SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Praćenje aktivnosti na društvenim mrežama pomoću reaktivnih agenata

Seminarski rad

Nikola Majcen
br. indeksa: 44445/15-R

Mentor:
Doc. dr. sc. Markus Schatten

Varaždin, 17. siječnja 2017.

Sadržaj

1 Uvod **1**

2 Opis zadatka **2**
2.1 Koncept zadatka 2
2.2 Višeagentni sustavi i primjena u društvenim mrežama 3

3 Implementacija rješenja **6**
3.1 Implementacija Facebook agenta 6
3.1.1 Prikaz implementacije modela 6
3.1.2 Prikaz implementacije API-a 8
3.1.3 Prikaz implementacije reaktivnog agenta 10
3.2 Implementacija Twitter agenta 13
3.2.1 Prikaz implementacije modela 14
3.2.2 Prikaz implementacije API-a 15
3.2.3 Prikaz implementacije reaktivnog agenta 16
3.3 Implementacija agenta za izvještaje 20
3.4 Implementacija kreiranja agenata 23

4 Demonstracija rješenja **28**

5 Zaključak **33**

Bibliografija **33**

Poglavlje 1

Uvod

Tema ovog seminarskog rada je praćenje aktivnosti na društvenim mrežama pomoću reaktivnih agenata, odnosno izrada jednostavne aplikacije gdje se pomoću agenata mogu pratiti objave na društvenim mrežama u realnom vremenu.

Seminarski rad uključuje teorijsku obradu agenata, ali praktični prikaz i implementaciju rješenja koje je vezano za praćenje aktivnosti na društvenim mrežama Twitter i Facebook. Praćenje na društvenim mrežama uključuje praćenje statusa i/ili hashtag-ova. Rad je podjeljen na cjeline i to:

- Opis zadatka
- Implementacija rješenja
- Demonstracija rješenja

Poglavlje 2

Opis zadatka

Tema ovog seminarskog rada jest izrada aplikacije koja korisniku omogućuje da pretražuje određene društvene mreže preko hashtag-ova ili statusa, na način da se na iste pretplati te obrada literature vezane za reaktivne agente koji se koriste u samoj aplikaciji.

2.1 Koncept zadatka

Da bi korisnik mogao pretraživati određene društvene mreže, za početak je potrebno pretpostaviti koji će biti elementi pretrage i kako će se moći podaci pratiti.

U ovom zadatku će se koristiti društvene mreže Facebook i Twitter, gdje će se kod svake od njih pratiti određena aktivnost korisnika. Zbog određenih mjera privatnosti, nije bilo moguće izvesti iste funkcionalnosti za svaku od društvenih mreža, pa je shodno tome omogućeno sljedeće; za društvenu mrežu Facebook omogućeno je:

- Praćenje statusa korisnika čiji su profili javni

dok je za društvenu mrežu Twitter omogućeno sljedeće:

- Praćenje statusa (tweet) svih korisnika (korisnički profil ne mora biti javan)
- Praćenje statusa (tweet) koji sadrže određeni hashtag

Pretplata na promjenu statusa ili praćenje statusa sa određenim hashtag-om odvija se na način da korisnik definira vrijeme u sekundama u kojim će intervalima i kolko dugo pratiti promjene na određenoj društvenoj mreži. Detaljnije informacije o implementaciji zadatka i potrebnim dodatnih podacima za spajanje na određene društvene mreže nalaze se u sljedećem poglavlju.

Za implementaciju rješenja ovog zadatka koristio sam višeagentni sustav koji se sastoji od dva, odnosno tri reaktivna agenta. Dva reaktivna agenta omogućuju pretraživanje i praćenje promjena na društvenim mrežama, a skupljene podatke šalju trećem agentu koji dohvaća sve promjene svih agenata. Za svaku društvenu mrežu definiran je posebni agent koji koristi specifične načine dohvata podataka za svaku od društvenih mreža. Nakon što određeni agent za dohvaćanje podataka s društvene mreže završi s radom, treći agent (koji sakuplja podatke) šalje izvještaj u obliku svih sakupljenih podataka koje mu je taj agent poslao. Na taj način završava praćenje i izvještavanje za tog agenta, dok svi ostali nastavljaju s radom.

2.2 Višeagentni sustavi i primjena u društvenim mrežama

U današnje vrijeme, društvene mreže su svuda oko nas. Gotovo da i ne postoji osoba koja nije barem jednom koristila društvenu mrežu te imala određene aktivnosti na istoj. Zbog velike količine podataka na društvenim mrežama, ponekad je vrlo teško pratiti sadržaj koji se kreće po navedenim mrežama.

Sve društvene mreže dijele četiri karakteristike (Lerman, 2008):

- Korisnici kreiraju ili doprinose određenom sadržaju
- Korisnici bilježe sadržaj sa oznakama (engl.tag)
- Korisnici ocjenjuju ili komentiraju navedeni sadržaj
- Korisnici stvaraju veze sa drugim poznanicima ili osobama istih interesa

Radi kreiranja tolike količine sadržaja, ponekad je vrlo teško uopće primijetiti pojedini sadržaj na mreži, jer zbog manjka komentara ili ocjena, sadržaj pojedinog korisnika vrlo brzo odlazi sa glavne stranice (engl. front page) ili određenog dijela stranice gdje se nalazi popularan sadržaj.

Većina društvenih mreža funkcionira upravo na takav način (Facebook, Twitter, LinkedIn, Digg...), a upravo se kao jedan od primjera ovdje također spominje Digg (Lerman, 2008). Da bi se to izbjeglo, kreiraju se razni algoritmi kako bi sadržaj ostao vidljiv korisnicima koji zapravo žele doprijeti do njega te ga vidjeti, a jedan od primjera kako je to implementirano je Facebook, gdje je moguće postaviti različite vrste prikaza novih vijesti odnosno aktivnosti korisnika.

Ipak, u ovome slučaju, cilj nije oponašanje korisnika niti evaluacija algoritama kako društvene mreže funkcioniraju, već je cilj vidjeti na koji način je moguće filtriranje ali i agregacija pojedinih sadržaja društvenih mreža. Ranije je već spomenuto kako društvene mreže generiraju veliku količinu sadržaja, a upravo taj sadržaj postaje zanimljiv ukoliko je moguće kreirati sustav koji može pratiti i filtrirati podatke te generirati određene statistike pa i informacije iz obrađenih podataka.

Jedan od načina na koji je moguće pratiti izvore podataka te ih obrađivati jest korištenje višeagentnih sustava. Višeagentni sustav predstavlja niz agenata koji su u međusobnoj interakciji, a definiranjem agenata, oni će raditi u skladu s ciljevima i zadacima na temelju kojih su definirani (Schatten, 2016).

Sljedeći prethodno navedeno, jedan od načina za takvo prikupljanje podataka te njihovu analizu jest kompozicijsko modeliranje višeagentnih sustava (engl. Compositional Modelling of Multi-Agent Systems) (Catholijn M. Jonker, 2002). Ovaj princip sastoji od sljedeća tri elementa:

- kompozicije procesa,
- kompozicije znanja,
- veze između kompozicije procesa i kompozicije znanja.

Kompozicija procesa identificira relevantne procese za različite razine apstrakcije te prikazuje kako proces može biti definiran pomoću procesa niže razine.

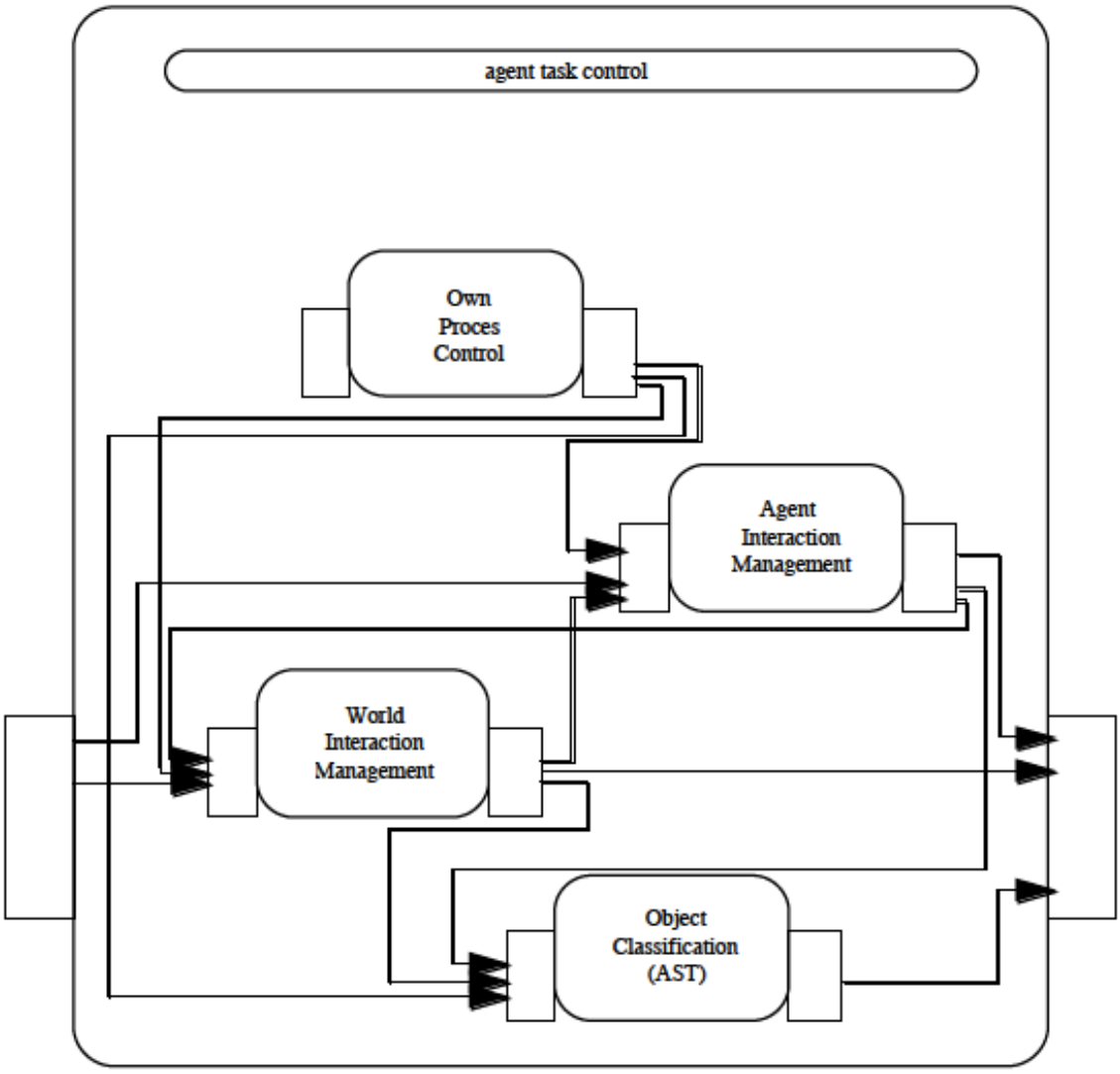
Kompozicija znanja identificira strukture znanja za različite razine apstrakcije te definira kako se navedena struktura znanja može definirati pomoću znanja niže razine. Osnovni građevni blokovi strukture znanja su informacijski tipovi i baze znanja, gdje informacijski tipovi definiraju ontologiju pomoću kojih se definiraju objekti, dok baze znanja definiraju znanje koje se koristi u jednom ili više procesa.

Veza između kompozicije procesa i znanja je vidljiva u tome što svaki proces koristi strukture znanja za određenu akciju odnosno aktivnost.

Na temelju navedenog, definiraju se moduli koji sadrže određena svojstva, i to svojstva ponašanja i okoline. Svojstva ponašanja se nalaze na izlazu komponente te mogu imati definirane uvjete koji jesu ili nisu ispunjeni, ovisno o ulazu, te na temelju toga kreiraju određen izlaz. S druge strane, svostva okoline definirana su na ulazu komponente za dohvat određenih parametara koji se kasnije procesuiraju (Catholijn M. Jonker, 2002).

S ovim pristupom dolazi se do ideje da se sustav podijeli na komponente, odnosno da se definiraju zasebni agenti koji rade samo određenu akciju, gdje se na ulazu dobiju određeni podaci koji se kasnije procesuiraju, a na kraju agent, u ovom slučaju kao komponenta sustava, na izlazu daje rezultat kao ulaz sljedećem agentu.

Da bi svaki pojedini agent mogao izvršavati svoje zadatke, isti se mora sastojati od četiri osnovne komponente koje se mogu vidjeti na slici 2.1.



Slika 2.1: Kompozicija odnosa unutar agenta (Catholijn M. Jonker, 2002)

Klasifikacija objekta (engl. Object Classification) je komponenta koja definira specifičnu akciju agenta. Upravljanje interakcijom sa svijetom (engl. World Interaction Management) je komponenta koja definira kako će agent biti u interakciji sa vanjskim svijetom. Upravljanje interakcijom agenta (engl. Agent Interaction Management) definira kako će agent komunicirati sa ostalim agentima u sustavu, dok

kontrola vlastitog procesa (engl. Own Process Control) definira ponašanje i svojstva samog agenta. Agent u smislu ponašanja može biti proaktivan ili reaktivan.

Proaktivno ponašanje agenta ima sljedeće karakteristike:

- motrenje,
- komuniciranje o opažanju s drugim agentima,
- traženje rezultata motrenja od drugih agenata,
- definiranje zaključaka o klasifikaciji objekata

dok reaktivno ponašanje agenta u smislu odgovora na zahtjev definira sljedeće:

- komuniciranje o rezultatima motrenja u čim kraćem vremenu (čim su rezultati dosupni),
- početak praćenja drugih agenata.

Prema navedenom, definiran je zadatak koji će pokušati na jednostavan način prikazati kako se uz korištenje društvenih mreža i velike količine podataka, uz kreiranje reaktivnih agenata može jednostavno prikupiti ciljana skupina podataka uz njihovu suradnju i podjelu na manje procese te filtiranje rezultata. Određeni agent osim karakteristika reaktivnih agenata, sadržavat će pojedinu karakteristiku proaktivnog agenta, međutim, neće nikako sadržavati sve karakteristike koje opisuju proaktivne agente.

U ovom radu će agenti biti definirani kao reaktivni agenti, odnosno agenti odlučuju o svojim akcijama bez razmatranja prošlosti tj. donose odluke na temelju trenutnog stanja okruženja (Schatten, 2016), te će u interakciji između sebe dohvaćati podatke iz vanjskog svijeta (društvene mreže), te iste dobavljati glavnom agentu (izvjestitelju).

Poglavlje 3

Implementacija rješenja

U ovom poglavlju prikazat ću implementaciju rješenja za praćenje na društvenim mrežama. Za implementaciju je korišten programski jezik Python verzije 2.7. Kao biblioteka koja omogućuje rad s agentima korišten je SPADE.

Za početak rada, potrebno je u izvornom direktoriju otvoriti datoteku *config.json* te unutar nje definirati naziv i lozinku agenta izvjestitelja. Zadano ime agenta je *reporter@127.0.0.1*, dok je lozinka *secret*.

Uz tu datoteku, postoji i datoteka *credentials-sample.json*, koja predstavlja podatke koji su potrebni da bi se aplikacija mogla povezati sa određenom društvenom mrežom. Detaljnije informacije za unos podataka za svaku od društvenih mreža bit će prikazane u zasebnom poglavlju za svakog agenta posebno, ali prije korištenja potrebno je istu preimenovati u *credentials.json*.

3.1 Implementacija Facebook agenta

Prije detaljnog objašnjavanja Facebook agenta, potrebno je prvo definirati potrebne podatke kako bi se aplikacija mogla povezati sa Facebook servisima. Za to je potrebno otvoriti datoteku *credentials.json* te unutar objekta `facebook` za ključ `access_token` unijeti vrijednost tokena koja se dobije kada se kreira nova aplikacija unutar Facebook-ovog sučelja za razvijatelje aplikacija. Bez vrijednosti `access_token`-a nije moguće koristiti Facebook agenta.

Prikaz implementacije podijeljen je na tri dijela, a prikazuju sljedeće:

- implementaciju modela,
- implementaciju API-a
- implementaciju reaktivnog agenta

3.1.1 Prikaz implementacije modela

Kako bi se dohvatili podaci za spajanje na Facebook-ov servis, kreiran je model *FacebookCredentials*, a sadrži sljedeće:

```
#! /usr/bin/usr python
# -*- coding: utf-8 -*-
```

```
import json


class FacebookCredentials:

    def __init__(self, filename):
        self.filename = filename

    def get_credentials(self):
        json_data = self.__read_configuration_file()
        api_key = json_data["facebook"]["access_token"]
        return api_key

    def __read_configuration_file(self):
        file = open(self.filename, "r")
        data = json.load(file)
        file.close()
        return data
```

Model sadrži dvije metode; `get_credentials()` za dohvat podataka za spajanje na servis, te `__read_configuration_file()` koja omogućuje čitanje datoteke u kojem se nalaze podaci za spajanje na servis.

Sljedeći modeli su *FacebookUser* i *FacebookStatus*:

```
# ! /usr/bin/env python
# -*- coding: utf-8 -*-

from datetime import datetime


class FacebookUser:

    def __init__(self, data):
        self.id = data["id"]
        self.username = data["username"]
        self.name = data["name"]
        city = data["location"]["city"]
        country = data["location"]["country"]
        state = data["location"]["state"]
        self.location = "{}, {}, {}".format(city, country, state)


class FacebookStatus:

    def __init__(self, data):
        self.name = None
```

```
        self.username = None
        self.status_type = data["status_type"]
        self.date = datetime.strptime(data["updated_time"],
                                      "%Y-%m-%dT%H:%M:%S+0000")
        if "message" in data:
            self.message = data["message"].strip()
        else:
            self.message = ""

        if "link" in data:
            self.link = data["link"]
        else:
            self.link = ""
```

Klasa *FacebookUser* definira osnovne informacije o korisniku, kao što je njegov id, korisničko ime, ime i prezime te lokacija. Podaci se dobivaju na način da se preko konstruktora prislijedi objekt koji sadrži navedene podatke te se oni otpakiraju na način da se proslijedi ključ, a bit će vraćena vrijednost za navedeni ključ. Klasa *FacebookStatus* definira osnovne informacije o statusu, a sadrži ime i prezime korisnika koji je objavio status, korisničko ime, tip statusa, vrijeme objave, te ovisno o tome da li se radi o tekstualnom statusu ili slici poruku odnosno poveznicu. Način dobivanja podataka je identičan kao i za klasu *FacebookUser*.

3.1.2 Prikaz implementacije API-a

Sljedeće što sam kreirao je Facebook API koji zapravo definira pozive metoda koje se spajaju na Facebook-ov službeni API te na temelju tih poziva se dobivaju određeni podaci koje korisnik traži.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import json

import facebook

from social.facebook.facebook_models import *


class FacebookAPI:

    def __init__(self, access_token):
        self.access_token = access_token

    def search_posts(self, username):
        user_id, user_username, user_name = self.__fetch_user_details(username)
        if user_id is None:
            print "Token is expired."
            return None
```

```
        graph = facebook.GraphAPI(access_token=self.access_token)
        batched_requests = '[{"method": "GET", "relative_url": "'
                           + user_id + '/feed"}]'

        try:
            posts_data = graph.request("", post_args={"batch": batched_requests})
        except:
            print "Token has expired."

        for key, value in posts_data[0].items():
            if key == "body":
                json_posts = json.loads(value)["data"]
                break

        posts = []
        for json_post in json_posts:
            post = FacebookStatus(json_post)
            post.username = user_username
            post.name = user_name
            posts.append(post)
        return posts

    def __fetch_user_details(self, username):
        graph = facebook.GraphAPI(access_token=self.access_token)
        batched_requests = '[ {"method": "GET", "relative_url": "'
                           + username + '"} ]'
        try:
            user_data = graph.request("", post_args={"batch": batched_requests})
        except:
            return None

        if user_data is None:
            return None

        for key, value in user_data[0].items():
            user = FacebookUser(json.loads(value))
            break
        return user.id, user.username, user.name
```

Klasa *FacebookAPI* definira samo jedan poziv, a to je dohvaćanje novih statusa. Taj poziv izvršava se preko metode `search_posts(username)`. Ova metoda na temelju korisničkog id-a traži nove statuse ili slike korisnika, te iste dohvaća i vraća kao rezultat. Prethodno navedena metoda mora obavezno imati id korisničkog računa potrebnog za dohvaćanje podataka, a kako bi se isti dobio, potrebno je pomoću metode `__fetch_user_details(username)` dohvatiti korisnikov id.

Svi pozivi dohvaćanja podataka vrše se preko Facebook-ovog sustava GraphAPI.

3.1.3 Prikaz implementacije reaktivnog agenta

Ovdje ću prikazati konačnu implementaciju agenta koji na Facebook društvenoj mreži dohvaća promjene u smislu novih statusa koji uključuju tekst ili sliku/e.

```
#! /usr/bin/usr python
# -*- coding: utf-8 -*-

import json
from datetime import datetime, timedelta

from spade import Agent
from spade import Behaviour
from spade import ACLMessage
from spade import AID

from social.facebook.facebook_api import FacebookAPI
from social.facebook.facebook_credentials import FacebookCredentials
from social.shared.report_message import ReportMessage


class FacebookAgent(Agent.Agent):

    class FetchBehaviour(Behaviour.PeriodicBehaviour):

        def __init__(self, time, period, credentials_filename):
            Behaviour.PeriodicBehaviour.__init__(self, period)
            self.periods = time / period
            self.current_date = datetime.now()
            facebook_credentials = FacebookCredentials(credentials_filename)
            self.facebook_api = FacebookAPI(facebook_credentials
                                            .get_credentials())

        def onStart(self):
            print "[" + self.myAgent.getName()
                + "] Status activity fetch started."
            self.__send_registration_message()

        def onEnd(self):
            print "[" + self.myAgent.getName()
                + "] Status activity fetch ended."
            self.__send_report_message()

        def _onTick(self):
```

```
            if self.periods > 0:
                self.periods -= 1
                self.fetch_data()
            else:
                self.kill()

        def fetch_data(self):
            print "[" + self.myAgent.getName()
                + "] Fetching statuses... (No. of periods left: "
                + str(self.periods) + ")"
            results = self.facebook_api.search_posts(self.myAgent.keyword)

            statuses_found = 0
            for status in results:
                if (status.date - self.current_date) > timedelta(seconds=1):
                    self.current_date = status.date
                    self.__send_status_message(status)
                    statuses_found += 1
            print "[" + self.myAgent.getName()
                + "] Found statuses: " + str(statuses_found)

        def __send_registration_message(self):
            message = ACLMessage.ACLMessage()
            message.addReceiver(self.myAgent.receiver)
            message.setOntology("register")
            message.setContent(self.myAgent.getName())
            self.myAgent.send(message)

        def __send_status_message(self, post):
            message = ACLMessage.ACLMessage()
            message.addReceiver(self.myAgent.receiver)
            message.setOntology("notify")
            object = ReportMessage("Facebook",
                                   post.status_type,
                                   self.myAgent.keyword,
                                   post.username,
                                   post.name, post.date,
                                   post.message + " (" + post.link + ")")
            value = json.dumps(object.__dict__)
            message.setContent(value)
            self.myAgent.send(message)

        def __send_report_message(self):
            message = ACLMessage.ACLMessage()
            message.addReceiver(self.myAgent.receiver)
```

```
            message.setOntology("report")
            message.setContent(self.myAgent.getName())
            self.myAgent.send(message)

    class ReportDeliveryBehaviour(Behaviour.EventBehaviour):

        def _process(self):
            received_message = self._receive()
            if received_message:
                content = json.loads(received_message.getContent())
                print ""
                print "[" + self.myAgent.getName()
                    + "] Received message from: "
                    + received_message.getSender().getName()
                print "[" + self.myAgent.getName()
                    + "] Total number of fetched data: "
                    + str(len(content))
                print ""
                for element in content:
                    notify_message = ReportMessage()
                    notify_message.load_json(element)
                    notify_message.print_message()
            self.myAgent.stop()

    def __init__(self, reporter_name, keyword,
                 credentials_filename="credentials.json",
                 time=60, period=10):
        agent_id, password = self.__generate_agent_credentials()
        Agent.Agent.__init__(self, agent_id, password)
        self.keyword = keyword
        self.credentials_filename = credentials_filename
        self.time = time
        self.period = period
        self.receiver = AID.aid(name=reporter_name,
                                addresses=["xmpp://" + reporter_name])

    def _setup(self):
        print "[" + self.getName() + "] Facebook fetch agent is starting..."
        fetch_behaviour = self.FetchBehaviour(self.time,
                                              self.period,
                                              self.credentials_filename)
        self.addBehaviour(fetch_behaviour)
        delivery_template = Behaviour.ACLTemplate()
        delivery_template.setOntology("report_delivery")
        self.addBehaviour(self.ReportDeliveryBehaviour(), delivery_template)
```

```
def __generate_agent_credentials(self):
    agent_name = "facebook_" + datetime.now().strftime("%H%M%S")
                 + "@127.0.0.1"
    return agent_name, "secret"
```

Agent ima dvije podklase koje definiraju ponašanje agenta, a to su *FetchBehaviour* i *ReportDeliveryBehaviour*.

`FetchBehaviour` je ponašanje koje je definirano kao `PeriodicBehaviour` a ono funkcionira na način da u određenim vremenskim intervalima pokreće metodu `_onTick()`. Vrijeme između dva pokretanja navedene metode definira se putem konstruktora klase `PeriodicBehaviour` (G. Aranda, 2012).

Sastoji se od metoda koje su definirane od strane nadklase `PeriodicBehaviour` gdje se kod metode pokretanja `onStart()` šalje poruka registracije agentu izvjestitelju o Facebook agentu pomoću metode `__send_registration_message()`. Na taj način agent izvjestitelj vodi evidenciju o prijavljenom agentu i njegovim podacima. U svakom periodu provjerava se da li je broj perioda koje je korisnik definirao iskorišten. Ukoliko jest, onda se prelazi iz metode `_onTick()` na samo jedno izvršavanje metode `onEnd()` gdje se agentu izvjestitelju šalje poruka da agent želi dobiti izvješće i to pomoću metode `__send_report_message()`. U suprotnosti, kada je broj perioda veći od 0, šalje se zahtjev za dohvaćanjem podataka sa Facebook servisa te se provjerava što promijenilo od zadnje provjere. To je omogućeno pomoću metode `fetch_data()`. Nakon dohvaćanja podataka, ukoliko je bilo kakve promjene, dohvaćeni podaci se šalju agentu izvjestitelju pomoću metode `__send_status_message()` te se tamo spremaju za daljnje korištenje.

`ReportDeliveryBehaviour` je ponašanje koje je definirano kao `EventBehaviour` što predstavlja ponašanje koje je kreirano na način da se okida na neki događaj. Isto tako, potrebno je napomenuti da ovo ponašanje nije instancirano niti pokrenuto tako dugo dok ga ne okine određen događaj (G. Aranda, 2012).

Sastoji se od metode `_process()` koja je definirana nadklasom `EventBehaviour`, a radi na način da očekuje poruku od agenta izvjestitelja. Kada ju dobije ispiše njen sadržaj na ekran i završi s radom agenta.

Konačno, sam agent `FacebookAgent` nasljeđuje klasu `Agent` te se time definiraju konstruktor i metoda `_setup()`. Konstruktor je u ovom slučaju definiran drugačije od zadanog, pa je potrebno ručno pozvati konstuktor i proslijediti potrebne podatke. Podaci koje je potrebno proslijediti jesu korisničko ime i lozinka agenta, a to se generira pomoću metode `__generate_agent_credentails()`. Ostali podaci koji su proslijeđeni kroz konstruktor su tražena riječ, vrijeme trajanja periode i broj perioda.

Da bi radilo slanje poruka između agenata, u metodi `_setup()` definirana su dva ponašanja, s time da je bilo potrebno za ponašanje `ReportDeliveryBehavior` definirati i predložak sa definiranom ontologijom što je u ovom slučaju `delivery_template`.

3.2 Implementacija Twitter agenta

U ovom potpoglavlju prikazat ću način na koji sam implementirao Twitter agenta. Prije prikaza same implementacije, potrebno je naglasiti, kao i kod Facebook agenta, da je važno da se unutar datoteke *credentials.json* unesu potrebne vrijednosti za `api_key`, `api_key_secret`, `access_token` i `access_token_secret`. Navedene podatke je moguće dobiti nakon što se kreira nova aplikacija unutar Twitter aplikacije u dijelu za razvijatelje aplikacija.

3.2.1 Prikaz implementacije modela

Za dobivanje podataka potrebnih za spajanje na Twitter-ov servis, koristio sam sljedeću klasu *TwitterCredentials*. Pomoću metode `__read_configuration()` dohvaća se datoteka unutar koje se nalaze podaci o spajanju na Twitter servis, dok se pomoću metode `get_credentials()` dohvaćaju navedeni podaci iz datoteke te vraćaju kao n-torka.

```
#! /usr/bin/usr python
# -*- coding: utf-8 -*-

import json


class TwitterCredentials:

    def __init__(self, filename):
        self.filename = filename

    def get_credentials(self):
        json_data = self.__read_configuration_file()
        api_key = json_data["twitter"]["api_key"]
        api_key_secret = json_data["twitter"]["api_key_secret"]
        access_token = json_data["twitter"]["access_token"]
        access_token_secret = json_data["twitter"]["access_token_secret"]
        return api_key, api_key_secret, access_token, access_token_secret

    def __read_configuration_file(self):
        file = open(self.filename, "r")
        data = json.load(file)
        file.close()
        return data
```

Sljedeća klasa koja predstavlja model je *TwitterPost*. Ova klasa sadrži osnovne podatke o statusu odnosno tweet-u, a to su redom; vrijeme objave, id posljednje objave, tekst statusa, korisničko ime, ime i prezime korisnika te lokacija.

```
#! /usr/bin/usr python
# -*- coding: utf-8 -*-

from datetime import datetime, timedelta


class TwitterPost:

    def __init__(self, data):
        self.date = datetime.strptime(data["created_at"],
                                      "%a %b %d %H:%M:%S +0000 %Y")
                                        + timedelta(hours=1)
```

```
        self.since_id = data["id_str"]
        self.text = data["text"]
        self.username = data["user"]["screen_name"]
        self.name = data["user"]["name"]
        self.location = data["user"]["location"]
```

3.2.2 Prikaz implementacije API-a

Sljedeće što je potrebno da bi Twitter agent radio i dohvaćao podatke je API koji definira metode pomoću kojih je moguće spajanje na Twitter servis. Na taj način se dobivaju podaci ovisno da li se radi o dohvaćanju statusa korisnika ili novih statusa na temelju hashtag-a.

```
#! /usr/bin/usr python
# -*- coding: utf-8 -*-

import requests
import requests_oauthlib

from social.twitter.twitter_models import *


class TwitterAPI:

    def __init__(self, credentials):
        self.__credentials = credentials

    def search_hashtag(self, hashtag, since_id=0):
        url = "https://api.twitter.com/1.1/search/tweets.json"
        params = {"q": hashtag, "result_type": "recent", "since_id": since_id}
        auth = requests_oauthlib.OAuth1(*self.__credentials)
        request = requests.get(url, params=params, auth=auth)
        hashtags = []
        for json_hashtag in reversed(request.json()["statuses"]):
            hashtag = TwitterPost(json_hashtag)
            hashtags.append(hashtag)
        return hashtags

    def search_tweets(self, username, since_id=0):
        url = "https://api.twitter.com/1.1/statuses/user_timeline.json"
        params = {"screen_name": username}
        if since_id > 0:
            params["since_id"] = since_id
        auth = requests_oauthlib.OAuth1(*self.__credentials)
        request = requests.get(url, params=params, auth=auth)
        tweets = []
        for json_tweet in reversed(request.json()):
```

```
            tweet = TwitterPost(json_tweet)
            tweets.append(tweet)
        return tweets
```

Definirane su dvije osnovne metode pomoću kojih se dohvaćaju podaci preko API-a. To su metode `search_hashtag(hashtag, since_id` i `search_tweets(username, since_id)`.

Metoda `search_hashtag(hashtag, since_id)` traži sve statuse koji sadrže navedeni hashtag, a nastali su nakon prethodnog traženja od strane korisnika što u ovom slučaju predstavlja parameter `since_id`. Metoda `search_tweets(username, since_id(` traži nove statuse od određenog korisnika koji su nastali nakon zadnje pretrage. Obje metode nakon pozivanja vraćaju statuse koji su dohvaćeni u navedenom vremenu.

3.2.3 Prikaz implementacije reaktivnog agenta

Ovdje ću prikazati konačnu implementaciju agenta koji na Twitter društvenoj mreži dohvaća promjene u smislu novih statusa od stane korisnika ili novih statusa koji sadrže definirani hashtag.

```
#! /usr/bin/usr python
# -*- coding: utf-8 -*-

import json
from datetime import datetime, timedelta

from spade import Agent
from spade import Behaviour
from spade import AID
from spade import ACLMessage

from social.twitter.twitter_api import TwitterAPI
from social.twitter.twitter_credentials import TwitterCredentials
from social.shared.report_message import ReportMessage


class TwitterFetchAgent(Agent.Agent):

    class FetchBehaviour(Behaviour.PeriodicBehaviour):

        def __init__(self, time, period, credentials_filename):
            Behaviour.PeriodicBehaviour.__init__(self, period)
            self.periods = time / period
            self.current_date = datetime.now()
            twitter_credentials = TwitterCredentials(credentials_filename)
            self.twitter_api = TwitterAPI(twitter_credentials.get_credentials())

        def onStart(self):
            print "[" + self.myAgent.getName() + "] "
                + self.myAgent.fetch_type.capitalize()
```

```
            + " activity fetch started."
        self.__send_registration_message()

    def onEnd(self):
        print "[" + self.myAgent.getName() + "] "
            + self.myAgent.fetch_type.capitalize()
            + " activity fetch ended."
        self.__send_report_message()

    def _onTick(self):
        if self.periods > 0:
            self.periods -= 1
            self.fetch_data()
        else:
            self.kill()

    def fetch_data(self):
        if self.myAgent.fetch_type == "tweet":
            print "[" + self.myAgent.getName()
                + "] Fetching tweets... (No. of periods left: "
                + str(self.periods) + ")"
            results = self.twitter_api.search_tweets(self.myAgent.keyword)
        else:
            print "[" + self.myAgent.getName()
                + "] Fetching hashtags... (No. of periods left: "
                + str(self.periods) + ")"
            results = self.twitter_api.search_hashtag(self.myAgent.keyword)

        tweets_found = 0
        for tweet in results:
            if (tweet.date - self.current_date) > timedelta(seconds=1):
                self.current_date = tweet.date
                self.__send_tweet_message(tweet)
                tweets_found += 1
        print "[" + self.myAgent.getName() + "] Found "
            + self.myAgent.fetch_type + "s: " + str(tweets_found)

    def __send_registration_message(self):
        message = ACLMessage.ACLMessage()
        message.addReceiver(self.myAgent.receiver)
        message.setOntology("register")
        message.setContent(self.myAgent.getName())
        self.myAgent.send(message)

    def __send_tweet_message(self, tweet):
```

```
            message = ACLMessage.ACLMessage()
            message.addReceiver(self.myAgent.receiver)
            message.setOntology("notify")
            object = ReportMessage("Twitter", self.myAgent.fetch_type,
                                   self.myAgent.keyword,
                                   tweet.username, tweet.name,
                                   tweet.date, tweet.text)
            value = json.dumps(object.__dict__)
            message.setContent(value)
            self.myAgent.send(message)

        def __send_report_message(self):
            message = ACLMessage.ACLMessage()
            message.addReceiver(self.myAgent.receiver)
            message.setOntology("report")
            message.setContent(self.myAgent.getName())
            self.myAgent.send(message)

    class ReportDeliveryBehaviour(Behaviour.EventBehaviour):

        def _process(self):
            received_message = self._receive()
            if received_message:
                content = json.loads(received_message.getContent())
                print ""
                print "[" + self.myAgent.getName()
                    + "] Received message from: "
                    + received_message.getSender().getName()
                print "[" + self.myAgent.getName()
                    + "] Total number of fetched data: "
                    + str(len(content))
                print ""
                for element in content:
                    notify_message = ReportMessage()
                    notify_message.load_json(element)
                    notify_message.print_message()
            self.myAgent.stop()

    def __init__(self, reporter_name, keyword, fetch_type="tweet",
                 credentials_filename="credentials.json", time=60, period=10):
        agent_id, password = self.__generate_agent_credentials()
        Agent.Agent.__init__(self, agent_id, password)
        self.keyword = keyword
        self.fetch_type = fetch_type
        self.credentials_filename = credentials_filename
```

```
        self.time = time
        self.period = period
        self.receiver = AID.aid(name=reporter_name,
                                addresses=["xmpp://" + reporter_name])

    def _setup(self):
        print "[" + self.getName() + "] Twitter fetch agent is starting..."
        fetch_behaviour = self.FetchBehaviour(self.time, self.period,
                                              self.credentials_filename)
        self.addBehaviour(fetch_behaviour)
        delivery_template = Behaviour.ACLTemplate()
        delivery_template.setOntology("report_delivery")
        self.addBehaviour(self.ReportDeliveryBehaviour(), delivery_template)

    def __generate_agent_credentials(self):
        agent_name = "twitter_" + datetime.now().strftime("%H%M%S")
                     + "@127.0.0.1"
        return agent_name, "secret"
```

Agent ima dvije podklase koje definiraju ponašanje agenta, a to su *FetchBehaviour* i *ReportDeliveryBehaviour*.

`FetchBehaviour` je ponašanje koje je definirano kao `PeriodicBehaviour`. Sastoji se od metoda koje su definirane od strane nadklase `PeriodicBehaviour` gdje se kod pokretanja kod metode `onStart()` šalje poruka registracije agentu izvjestitelju o Twitter agentu pomoću metode `__send_registration_message()` kako bi agent izvjestitelj mogao voditi evidenciju o njemu i njegovim podacima. U svakom periodu u metodi `_onTick()` provjerava se da li je broj perioda koje je korisnik definirao iskorišten. Ukoliko jest, onda se prelazi iz metode `_onTick()` na samo jedno izvršavanje metode `onEnd()` gdje se agentu izvjestitelju šalje poruka da agent želi dobiti izvješće i to pomoću metode `__send_report_message()`. U suprotnosti se šalje zahtjev za provjerom da li se na Twitter servisu što promijenilo od zadnje provjere. Ovdje je važno spomenuti da se na temelju korisničkog unosa odnosno definiranja pretrage šalje različiti zahtjev, tj. poziva se različita metoda. Metoda koja se koristi za dohvaćanje novih podataka je `fetch_data()` te ona poziva `self.twitter_api.search_tweets(username, since_id)` ako je korisnik kao vrstu pretrage odabrao *hashtag*, dok se u slučaju vrste pretrage *tweet* poziva metoda `self.twitter_api.search_hashtags(hashtag, since_id)`. Nakon što se podaci dohvate, isti se šalju agentu izvjestitelju.

`ReportDeliveryBehaviour` je ponašanje koje je definirano kao `EventBehaviour`. Sastoji se od samo jedne metode koja je definirana nadklasom `EventBehaviour`, a radi na način da očekuje poruku od agenta izvjestitelja, a kada ju dobije ju ispiše na ekran i završi s radom agenta.

Konačno, sam agent `TwitterAgent` nasljeđuje klasu `Agent` te se time definiraju konstruktor i metoda `_setup()`. Konstruktor je u ovom slučaju definiran drugačije od zadanog, pa je potrebno ručno pozvati konstuktor i proslijediti potrebne podatke. Podaci koje je potrebno proslijediti jesu korisničko ime i lozinka agenta, a to se generira pomoću metode `__generate_agent_credentails()`.

Radi slanje poruka između agenata, u metodi `_setup()` definirana su dva ponašanja isto kao i kod Facebook agenta kako bi se poruke procesuirale na odgovarajući način.

3.3 Implementacija agenta za izvještaje

Važna stavka ovog rješenja je agent izvjestitelj. Ovaj agent zadužen je za prihvaćanje svih informacija koje dobije od agenata koji dohvaćaju podatke s društvenih mreža te na kraju rada svakog pojedinog agenta vraća istome konačan skup njegovih podataka.

Da bi agent mogao komunicirati s agentima koji dohvaćaju podatke s društvenih mreža (Facebook, Twitter), definirana je klasa koja predstavlja model poruke pomoću koje se šalju konačni rezultati odnosno izvještaj za određenog agenta. Klasa *ReportMessage* sadrži informacije o poruci kao što je vrsta mreže koja predstavlja o kojem se agentu radi (Facebook ili Twitter), vrsti poruke koja predstavlja da li se radi o statusu ili slici, traženoj riječi, korisničkom imenu, imenu i prezimenu korisnika, datumu i sadržaju statusa. Također, pošto se radi o JSON tipu poruke koja služi za komunikaciju između agenata, potrebno je moći pročitati sadržaj, a za to služi metoda `load_json(data)`. Uz to, definirana je metoda za ispis rezultata kako bi on bio definiran na razini modela, a naziv metode je `print_message()`.

```
#! /usr/bin/usr python
# -*- coding: utf-8 -*-

from datetime import datetime


class ReportMessage:

    def __init__(self, network=None, message_type=None,
                 keyword=None, username=None,
                 name=None, date=None, text=None):
        self.network = network
        self.message_type = message_type
        self.keyword = keyword
        self.username = username
        self.name = name
        if date:
            self.date = date.strftime("%H:%M:%S, %d.%m.%Y")
        else:
            self.date = date
        self.text = text

    def load_json(self,data):
        self.network = data["network"]
        self.message_type = data["message_type"]
        self.keyword = data["keyword"]
        self.username = data["username"]
        self.name = data["name"]
        self.date = data["date"]
        self.text = data["text"]

    def print_message(self):
```

```
        print "network=" + self.network + "; type="
            + self.message_type + "; keyword=" + self.keyword
        print "username=" + self.username + "; name=" + self.name
        print "date=" + self.date
        if self.text == "":
            print "message=<empty_message>"
        else:
            print "message=" + self.text.replace("\n", " ").replace("\r", "")
        print ""
```

Agent je implementiran u klasi *ReportAgent* te ima tri podklase koje definiraju ponašanje agenta, a to su *RegisterBehaviour*, *NotifyBehaviour* i *ReportBehaviour*.

Sva tri ponašanja definirana su kao `EventBehaviour`, a nešto više o toj vrsti ponašanja rečeno je u ranijim poglavljima.

`RegisterBehaviour` se sastoji od metode `_process()` koja je definirana od strane nadklase `EventBehaviour`, a radi na način da dohvaća poruke od strane drugih agenata (Facebook, Twitter) te ih registrira na način da ih spremi u rječnik (engl. dictionary). Pošto svaki od agenata ima drugačije ime, rječnik je dobar način spremanja pošto ime kao ključ nikad neće biti isto.

`NotifyBehaviour` se sastoji od metode `_process()` koja je definirana od strane nadklase `EventBehaviour`, a radi na način da dohvaća poruke od strane drugih agenata (Facebook, Twitter). Ukoliko je poruka primljena, poruka se otpakira iz JSON formata u određeni model te se kao takva sprema u rječnik i ispisuje.

`ReportBehaviour` se sastoji od metode `_process()` koja je definirana od strane nadklase `EventBehaviour`, a radi na način da dohvaća poruke od strane drugih agenata (Facebook, Twitter). Navedeni agenti kada su pri kraju s radom šalju zahtjev za konačnim izvješćem od strane agenta izvjestitelja, pa agent dohvaća podatke iz rječnika i šalje navedenom agentu. Nakon što se poslala poruka s podacima, isti se agent briše iz rječnika. Metoda pomoću koje se šalje poruka je `send_report(agent_name)`.

Agent `ReportAgent` nasljeđuje klasu `Agent` te se time definiraju konstruktor i metoda `_setup()`. Unutar ove metode potrebno je definirati predloške koji će biti primjenjeni za svako od navedenih ponašanja kako bi poruke stizale samo za navedena ponašanja.

```
#! /usr/bin/usr python
# -*- coding: utf-8 -*-

import json

from spade import Agent
from spade import Behaviour
from spade import ACLMessage
from spade import AID

from social.shared import report_message


class ReportAgent(Agent.Agent):
```

```
class RegisterBehaviour(Behaviour.EventBehaviour):

    def _process(self):
        received_message = self._receive()
        if received_message:
            agent_name = received_message.getContent()
            self.myAgent.reports[agent_name] = []
            print "[" + self.myAgent.getName()
                + "] New agent (" + agent_name + ") started fetching."
        print "[" + self.myAgent.getName() + "] No. of agents working: "
            + str(len(self.myAgent.reports))

class NotifyBehaviour(Behaviour.EventBehaviour):

    def _process(self):
        received_message = self._receive()
        if received_message:
            content = received_message.getContent()
            notify_message = report_message.ReportMessage()
            notify_message.load_json(json.loads(content))
            self.myAgent.reports[received_message.getSender().getName()]
                .append(notify_message)
            print "[" + self.myAgent.getName()
                + "] Received message from: "
                + received_message.getSender().getName()
            notify_message.print_message()

class ReportBehaviour(Behaviour.EventBehaviour):

    def _process(self):
        received_message = self._receive()
        if received_message:
            agent_name = received_message.getContent()
            if agent_name in self.myAgent.reports:
                self.send_report(agent_name)
                del(self.myAgent.reports[agent_name])
                print  "[" + self.myAgent.getName()
                    + "] Agent (" + agent_name
                    + ") stopped fetching."
        print "[" + self.myAgent.getName()
            + "] No. of agents working: "
            + str(len(self.myAgent.reports))

    def send_report(self, agent_name):
        report_messages = self.myAgent.reports[agent_name]
```

```
            receiver = AID.aid(name=agent_name,
                               addresses=["xmpp://" + agent_name])
            message = ACLMessage.ACLMessage()
            message.addReceiver(receiver)
            message.setOntology("report_delivery")
            message.setContent(json.dumps([obj.__dict__
                                           for obj in report_messages]))
            self.myAgent.send(message)

    def _setup(self):
        print "[" + self.getName() + "] Report agent is online."
        self.reports = {}

        register_template = Behaviour.ACLTemplate()
        register_template.setOntology("register")
        self.addBehaviour(self.RegisterBehaviour(), register_template)

        notify_template = Behaviour.ACLTemplate()
        notify_template.setOntology("notify")
        self.addBehaviour(self.NotifyBehaviour(), notify_template)

        report_template = Behaviour.ACLTemplate()
        report_template.setOntology("report")
        self.addBehaviour(self.ReportBehaviour(), report_template)
```

3.4 Implementacija kreiranja agenata

Na kraju, vrijedi prikazati implementaciju klase *SocialNotifier* unutar koje je definirano na koji način se pokreće program. Korištena je biblioteka *argparse* koja omogućuje definiranje korištenja argumenata na vrlo jednostavan način, odnosno da se definira broj potrebnih argumenata, koliko je obaveznih te koliko opcionalnih argumenata i na kraju da se navedeni podaci vrlo lako koriste u daljnjem izvođenju programa. U programskom kodu je vidljivo da se definira `parser` koji obuhvaća tri načina unosa i to kao `reporter_parser`, `twitter_parser` i `facebook_parser`. U ovisnosti o načinu pokretanja programa, pokrenut će se drugačiji agent i to:

- ReportAgent
- FacebookAgent
- TwitterAgent

Nakon provjere argumenata, provjerava se koja je od komandi unesena što predstavlja zapravo naziv prvog argumenta. Ukoliko je komanda unosa odgovarajuća, poziva se jedna od navedenih metoda ovisno u nazivu komande; `__start_reporter(result)` za pokretanje *ReportAgent*-a, `__start_twitter_fetch(result)` za pokretanje *TwitterAgent*-a te `__start_facebook_fetch(result)` za pokretanje *FacebookAgent*-a.

Za svakog od agenata postoje predefinirani odnosno opcionalni argumenti, a kod svih agenata zajednički je zadan argument *config* koji je zapravo naziv za datoteku konfiguracije, u ovom slučaju

config.json. Za promjenu iste, potrebno je kao argument pri pokretanju unijeti drugi naziv konfiguracijske datoteke ukoliko je to potrebno. Isto vrijedi i za ostale argumente, gdje su neki obavezni, a neki opcionalni, a tu spadaju redom:

- keyword - tražena riječ u pretrazi
- type - način tj. vrsta pretrage
- credentials - naziv datoteke s podacima za spajanje na društvenu mrežu
- time - ukupno vrijeme trajanja dohvaćanja podataka
- period - vrijeme trajanje jedne periode

Posljednja metoda koja se nalazi u kodu jest `__read_config_file(filename)` pomoću koje se dohvača sadržaj datoteke *config.json* u kojoj se nalazi korisničko ime i lozinka agenta izvjestitelja.

```
#! /usr/bin/usr python
# -*- coding: utf-8 -*-

import argparse
import sys
import json
from os import path

from agents.facebook_agent import FacebookAgent
from agents.twitter_agent import TwitterFetchAgent
from agents.report_agent import ReportAgent


class SocialNotifier:

    def __init__(self):
        self.parse_parameters()

    def parse_parameters(self):
        parser = argparse.ArgumentParser()
        subparsers = parser.add_subparsers(dest="command")

        # Reporter
        reporter_parser = subparsers.add_parser("reporter",
                                                help="Creates social reporter")
        reporter_parser.add_argument("-config",
                                     default="config.json",
                                     action="store",
                                     help="Configuration filename")

        # Twitter
```

```
        twitter_parser = subparsers.add_parser("twitter",
                                               help="Creates a Twitter fetch agent")
        twitter_parser.add_argument("keyword",
                                    action="store",
                                    help="Defines search keyword (username or hashtag
        twitter_parser.add_argument("type",
                                    action="store",
                                    help="Defines search type ('tweet' or 'hashtag')"
        twitter_parser.add_argument("--credentials",
                                    default="credentials.json",
                                    action="store",
                                    help="Credentials filename")
        twitter_parser.add_argument("--config",
                                    default="config.json",
                                    action="store",
                                    help="Configuration filename")
        twitter_parser.add_argument("--time", default=120, action="store",
                                    help="Length of agent lifetime in secods",
                                    type=int)
        twitter_parser.add_argument("--period", default=15, action="store",
                                    help="Lenght of period in seconds",
                                    type=int)

        # Facebook
        facebook_parser = subparsers.add_parser("facebook",
                                                help="Creates a Facebook fetch agent"
        facebook_parser.add_argument("keyword",
                                     action="store",
                                     help="Defines search keyword (username)")
        facebook_parser.add_argument("--credentials",
                                     default="credentials.json",
                                     action='store',
                                     help="Credentials filename")
        facebook_parser.add_argument("--config",
                                     default="config.json",
                                     action="store",
                                     help="Configuration filename")
        facebook_parser.add_argument("--time",
                                     default=120,
                                     help="Length of agent lifetime in secods",
                                     type=int)
        facebook_parser.add_argument("--period",
                                     default=15,
                                     help="Lenght of period in seconds",
                                     type=int)
```

```
        result = parser.parse_args()
        if not path.isfile(result.config):
            sys.exit("Error: No configuration file found.")

        if result.command == "reporter":
            self.__start_reporter(result)
        elif result.command == "twitter":
            if not path.isfile(result.credentials):
                sys.exit("Error: No credentials file found.")
            if result.type != "tweet" and result.type != "hashtag":
                sys.exit("Error: Type must be 'tweet' or 'hashtag'.")
            self.__start_twitter_fetch(result)
        elif result.command == "facebook":
            if not path.isfile(result.credentials):
                sys.exit("Error: No credentials file found.")
            self.__start_facebook_fetch(result)
        else:
            sys.exit("Error: Problem with parameters.")

    def __start_reporter(self, result):
        name, password = self.__read_config_file(result.config)
        reporter = ReportAgent(name, password)
        reporter.start()

    def __start_twitter_fetch(self, result):
        reporter_name, _ = self.__read_config_file(result.config)
        agent = TwitterFetchAgent(reporter_name, result.keyword,
                                  result.type, result.credentials,
                                  result.time, result.period)
        agent.start()

    def __start_facebook_fetch(self, result):
        reporter_name, _ = self.__read_config_file(result.config)
        agent = FacebookAgent(reporter_name, result.keyword,
                              result.credentials, result.time,
                              result.period)
        agent.start()

    def __read_config_file(self, filename):
        file = open(filename, "r")
        json_data = json.load(file)
        agent_name = json_data["agent_name"]
        agent_password = json_data["agent_password"]
        file.close()
```

```
        return agent_name, agent_password


if __name__ == "__main__":
    social_notifier = SocialNotifier()
```

Poglavlje 4

Demonstracija rješenja

U ovom poglavlju prikazat ću na koji način navedeno rješenje funkcionira, odnosno kako se izvršava. Prije samog pokretanja programa, potrebno je imati instalirane određene biblioteke kako bi program mogao raditi. Te biblioteke su *SPADE*, *requests*, *requests_oauthlib* i *facebook-sdk*.

Nakon instalacije navedenih biblioteka, moguće je početi s izvršavanjem programa, a prvo je potrebno pokrenuti *SPADE* pomoću skripte `runspade.py`.

Nakon toga, potrebno je pokrenuti agenta izvjestitelja. Pokretanje izvjestitelja prikazano je na slici 4.1.

```
nikolamajcen@ubuntu:~/Development/Python/social-notifier$ python social-notifier.py reporter
[reporter@127.0.0.1] Report agent is online.
```

Slika 4.1: Pokretanje agenta izvjestitelja

Kada je agent izvjestitelj aktivan, moguće je pokrenuti i ostale agente. Ovdje ću za početak prikazati rad Twitter agenta i to tako da traži nove statuse sa određenih hashtag-om. Za pokretanje Twitter agenta koristi se sjedeći način pokretanja koji se može vidjeti na slici 4.2. zajedno sa načinom rada agenta. Važno je reći da su u ovom slučaju uneseni samo potrebni parametri odnosno argumenti, a svi ostali argumenti odnosno parametri programa su zadani u slučaju ako nisu uneseni od strane korisnika.

```
nikolamajcen@ubuntu:~/Development/Python/social-notifier$ python social-notifier.py twitter "#trump" hashtag
[twitter_225616@127.0.0.1] Twitter fetch agent is starting...
[twitter_225616@127.0.0.1] Hashtag activity fetch started.
[twitter_225616@127.0.0.1] Fetching hashtags... (No. of periods left: 7)
[twitter_225616@127.0.0.1] Found hashtags: 0
[twitter_225616@127.0.0.1] Fetching hashtags... (No. of periods left: 6)
[twitter_225616@127.0.0.1] Found hashtags: 0
[twitter_225616@127.0.0.1] Fetching hashtags... (No. of periods left: 5)
[twitter_225616@127.0.0.1] Found hashtags: 4
[twitter_225616@127.0.0.1] Fetching hashtags... (No. of periods left: 4)
[twitter_225616@127.0.0.1] Found hashtags: 0
[twitter_225616@127.0.0.1] Fetching hashtags... (No. of periods left: 3)
[twitter_225616@127.0.0.1] Found hashtags: 7
[twitter_225616@127.0.0.1] Fetching hashtags... (No. of periods left: 2)
[twitter_225616@127.0.0.1] Found hashtags: 0
```

Slika 4.2: Pokretanje Twitter agenta za hashtag pretragu

Kako agent, u ovom slučaju Twitter agent, dohvaća nove statuse sa zadanih hashtag-om, on ispisuje koliko je statusa dohvatio u pojedinom periodu te odbrojava broj preostalih perioda. U svakom od tih perioda šalje agentu izvjestitelju u realnom vremenu podatke o dohvaćenim statusima ukoliko ih ima. Agent izvjestitelj te podatke dohvaća, ispisuje i sprema u rječnik.

```
nikolamajcen@ubuntu:~/Development/Python/social-notifier$ python social-notifier.py reporter
[reporter@127.0.0.1] Report agent is online.
[reporter@127.0.0.1] New agent (twitter_225616@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] Received message from: twitter_225616@127.0.0.1
network=Twitter; type=hashtag; keyword=#trump
username=JoLuMorales; name=José Luis Morales
date=22:56:19, 10.01.2017
message=RT @MatthewBird_PR: Privatizing #Philanthropy Make Money While Doing Good @MatthewBird_pr #un #sdgs #trump #leadershipship https://t.co/08U…

[reporter@127.0.0.1] Received message from: twitter_225616@127.0.0.1
network=Twitter; type=hashtag; keyword=#trump
username=Ueverthink; name=2017 MAGA Tammy
date=22:56:21, 10.01.2017
message=RT @MightyBusterBro: . HILLARY RIDICULED by TAIWAN video DEMS are FURIOUS  #CrookedHillary #Liberals  #LiberalLogic #Trump #MAGA #TrumpStro…

[reporter@127.0.0.1] Received message from: twitter_225616@127.0.0.1
network=Twitter; type=hashtag; keyword=#trump
username=DefenceAssoc; name=UKNDA
date=22:56:23, 10.01.2017
message=RT @NATOSource: European Allies Appeal to Donald #Trump Through Letter https://t.co/es71c4nMY2 #Russia #Putin #NATO https://t.co/MgcDNeeRVl

[reporter@127.0.0.1] Received message from: twitter_225616@127.0.0.1
network=Twitter; type=hashtag; keyword=#trump
username=aonbhar; name=Aonbhar
date=22:56:25, 10.01.2017
message=#Trump selections based on lack of ethics, easier to manipulate? #Resist https://t.co/BFHz9fv65v
```

Slika 4.3: Dobivanje poruka od Twitter agenta

Nakon što Twitter agent dohvati podatke određen broj puta, Twitter agent šalje nakon toga zahtjev za konačnim izvještajem o svim dohvaćenim podacima, a rezultat toga može se vidjeti na slici 4.4.

Sve do ovog koraka, Twitter agent ne zna koje je sve podatke dohvatio, već su svi podaci pohranjeni kod agenta izvjestitelja. Upravo zato potrebno je da se na kraju svakog rada pojedinog agenta zatraži izvještaj sa svim dohvaćenim podacima koje se nalaze kod agenta izvjestitelja.

```
[twitter_225616@127.0.0.1] Found hashtags: 0
[twitter_225616@127.0.0.1] Hashtag activity fetch ended.

[twitter_225616@127.0.0.1] Received message from: reporter@127.0.0.1
[twitter_225616@127.0.0.1] Total number of fetched data: 16

network=Twitter; type=hashtag; keyword=#trump
username=JoLuMorales; name=José Luis Morales
date=22:56:19, 10.01.2017
message=RT @MatthewBird_PR: Privatizing #Philanthropy Make Money While Doing Good @MatthewBird_pr #un #sdgs #trump #leadershipship https://t.co/08U…

network=Twitter; type=hashtag; keyword=#trump
username=Ueverthink; name=2017 MAGA Tammy
date=22:56:21, 10.01.2017
message=RT @MightyBusterBro: . HILLARY RIDICULED by TAIWAN video DEMS are FURIOUS  #CrookedHillary #Liberals  #LiberalLogic #Trump #MAGA #TrumpStro…

network=Twitter; type=hashtag; keyword=#trump
username=DefenceAssoc; name=UKNDA
date=22:56:23, 10.01.2017
message=RT @NATOSource: European Allies Appeal to Donald #Trump Through Letter https://t.co/es71c4nMY2 #Russia #Putin #NATO https://t.co/MgcDNeeRVl

network=Twitter; type=hashtag; keyword=#trump
username=aonbhar; name=Aonbhar
date=22:56:25, 10.01.2017
message=#Trump selections based on lack of ethics, easier to manipulate? #Resist https://t.co/BFHz9fv65v

network=Twitter; type=hashtag; keyword=#trump
username=LuigiVampa35; name=LuigiVampa35
date=22:56:39, 10.01.2017
message=RT @ANTIBURREGO: Promesa d #Trump a #Israel sobre Al-Quds es una bofetada al mundo https://t.co/AWZZMO7QIB //Quien era el MAL MAYOR?. @Luig…

network=Twitter; type=hashtag; keyword=#trump
username=craig_brennan; name=Craig Brennan
date=22:56:41, 10.01.2017
message=Meh:  #JeffSessions expresses desire to prosecute porn. Awesome:  #Trump has appeared in two soft-core porn movies. https://t.co/xbt3L4Ckwg

network=Twitter; type=hashtag; keyword=#trump
username=MiamiGives; name=Miami Gives
date=22:56:44, 10.01.2017
message=RT @MatthewBird_PR: Privatizing #Philanthropy Make Money While Doing Good @MatthewBird_pr #un #sdgs #trump #leadershipship https://t.co/08U…

network=Twitter; type=hashtag; keyword=#trump
username=romioneinlove; name=Amortentia
date=22:56:46, 10.01.2017
message=RT @RimastoConHarry: #merylstreep #trump #JKRowling #twitter #supporto #complimento #offesa seguici su Facebook! https://t.co/lpvrNbfv9m

network=Twitter; type=hashtag; keyword=#trump
username=Artfull01; name=George
date=22:56:48, 10.01.2017
message=RT @DaysOfTrump: 10 Days to #Trump : Counting down to Inauguration Day and @DaysOfTrump in the White House! #MakeAmericaGreatAgain https://…
```

Slika 4.4: Dobivanje izvještaja od strane agenta izvjestitelja

Kada je Twitter agent dobio konačni izvještaj, on završava sa radom. Toga je svjestan i agent izvjestitelj, te on prikazuje novo stanje broja aktivnih agenata svaki puta kada pošalje izvještaj pojedinom agenta. Razlog tome je što svaki agent završava s radom kad dobije izvještaj o dohvaćanju podataka pa se na temelju toga može vidjeti koliko je još aktivnih agenata.

Naravno, novi agenti se registriraju na početku rada svakog agenta, tako da se i u tom slučaju ispisuju podaci o aktivnim agentima. Primjer praćenja broja agenata može se vidjeti na slici 4.5.

```
[reporter@127.0.0.1] Agent (twitter_225616@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 0
```

Slika 4.5: Ažuriranje broja aktivnih agenata

Drugi način rada Twitter agenta je dohvaćanje statusa pojedinog korisnika, tako da se umjesto tipa `hashtag` unese `tweet`. Također, potrebno je umjesto ključne riječi ovaj put unijeti korisničko ime korisnika čije statuse se želi pratiti.

```
nikolamajcen@ubuntu:~/Development/Python/social-notifier$ python social-notifier.py twitter "nikolamajcen" tweet
[twitter_230129@127.0.0.1] Twitter fetch agent is starting...
[twitter_230129@127.0.0.1] Tweet activity fetch started.
[twitter_230129@127.0.0.1] Fetching tweets... (No. of periods left: 7)
[twitter_230129@127.0.0.1] Found tweets: 0
[twitter_230129@127.0.0.1] Fetching tweets... (No. of periods left: 6)
[twitter_230129@127.0.0.1] Found tweets: 0
[twitter_230129@127.0.0.1] Fetching tweets... (No. of periods left: 5)
[twitter_230129@127.0.0.1] Found tweets: 0
[twitter_230129@127.0.0.1] Fetching tweets... (No. of periods left: 4)
[twitter_230129@127.0.0.1] Found tweets: 0
[twitter_230129@127.0.0.1] Fetching tweets... (No. of periods left: 3)
[twitter_230129@127.0.0.1] Found tweets: 1
[twitter_230129@127.0.0.1] Fetching tweets... (No. of periods left: 2)
[twitter_230129@127.0.0.1] Found tweets: 0
[twitter_230129@127.0.0.1] Fetching tweets... (No. of periods left: 1)
[twitter_230129@127.0.0.1] Found tweets: 1
[twitter_230129@127.0.0.1] Fetching tweets... (No. of periods left: 0)
[twitter_230129@127.0.0.1] Found tweets: 0
[twitter_230129@127.0.0.1] Tweet activity fetch ended.

[twitter_230129@127.0.0.1] Received message from: reporter@127.0.0.1
[twitter_230129@127.0.0.1] Total number of fetched data: 2

network=Twitter; type=tweet; keyword=nikolamajcen
username=nikolamajcen; name=Nikola
date=23:02:22, 10.01.2017
message=Testing VAS project :) #VAS #FOI

network=Twitter; type=tweet; keyword=nikolamajcen
username=nikolamajcen; name=Nikola
date=23:02:50, 10.01.2017
message=Closing Twitter for now :)
```

Slika 4.6: Pokretanje Twitter agenta za dohvaćanjem statusa korisnika

Na slici 4.6 se može vidjeti rad tj. pokretanje Twitter agenta uz način rada `tweet` koji definira dohvaćanje novih statusa određenog korisnika.

Agent u određenom broju perioda dohvaća statuse navedenog korisnika, te pronađene statuse šalje agentu izvjestitelju. Agent u ovom slučaju ne zna za podatke koje je dohvatio, nego iste direktno šalje agentu izvjestitelju kod kojeg su pohranjeni svi dohvaćeni podaci. Iz tog razloga, kada dohvaćanje podataka završi, Twitter agent od agenta izvjestitelja dobiva izvještaj sa svim dohvaćenim statusima navedenog korisnika.

Kako to izgleda kod agenta izvjestitelja, moguće je vidjeti slici 4.7. Moguće je vidjeti početak rada agenta, što zapravo predstavlja registraciju Twitter agenta kod agenta izvjestitelja čime se povećav broj trenutnih aktivnih agenata. U svakom od perioda šalju se dohvaćeni podaci agentu izvjestitelju koji te podatke ispisuje te ih sprema u rječnik. Podaci su kod agenta izvjestitelja tako dugo dok Twitter agent dohvaća podatke.

Kada Twitter agent završi s dohvaćanjem, zatraži od agenta izvjestitelja da mu pošalje izvještaj sa svim podacima. Upravo ovdje se vidi da je agent završio s radom te se nakon slanja izvještaja broj aktivnih agenata smanjuje za jedan.

Ovime je završena demonstracija Twitter agenta, gdje su prikazana dva načina rada i to; dohvaćanje statusa na temelju hashtag-a od strane svih korisnika društvene mreže te dohvaćanje statusa pojedinog korisnika na temelju unesenog korisničkog imena od strane korisnika.

Moguće je uz navedene načine pokretanja programa još i promijeniti vrijeme trajanja dohvaćanja podataka, vrijeme trajanja jedne periode pa sve do promjene naziva konfiguracijske datoteke ili datoteke s podacima za spajanje na društvenu mrežu.

```
[reporter@127.0.0.1] New agent (twitter_230129@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] Received message from: twitter_230129@127.0.0.1
network=Twitter; type=tweet; keyword=nikolamajcen
username=nikolamajcen; name=Nikola
date=23:02:22, 10.01.2017
message=Testing VAS project :) #VAS #FOI

[reporter@127.0.0.1] Received message from: twitter_230129@127.0.0.1
network=Twitter; type=tweet; keyword=nikolamajcen
username=nikolamajcen; name=Nikola
date=23:02:50, 10.01.2017
message=Closing Twitter for now :)

[reporter@127.0.0.1] Agent (twitter_230129@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 0
```

Slika 4.7: Dohvaćanje statusa od strane agenta izvjestitelja

Sljedeći agent je Facebook agent, koji dohvaća statuse ili slike sa profila korisnika. Važno je napomenuti da u ovom slučaju to radi sa profilima koji su javno dostupni, a ne i sa privatnim profilima korisnika na društvenoj mreži Facebook.

Kod pokretanja ovog agenta iskoristio sam mogućnost promjene opcionalnih parametara, tako da je promijenjeno ukupno vrijeme dohvaćanja odnosno argument `time` i vrijeme trajanja jedne periode odnosno argument `period`. Uz navedene argumente, obavezno je i unijeti korisničko ime profila koji se želi pratiti.

Agent radi na isti način kao i Twitter agent, tako da dohvaća podatke u zadanih periodima, te ih u svakom periodu šalje agentu izvjestitelju ukoliko su podaci dohvaćeni.

Na kraju rada agenta, šalje se zahtjev za izvještajem, a agent izvjestitelj šalje izvještaj s podacima ukoliko su podaci dohvaćeni. U ovom slučaju nisu dohvaćeni nikakvi podaci u smislu statusa ili slika na navedenom profilu, tako da izvješće ne sadrži nikakve podatke, već se u izvješću nalazi samo broj dohvaćenih podataka, što je u ovom slučaju nula (0).

```
nikolamajcen@ubuntu:~/Development/Python/social-notifier$ python social-notifier.py facebook -time 60 -period 10 kimkardashian
[facebook_232036@127.0.0.1] Facebook fetch agent is starting...
[facebook_232036@127.0.0.1] Status activity fetch started.
[facebook_232036@127.0.0.1] Fetching statuses... (No. of periods left: 5)
[facebook_232036@127.0.0.1] Found statuses: 0
[facebook_232036@127.0.0.1] Fetching statuses... (No. of periods left: 4)
[facebook_232036@127.0.0.1] Found statuses: 0
[facebook_232036@127.0.0.1] Fetching statuses... (No. of periods left: 3)
[facebook_232036@127.0.0.1] Found statuses: 0
[facebook_232036@127.0.0.1] Fetching statuses... (No. of periods left: 2)
[facebook_232036@127.0.0.1] Found statuses: 0
[facebook_232036@127.0.0.1] Fetching statuses... (No. of periods left: 1)
[facebook_232036@127.0.0.1] Found statuses: 0
[facebook_232036@127.0.0.1] Fetching statuses... (No. of periods left: 0)
[facebook_232036@127.0.0.1] Found statuses: 0
[facebook_232036@127.0.0.1] Status activity fetch ended.

[facebook_232036@127.0.0.1] Received message from: reporter@127.0.0.1
[facebook_232036@127.0.0.1] Total number of fetched data: 0
```

Slika 4.8: Pokretanje Facebook agenta

Vrijedi još prikazati rad agenta izvjestitelja sa više agenata, odnosno prikazati kako se prikazuje rad s više agenata i njihovih promjena u dohvaćanju podataka. Ovo je vrlo važna odlika rada agenta, jer je vrlo važno omogućiti da više agenata mogu istovremeno raditi sa agentom izvjestiteljem.

Na slici 4.9 može se vidjeti primjer rada agenta izvjestitelja sa više agenata koji dohvaćaju podatke s društvenih mreža.

```
[reporter@127.0.0.1] Agent (twitter_230129@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 0
[reporter@127.0.0.1] New agent (facebook_231457@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] New agent (facebook_231555@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 2
[reporter@127.0.0.1] Agent (facebook_231457@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] Agent (facebook_231555@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 0
[reporter@127.0.0.1] New agent (facebook_231800@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] Agent (facebook_231800@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 0
[reporter@127.0.0.1] New agent (facebook_231808@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] Agent (facebook_231808@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 0
[reporter@127.0.0.1] New agent (facebook_231829@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] New agent (facebook_231936@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 2
[reporter@127.0.0.1] Agent (facebook_231936@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] New agent (facebook_231954@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 2
[reporter@127.0.0.1] Agent (facebook_231954@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] New agent (facebook_232000@127.0.0.1) started fetching.
[reporter@127.0.0.1] No. of agents working: 2
[reporter@127.0.0.1] Agent (facebook_232000@127.0.0.1) stopped fetching.
[reporter@127.0.0.1] No. of agents working: 1
[reporter@127.0.0.1] New agent (facebook_232036@127.0.0.1) started fetching.
```

Slika 4.9: Prikaz rada agenta izvjestitelja s više agenata

Poglavlje 5

Zaključak

U ovom seminarskom radu predstavio sam višeagentne sustave u svrhu prikupljanja podataka sa društvenih mreža. Društvene mreže predstavljaju izvore velikih količina podataka iz kojih se uz određene analize mogu dobiti potrebne informacije, bilo da li se radi o raznim događanjima, promjenama pa i potrebnim informacijama za pojedinca ili organizaciju.

Višeagentni sustavi jedan su od načina na koji se može dizajnirati sustav kao skup komponenta, gdje agenti kao reaktivni agenti predstavljaju pojedini proces u cjelokupnom radu sustava.

Predstavio sam jednostavan primjer prikupljanja podataka sa društvenih mreža Facebook i Twitter, gdje su agenti u međusobnoj komunikaciji između sebe te sa vanjskim svijetom, gdje dohvaćaju specificirane podatke.

Iako je sustav vrlo jednostavan, vrlo je lako uvidjeti da se od jednostavnih komponenti može doći do specifičnih podataka koji daljnim razvojem komponenti odnosno agenata može doprinjeti daljnjom analizom podataka do određenih informacija koje su važne korisniku ili organizaciji te mogu prikazati promjene kod korisnika ili na cjelokupnoj društvenoj mreži.

Bibliografija

Catholijn M. Jonker, J.T., 2002. *Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness*. International Journal of Cooperative Information Systems.

G. Aranda, J.P., 2012. *SPADE User's Manual*. Available at `https://pythonhosted.org/SPADE`.

Lerman, K., 2008. *Social Information Processing in Social News Aggregation*.

Schatten, M., 2016. *Materijali s predavanja*. Available at `https://elf.foi.hr/course/view.php?id=292`.