_exists", True):
15                     print ("No new posts.")
16                 elif response.metadata.get ("json_empty", True):
17                     print ("No local file.")
18                     message = spade.message.Message (to=str (database_agent.jid))
19                     message.body = json.dumps (data)
20                     await self.send (message)
21                 else:
22                     print ("New posts detected")
23                     last_post = response.body
24                     last_post_index = next ((i for i, post in enumerate (data) if
25                                     ↳ post ["poveznica"] == last_post), None)
26                     posts_to_send = data [:last_post_index] if last_post_index is not
27                                     ↳ None else data
28                     if posts_to_send:
29                         message = spade.message.Message (to=str (database_agent.jid))
30                         message.body = json.dumps (posts_to_send)
31                         await self.send (message)
32                         print (f"Sent {len (posts_to_send)} posts to DatabaseAgent.")
33                         await self.price_agent.start ()
34
35                     message =
36                         ↳ spade.message.Message (to=str (self.price_agent.jid))
37                     message.body = json.dumps (posts_to_send)
38                     await self.send (message)
39                     print (f"Sent {len (posts_to_send)} posts to PriceAgent.")

```

Isječak koda 2: TopLevelAgent, glavna funkcija usporedbe

šavaju tijekom njegove aktivnosti. Isječak kôda 3 prikazuje ponašanje *CheckIfEmptyBehavior*, koje se izvršava prilikom pokretanja *DatabaseAgent*a. Ovo ponašanje ima zadatak provjeriti postoji li lokalna datoteka (*Index_auti.json*).

Ako datoteka ne postoji, šalje se odgovor da datoteka nije pronađena. *TopLevelAgent* potom šalje sve dohvaćene podatke i kreira se lokalna datoteka i agent se zaustavlja. Ako datoteka postoji, pokreće se drugo ponašanje *ProcessDataBehavior*.

Isječak kôda 4 obuhvaća funkcionalnosti *ProcessDataBehavior* ponašanja, koje se izvršava nakon što je *DatabaseAgent* utvrdio da lokalna baza podataka nije prazna i da je primio novi oglas od *TopLevelAgent*a. To se postiže usporedbom poveznice zadnjeg oglasa u lokalnoj bazi (*self.agent.local_data[0][\"poveznica\"]*) s poveznicom primljenog oglasa (*json.loads(self.agent.data)[\"poveznica\"]*).

Ako su poveznice jednake, to znači da primljeni oglas već postoji u lokalnoj bazi, te agent šalje poruku *\"Data is in sync!\"* *TopLevelAgent*u, označava da su podaci usklađeni, i čeka na nove podatke u sljedećem ciklusu.

Ako poveznice nisu jednake, to znači da postoje novi oglas koji treba obraditi. Agent prvo šalje poruku *textitTopLevelAgent*u s poveznicom zadnjeg oglasa u lokalnoj bazi. Ovo se postiže slanjem poruke s tijelom *self.agent.local_data[0][\"poveznica\"]*. Osim toga, agent označava da nema prazne JSON datoteke (*\"json_empty\": False*) i da podaci postoje (*\"data_exists\": False*). Nakon toga agent čeka listu oglasa od *TopLevelAgent*a. Konačno, agent poziva funkciju *append_json(data)* kako bi ažurirao lokalnu bazu podataka s novim podacima, nakon čega zaustavlja svoje izvršavanje.

Ovaj isječak kôda ključan je za održavanje ažurnosti lokalne baze podataka, spremanje novih podataka i komunikaciju s *TopLevelAgent*om kako bi ga obavijestio o stanju ažurnosti podataka.

Isječak kôda 5 prikazuje strukturu *DataBaseAgent*a i agentovo početno stanje.

5.2.3. PriceAgent

PriceAgent je agent specijaliziran za praćenje cijena automobila putem novih oglasa. Koristeći konačni automat, *PriceAgent* organizira svoje ponašanje kroz tri ključna stanja. Tijekom svog ciklusa, agent provjerava i ažurira prosječne cijene automobila, pravi listu obavijesti o oglasima ispod prosjeka, te kreira i interagira s dodatnim agentima *Buyer* i *Seller* koji simuliraju pregovaranje. Sve informacije o prosječnim cijenama pohranjuju se u datoteku nazvanu *average_prices.json*. A lista obavijesti popunjeni sa podacima proslijeđuju se *TelegramAgent*u.

Isječak koda 6 prikazuje prvo stanje (*StateOne*) u kojem *PriceAgent* provjerava postoji li datoteka *average_prices.json* koja sadrži prosječne cijene. Ako datoteka ne postoji, agent ju stvara i popunjava prosječnim cijenama automobila na temelju podataka iz datoteke *Index_auti.json*. Nakon tog koraka, agent završava svoje izvršavanje kako bi spriječio daljnje korake budući da su podaci već ažurirani od strane *DatabaseAgent*-a.

Isječak kôda 7 predstavlja drugo stanje (*StateTwo*) unutar *PriceAgent*-a. U ovom sta-

```

1 class CheckIfEmptyBehavior(OneShotBehaviour):
2     async def on_start(self):
3         print(f"Database agent is starting...")
4
5     async def run(self):
6         self.agent.local_data = []
7         try:
8             with open('Index_auti.json', 'r', encoding='utf-8') as jsonfile:
9                 self.agent.local_data = json.load(jsonfile)
10                if self.agent.local_data:
11                    self.agent.empty = False
12            except FileNotFoundError:
13                print("Local JSON file not found.")
14                self.agent.empty = True
15
16        msg = await self.receive(timeout=100000)
17        if msg:
18            self.agent.data = msg.body
19            self.agent.sender_jid = msg.sender
20            print("database agent got message")
21            if self.agent.empty:
22                message = spade.message.Message(to=str(self.agent.sender_jid))
23                message.body = "File not found"
24                message.metadata["json_empty"] = True # Add metadata indicating the
25                ↪ JSON file is empty
26                # After updating local data
27                message.metadata["data_exists"] = False
28
29            print("sent message")
30            response = await send_and_wait_for_response(self, message)
31            if response:
32                data = response.body
33                try:
34                    await write_json(data)
35                    await self.agent.stop()
36                except json.JSONDecodeError as e:
37                    print(f"Error decoding JSON: {e}")
38            elif not self.agent.empty:
39                b = self.agent.ProcessDataBehavior()
40                self.agent.add_behaviour(b)
41        else:
42            await self.agent.stop()

```

Isječak koda 3: DatabaseAgent, Ponašanje *CheckIfEmptyBehaviour*

```

1 class ProcessDataBehavior(OneShotBehaviour):
2     async def run(self):
3         print("Database agent: Processing data")
4         print("Last post ", str(self.agent.local_data[0]), "\n")
5         print("data:", str(self.agent.data), "\n")
6
7         # Check if the received data is the same as the last post in the local data
8         if self.agent.local_data[0]["poveznica"] ==
9             ↳ json.loads(self.agent.data)["poveznica"]:
10             print("Received post is the same as the last one in the database.")
11             message = spade.message.Message(to=str(self.agent.sender_jid))
12             message.body = "Data is in sync!"
13             message.metadata["json_empty"] = False
14             message.metadata["data_exists"] = True
15             await self.send(message)
16         else:
17             # Process the new post, update local data, and send a response
18             print("Received a new post. Processing...")
19             # Add your logic to process the new post here
20
21             # Send a response indicating successful processing
22             response_message = spade.message.Message(to=str(self.agent.sender_jid))
23             response_message.body = self.agent.local_data[0]["poveznica"]
24             response_message.metadata["json_empty"] = False
25             response_message.metadata["data_exists"] = False
26             print("sent message to top LevelAgent: ", response_message.body)
27             response = await send_and_wait_for_response(self, response_message)
28             if response:
29                 data = response.body
30                 try:
31                     await append_json(data)
32                 except json.JSONDecodeError as e:
33                     print(f"Error decoding JSON: {e}")
34                 await self.agent.stop()
35                 self.write_json(json.dumps(local_data))

```

Isječak koda 4: DatabaseAgent, Ponašanje *ProcessDataBehavior*

```

1 class DatabaseAgent (Agent):
2     class CheckIfEmptyBehavior (OneShotBehaviour):
3     class ProcessDataBehavior (OneShotBehaviour):
4     async def setup(self):
5         b = self.CheckIfEmptyBehavior()
6         self.add_behaviour(b)

```

Isječak koda 5: DatabaseAgent, struktura

```

1  class StateOne(State):
2      async def run(self):
3          print("I'm at state one (initial state)")
4          if not os.path.exists('average_prices.json'):
5              print("Prices file does not exist. Creating and populating...")
6              # Load data from index_auti.json
7              if self.create_local_file():
8                  self.agent.stop()
9          else:
10             self.set_next_state(STATE_TWO)

11     def create_local_file(self):
12         with open('Index_auti.json', 'r', encoding='utf-8') as index_file:
13             data = json.load(index_file)

14         average_prices = {}

15         for entry in data:
16             marka = entry.get('marka', 'undefined')
17             cijena = entry.get('cijena', 0)

18             # Exclude entries with a price of 0
19             if cijena > 0:
20                 if marka not in average_prices:
21                     average_prices[marka] = {'sum': cijena, 'count': 1}
22                 else:
23                     average_prices[marka]['sum'] += cijena
24                     average_prices[marka]['count'] += 1

25             # Calculate the average for each 'Marka'
26             for marka, values in average_prices.items():
27                 average_prices[marka]['average'] = values['sum'] / values['count']

28         average_prices["Undefined"] = average_prices.pop("")
29         # Save the average prices to a JSON file
30         json_filename = 'average_prices.json'
31         with open(json_filename, 'w', encoding='utf-8') as jsonfile:
32             json.dump(average_prices, jsonfile, indent=2, ensure_ascii=False)

33         print(f"Average prices saved to {json_filename}")
34         return True

```

Isječak koda 6: PriceAgent, StateOne

nju, agent čeka na pristigle poruke, posebno nove oglase koji sadrže informacije o cijenama automobila. Čim primi poruku, PriceAgent obrađuje informacije u JSON formatu, te izvršava dvije ključne funkcionalnosti.

Prvo, poziva metodu *check_for_prices(data)* kako bi provjerio nove cijene automobila u usporedbi s prosječnim cijenama. Ako pronade oglase s cijenama ispod prosjeka, stvara obavijesti o tim oglasima koje će kasnije poslati TelegramAgent-u. Ovaj proces potiče interakciju s dodatnim agentima *Buyer* i *Seller* za simulaciju pregovaranja. Ovi agenti objašnjeni su u sljedećem potpoglavlju, ali na kraju daju preporučenu cijenu koja se šalje zajedno s obavijestima TelegramAgent-u. Drugo, agent poziva metodu *textitupdate_average_prices(data)* kako bi ažurirao prosječne cijene na temelju novih podataka. Ovo uključuje provjeru i prilagodbu prosječnih cijena za svaku marku automobila. Nakon obrade primljenih podataka, PriceAgent postavlja sljedeće stanje na *StateThree*, koje označava završetak njegovog ciklusa. Detalji pojedinih funkcija vidljivi su na isječcima kôda 8 i 9.

Isječak koda kôda predstavlja treće i konačno stanje (*StateThree*) unutar PriceAgent-a. U ovom stanju, agent obavlja završne korake svog ciklusa.

Prvo, agent priprema listu obavijesti (*notification_list*) koja sadrži informacije o oglasima s cijenama ispod prosječne. Ove obavijesti generirane su u prethodnom stanju (*StateTwo*). Nakon toga, stvara se instanca agenta *TelegramAgent* koji je odgovoran za slanje obavijesti putem platforme Telegram. Agent komunicira s Telegramom putem posebne adrese *telegramAgent@localhost*. Sljedeći korak je pokretanje *TelegramAgent*-a pozivom *await telegram_agentstart()*. Zatim, stvara se poruka (*message*) koja sadrži listu obavijesti u JSON formatu, te se šalje *TelegramAgent*-u pozivom *await self.send(message)*. Važno je napomenuti kako se koristi *wait_until_finished(telegram_agent)* kako bi se pričekalo da *TelegramAgent* završi svoj proces prije nego što se *PriceAgent* zaustavi. Na kraju, poziva se *self.kill()* kako bi se zatvorio ciklus izvršavanja *PriceAgent*-a.

Isječak kôda 11 prikazuje strukturu *PriceAgent*-a

5.2.3.1. Buyer

BuyerAgent predstavlja potencijalnog kupca unutar sustava pregovaranja. Njegova uloga je pokušati postići što povoljniju cijenu od *SellerAgent*. Agent ima ključne parametre i ponašanja koja kontroliraju proces pregovaranja.

Isječak kôda 12 prikazuje definiciju klase *BuyerBehaviour*, koja je cikličko ponašanje unutar *BuyerAgent*. Ova klasa kontrolira ponašanje kupca tijekom procesa pregovaranja. *On_start* metoda Postavlja početne parametre pregovaranja poput maksimalne i minimalne prihvatljive cijene, početnog prijedloga te drugih varijabli. Oni uključuju:

- *max_price*: Maksimalna cijena koju je kupac spreman platiti.
- *min_price*: Minimalna cijena koju kupac postavlja kao početnu ponudu.
- *proposal*: Trenutna ponuda kupca.

```

1  async def check_for_prices(self, data):
2      with open('average_prices.json', 'r', encoding='utf-8') as jsonfile:
3          average_prices = json.load(jsonfile)
4      self.agent.notification_list = []
5      counter = 0
6      for entry in data:
7          marka = entry.get('marka', 'Undefined')
8          cijena = entry.get('cijena', 0)
9          marka = "Undefined" if marka == "" else marka

10         if cijena > 0:
11             if marka in average_prices:
12                 average_price = average_prices[marka]['average']
13                 if cijena < average_price:
14                     counter += 1
15                     buyer_jid = f"buyer_{counter}@localhost"
16                     seller_jid = f"seller_{counter}@localhost"
17                     buyerAgent = BuyerAgent(buyer_jid, "buyer", cijena, seller_jid)
18                     sellerAgent = SellerAgent(seller_jid, "seller", cijena,
19                                             ↪ buyer_jid)
20                     await buyerAgent.start()
21                     await sellerAgent.start()

22                 received_msg = await self.receive(timeout=10)
23                 if received_msg:
24                     print("Received message")
25                     negotiation_price = received_msg.body
26                     print(f"Post with marka '{marka}' has a price lower than
27                         ↪ average: {cijena} < {average_price}")
28                     notification_message = (
29                         f"Link: {entry['poveznica']}\n"
30                         f"Marka: {marka}\n"
31                         f"Cijena: {cijena}\n"
32                         f"Godina proizvodnje: {entry['godina_proizvodnje']}\n"
33                         f"-----\n"
34                         f"Opis: {entry['opis']}\n"
35                         f"-----\n"
36                         f"Prosječna cijena za {marka}: {int(average_price)}\n"
37                         f"Preporučena cijena za pregovaranje:
38                         ↪ {negotiation_price}\n"
39                     )
40                     self.agent.notification_list.append(notification_message)
41             else:
42                 print("No offers below average!")

```

Isječak koda 7: PriceAgent, funkcija *check_for_prices*

```

1  async def update_average_prices(self, new_data):
2      with open('average_prices.json', 'r', encoding='utf-8') as jsonfile:
3          average_prices = json.load(jsonfile)
4          undefined_sum = average_prices.get("Undefined", {"sum": 0})["sum"]
5          undefined_count = average_prices.get("Undefined", {"count": 0})["count"]
6      for entry in new_data:
7          marka = entry.get('marka', 'undefined')
8          cijena = entry.get('cijena', 0)
9          # Exclude entries with a price of 0
10         if cijena > 0:
11             if marka not in average_prices:
12                 average_prices[marka] = {'sum': cijena, 'count': 1}
13             else:
14                 average_prices[marka]['sum'] += cijena
15                 average_prices[marka]['count'] += 1
16             if marka == "":
17                 undefined_sum += cijena
18                 undefined_count += 1
19         undefined_sum += average_prices.get("Undefined", {"sum": 0})["sum"]
20         undefined_count += average_prices.get("Undefined", {"count": 0})["count"]
21     for marka, values in average_prices.items():
22         average_prices[marka]['average'] = values['sum'] / values['count']
23     if undefined_count > 0:
24         average_prices["Undefined"] = {
25             'sum': undefined_sum,
26             'count': undefined_count,
27             'average': undefined_sum / undefined_count
28         }
29     else:
30         # No entries with marka="", remove the "Undefined" key
31         average_prices.pop("Undefined", None)
32     average_prices.pop("", None)
33     json_filename = 'average_prices.json'
34     with open(json_filename, 'w', encoding='utf-8') as jsonfile:
35         json.dump(average_prices, jsonfile, indent=2, ensure_ascii=False)
36     print(f"Updated average prices saved to {json_filename}")

```

Isječak koda 8: PriceAgent, funkcija *update_average_prices*

```

1  async def run(self):
2      print("I'm at state three (final state)")
3      notification_list = self.agent.notification_list
4      telegram_agent = TelegramAgent("telegramAgent@localhost", "telegram")
5      await telegram_agent.start()
6      message = spade.message.Message(to=str(telegram_agent.jid))
7      message.body = json.dumps(notification_list)
8      await self.send(message)
9      await wait_until_finished(telegram_agent)
10     self.kill()

```

Isječak koda 9: PriceAgent, StateThree

```

1 class PriceFSMBehaviour(FSMBehaviour):
2     async def on_start(self):
3         print(f"FSM starting at initial state {self.current_state}")
4
5     async def on_end(self):
6         print(f"FSM finished at state {self.current_state}")
7         await self.agent.stop()
8
9 class StateOne(State):
10 class StateTwo(State):
11 class StateThree(State):
12
13 class PriceAgent(Agent):
14     async def setup(self):
15         fsm = PriceFSMBehaviour()
16         fsm.add_state(name=STATE_ONE, state=StateOne(), initial=True)
17         fsm.add_state(name=STATE_TWO, state=StateTwo())
18         fsm.add_state(name=STATE_THREE, state=StateThree())
19         fsm.add_transition(source=STATE_ONE, dest=STATE_TWO)
20         fsm.add_transition(source=STATE_TWO, dest=STATE_THREE)
21         fsm.add_transition(source=STATE_THREE, dest=STATE_TWO)
22         self.add_behaviour(fsm)

```

Isječak koda 10: PriceAgent, PriceFSMBehaviour, and States

```

1 class BuyerBehaviour(CyclicBehaviour):
2     async def on_start(self):
3         await asyncio.sleep(0.5)
4         print(f"{self.agent.jid} agent started!\n")
5         self.max_price = self.agent.price
6         self.min_price = int(6/10 * self.max_price)
7         self.proposal = random.uniform(self.min_price, self.max_price)
8         self.round = 1
9         self.accepted_offer = False
10        self.minimal_acceptance_price = self.min_price + (self.max_price - self.min_price) *
11        ↪ 0.1
12        self.acceptance_threshold = self.minimal_acceptance_price
13        self.concession_factor = 0.1 # Concession factor for adjusting offers
14        self.feedback = []

```

Isječak koda 11: Definicija klase *BuyerBehaviour*

- round: Brojač runde pregovora.
- accepted_offer: Logička varijabla koja označava je li ponuda prihvaćena.
- minimal_acceptance_price: Minimalna prihvatljiva cijena za kupca.
- acceptance_threshold: Prag prihvaćanja, smanjuje se tijekom pregovora.
- concession_factor: Faktor koncesije za prilagodbu ponuda.
- feedback: Lista koja bilježi tijek pregovora.

Isječak koda 13 prikazuje *run* metodu u okviru *BuyerBehaviour* ponašanja unutar *BuyerAgent* klase. Ova metoda opisuje ponašanje kupca tijekom pregovora s agentom prodavatelja (*seller*). Metoda *run* u *BuyerBehaviour* ponašanju *BuyerAgent* klase ima sljedeći tok izvršavanja:

1. Ako ponuda kupca još nije prihvaćena, agent čeka da prodavatelj započne pregovore.
2. Kupac šalje inicijalnu ponudu prodavatelju putem *Message* objekta s trenutnom ponudom *proposal* kao tijelom poruke.
3. Agent čeka na primanje poruke od prodavatelja unutar određenog vremenskog ograničenja.
4. Ako je primljena poruka od prodavatelja:
 - (a) Primljena ponuda prodavatelja pretvara se u decimalni broj.
 - (b) Ispisuje se trenutna ponuda kupca i runda pregovora.
 - (c) Provjerava se je li primljena ponuda ispod minimalno prihvatljive cijene.
 - (d) Ako je, kupac prihvaća ponudu, šalje odobrenu cijenu *PriceAgent*-u i zaustavlja se.
 - (e) Inače, prilagođava prag prihvaćanja na temelju tijeka pregovora.
 - (f) Prilagođava trenutnu ponudu primljenom prijedlogu i primjenjuje faktor koncesije.
 - (g) Ponuda se šalje prodavatelju, a detalji pregovora dodaju se u povratne informacije (feedback) i povećava se broj runde.
5. Ako nije primljena poruka od prodavatelja, ispisuje se da ništa nije primljeno.

5.2.3.2. Seller

SellerAgent predstavlja prodavatelja unutar sustava pregovaranja. Glavna uloga ovog agenta je postizanje što povoljnije cijene od strane *BuyerAgent*-a. Ključni parametri i ponašanja kontroliraju proces pregovaranja.

Isječak kôda 14 prikazuje definiciju klase *SellerBehaviour*, cikličkog ponašanja unutar *SellerAgent*-a. Ovo ponašanje određuje kako će se prodavatelj ponašati tijekom pregovora.

On_start metoda postavlja početne parametre pregovaranja, uključujući:

- *max_price*: Maksimalna cijena koju prodavatelj može postaviti.
- *min_price*: Minimalna prihvatljiva cijena za prodavatelja, postavljena na 60
- *proposal*: Početni prijedlog cijene koji se generira slučajnim odabirom unutar raspona između minimalne i maksimalne cijene.
- *round*: Brojač runde pregovora.

Isječak koda 15 prikazuje *run* metodu unutar *SellerBehaviour* ponašanja *SellerAgent*-a. Ova metoda opisuje ponašanje prodavatelja tijekom pregovora s *BuyerAgent*-om.

Run metoda *SellerBehaviour* ponašanja *SellerAgent*-a ima sljedeći tok izvršavanja:

```

1  async def run(self):
2      # Wait for the seller agent to start
3      if not self.accepted_offer:
4          msg = Message(to=self.agent.seller_jid)
5          msg.body = str(self.proposal)
6          await self.send(msg)
7
8          received_msg = await self.receive(timeout=20)
9
10         if received_msg:
11             received_proposal = float(received_msg.body)
12             print(f"{self.agent.jid} (Round {self.round}): {self.proposal}")
13
14             # Check if the received offer is below the minimal acceptance
15             ⇨ price
16             if received_proposal <= self.minimal_acceptance_price:
17                 print(f"{self.agent.jid} (Round {self.round}): Accepted the
18                 ⇨ offer! (Minimal acceptance price reached)")
19                 self.accepted_offer = True
20                 self.accepted_proposal = received_proposal
21                 msg = Message(to="price@localhost")
22                 msg.body = str(int(self.accepted_proposal))
23                 await self.send(msg)
24
25                 await self.agent.stop()
26             else:
27                 # Adjust acceptance threshold based on negotiation progress
28                 acceptance_threshold_decay = 0.95
29                 self.acceptance_threshold *= acceptance_threshold_decay
30
31                 # Apply concession strategy
32                 self.proposal = max(received_proposal, self.min_price)
33                 self.proposal *= (1 + self.concession_factor)
34                 self.proposal = min(self.proposal, self.max_price)
35
36                 print(f"{self.agent.jid} (Round {self.round}):
37                 ⇨ Counteroffering with {self.proposal}\n")
38                 msg = Message(to=self.agent.seller_jid)
39                 msg.body = str(self.proposal)
40                 await self.send(msg)
41                 self.feedback.append((self.round, received_proposal,
42                 ⇨ self.proposal))
43                 self.round += 1
44             else:
45                 print(f"{self.agent.jid} (Round {self.round}): Didn't receive
46                 ⇨ anything")

```

Isječak koda 12: Buyer, run metoda

```

1  class SellerBehaviour(CyclicBehaviour):
2  async def on_start(self):
3      print(f"{self.agent.jid} agent started!\n")
4      self.max_price = self.agent.price
5      self.min_price = float(6/10 * self.max_price)
6      self.proposal = random.uniform(self.min_price, self.max_price)
7      self.round = 1

```

Isječak koda 13: Definicija klase *SellerBehaviour*

```

1  async def run(self):
2      # Wait for the buyer agent to start
3      received_msg = await self.receive(timeout=20)
4      if received_msg:
5          print(f"{self.agent.jid} (Round {self.round}): My proposal {self.proposal}!")

6          # Apply concession strategy
7          self.proposal *= 0.9
8          self.proposal = max(self.min_price, self.proposal)

9          print(f"{self.agent.jid} (Round {self.round}): Counteroffering with
    ↪ {self.proposal}\n")
10         msg = Message(to=self.agent.buyer_jid)
11         msg.body = str(self.proposal)
12         await self.send(msg)
13     else:
14         await self.agent.stop()
15     self.round += 1

```

Isječak koda 14: Seller, run metoda

1. Čeka da *BuyerAgent* započne pregovore.
2. Čeka na primanje poruke od *BuyerAgent*-a unutar određenog vremenskog ograničenja.
3. Ako je primljena poruka od *BuyerAgent*-a:
 - (a) Ispisuje trenutni prijedlog cijene prodavatelja i rundu pregovora.
 - (b) Primenjuje strategiju koncesije na trenutni prijedlog cijene.
 - (c) Šalje kontra ponudu s prilagođenom cijenom *BuyerAgent*-u.
4. Ako nije primljena poruka od *BuyerAgent*-a, agent završava s radom.
5. Povećava broj runde.

5.2.4. TelegramAgent

TelegramAgent predstavlja agenta odgovornog za slanje obavijesti putem Telegrama unutar sustava pregovaranja. Ključni parametri i ponašanja kontroliraju proces slanja obavijesti putem Telegrama.

Isječak kôda 16 prikazuje definiciju klase *TelegramAgentBehavior*, jednokratnog ponašanja unutar *TelegramAgent*-a. Ovo ponašanje određuje kako će agent Telegrama reagirati prilikom pokretanja i tijekom izvršavanja.

On_start metoda postavlja početne parametre Telegram agenta, uključujući:

- `bot_token`: Token za identifikaciju Telegram bota.
- `chat_id`: Identifikacija chat-a na Telegramu gdje će se slati obavijesti.

```

1 class TelegramAgentBehavior(OneShotBehaviour):
2     async def on_start(self):
3         print(f"Telegram agent is starting...")
4         self.bot_token = ''
5         self.chat_id = ''
6     async def send_telegram_notification(self, bot_token, chat_id, message):
7         bot = Bot(token=bot_token)
8         await bot.send_message(chat_id=chat_id, text=message)
9         print("Notification sent to Telegram!")
10
11 async def run(self):
12     print("run the bot")
13     msg = await self.receive(timeout=10)
14     if msg:
15         notification_list = json.loads(msg.body)
16         for entry in notification_list:
17             message = f"Pozdrav, ovaj oglas mogao bi te zanimati\n{entry}"
18             await self.send_telegram_notification(self.bot_token, self.chat_id,
19                 ↪ message)

```

Isječak koda 15: Definicija klase *TelegramAgentBehavior*

Ovaj agent Telegrama ima sljedeći tok izvršavanja:

1. Na pokretanju agenta, postavlja se *bot_token* i *chat_id*.
2. Pokreće metodu *run*, koja čeka primanje poruke unutar određenog vremenskog ograničenja.
3. Ako je primljena poruka, pretvara je u listu obavijesti i šalje ih korisniku putem Telegrama.

Ispis obavijesti u telegram aplikaciji vidljiv je na slici 4.

index_oglasi_bot

bot

mali gradski potrosac

Prosječna cijena za Citroen: 4009

Preporučena cijena za pregovaranje: 1299

www.index.hr/oglasi

Citroen C3 1.4HDI 2006godina | INDEX OGLASI

240tkm org km 2006godina Reg 18.06.2024 1.4 hdi
50kw Uredan mali gradski potrosac



21:48

Pozdrav, ovaj oglas mogao bi te zanimati

Link: <https://www.index.hr/oglasi/vw-passat-variant-1-9-tdi/oid/4495014>

Marka: VW

Cijena: 3500

Godina proizvodnje: 2006

Opis: Passat variant 2006 g1.9tdi 77kwDugogodišnji vlasnikDva kljuca Reg 6/23Mehanički topLimarija nije cvijeće Cijena se da malo iskorigirat kad se auto vidi

Prosječna cijena za VW: 9445

Preporučena cijena za pregovaranje: 2743

www.index.hr/oglasi

VW Passat Variant 1.9 tdi | INDEX OGLASI

Passat variant 2006 g 1.9tdi 77kw Dugogodišnji vlasnik Dva kljuca Reg 6/23 Mehanički top
Limarija nije cvijeće Cijena se da malo iskorigir...



21:48

Slika 4: Primjer primljenih obavijesti

6. Zaključak

U projektu prikazani su ključne aspekti implementiranog višeagentnog sustava za interakciju s oglasima automobila preuzetim s web stranice *Index Oglasi*. Kroz rad istražene su i implementirane funkcionalnosti koje omogućuju automatsko preuzimanje, analizu cijena, te simulacija pregovaranja između kupca i prodavatelja unutar sustava.

Implementirani višeagentni sustav koristi platformu *Spade* za izradu agenata, a interakcija s korisnikom ostvarena je putem *Telegram* sučelja. Kroz primjenu web scrapinga, analize cijena te mehanizama pregovaranja, omogućeno je korisnicima praćenje oglasa i primanje obavijesti o potencijalno zanimljivim vozilima.

Sustav je pokazao funkcionalnost u praćenju i pregovaranju s obzirom na promjene cijena, a implementirano Telegram sučelje omogućava korisnicima intuitivan pristup informacijama. Unatoč ostvarenim funkcionalnostima, daljnji rad na sustavu može obuhvatiti proširenje mogućnosti analize, uvođenje dodatnih kriterija za praćenje oglasa te poboljšanje mehanizama pregovaranja kako bi se povećala fleksibilnost i prilagodljivost sustava različitim korisničkim preferencijama.

Popis literature

- [1] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2. izdanje. Glasgow, UK: John Wiley & Sons Ltd., 2009., 484 str., ISBN: 978-0-470-51946-2.
- [2] H. Donancio, A. Casals i A. A. F. Brandão, „Exposing agents as web services: a case study using JADE and SPADE,” 2019.
- [3] L. Holborow, *Mind*, sv. 81, br. 323, str. 458–468, 1972., ISSN: 00264423, 14602113.
- [4] zkardija, *python-web-scraping-primjeri*, <https://github.com/zkradija/python-web-scraping-primjeri>, Datum pristupa: 10. 01. 2024, 2023.

Popis slika

1.	Stranica s koje se preuzimaju oglasi	6
2.	Dijagram arhitekture rješenja	8
3.	Dijagram slijeda	9
4.	Primjer primljenih obavijesti	24

Popis isječaka koda

1.	TopLevelAgent, inicijalizacija i run petlja	10
2.	TopLevelAgent, glavna funkcija usporedbe	11
3.	DatabaseAgent, Ponašanje <i>CheckIfEmptyBehaviour</i>	13
4.	DatabaseAgent, Ponašanje <i>ProcessDataBehavior</i>	14
5.	DatabaseAgent, struktura	14
6.	PriceAgent, StateOne	15
7.	PriceAgent, funkcija <i>check_for_prices</i>	17
8.	PriceAgent, funkcija <i>update_average_prices</i>	18
9.	PriceAgent, StateThree	18
10.	PriceAgent, PriceFSMBehaviour, and States	19
11.	Definicija klase <i>BuyerBehaviour</i>	19
12.	Buyer, run metoda	21
13.	Definicija klase <i>SellerBehaviour</i>	21
14.	Seller, run metoda	22
15.	Definicija klase <i>TelegramAgentBehavior</i>	23