Zasnova splošnega ogrodja in podatkovnega modela za obdelavo naravnega jezika – ANGLEr

Slavko ŽITNIK
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko

Povzetek

S področjem analize naravnega jezika se danes ne ukvarjajo le strokovnjaki. Obdelava naravnega jezika se uporablja na najrazličnejših vsakodnevnih področjih, kot je na primer analiza sentimenta v ocenah izdelkov ali avtomatsko uvrščanje bančnih transakcij. Zaradi uspešnega razvoja področja v zadnjih letih bi vse več uporabnikov želelo analizirati besedila, vendar jim primanjkuje tehničnega znanja. V razdelku zato pregledamo obstoječa ogrodja za demokratizacijo obdelave naravnega jezika in predlagamo lastno splošno ogrodje ANGLEr, vključno s celostnim podatkovnim modelom. Ogrodje bi omogočilo raziskovalcem enostavno vključitev novih metod v ogrodje ali njihovo prototipiranje oz. uporabo v okviru predvidenega cevovoda. Tehnično nevešči uporabniki pa bi lahko preko grafičnega vmesnika dodajali komponente in izvajali analize besedil za svoje namene. Ključna prednost predlaganega ogrodja v primerjavi z obstoječimi je (a) enoten, razširljiv, hierarhičen, verzioniran, odprt, tehnološko agnostičen in enostavno razumliv podatkovni model ter (b) šibko sklopljeno ogrodje, temelječe na osnovi Docker vsebnikov in spletnih tehnologijah. Odprtokodno ogrodje ponuja tudi širše možnosti monetizacije storitev ali razvoj tržnice modulov. Vsekakor pa bi krovna organizacija oz. skupnost morala skrbeti za smer razvoja ogrodja in aktivno promocijo uporabe.

Ključne besede: ANGLEr, demokratizacija, obdelava naravnega jezika, ogrodje, podatkovni model

Abstract

Natural language processing is no longer confined to professionals only; its techniques are now used in various areas such as sentiment analysis in product review or automatic classification of bank transactions. Due to the latest advancements in the field, more users want to analyze texts but they lack technical knowledge. In this chapter we review existing frameworks for democratization of the natural language processing and propose a new general framework ANGLEr along with a comprehensive data model. The framework would enable researchers that develop new techniques to easily incorporate their methods into the framework, prototype them or use them within a larger workflow. Technically unsavvy users on the other side would use a user interface, add components and process texts for their needs. The key advantages of the proposed framework compared to existing ones are (a) a unified, extensible, hierarchical, versioned, open, technologically agnostic and easily comprehensible data model, and (b) a loosely-coupled framework based on Docker containers and Web technologies. The opensource framework offers multiple possibilities of monetization and development of a marketplace of additional modules. We strongly believe that one organization or community would need to take care of future developments and promote the usage of the framework.

Keywords: ANGLEr, democratization, natural language processing, framework, data model

1 Uvod

V zadnjem času opažamo porast interesa za različna splošna ogrodja, ki omogočajo uporabo naprednih metod za obdelavo naravnega jezika za tehnično manj vešče uporabnike. Tehnično vešči uporabniki si lahko brez težav najdejo ustrezno orodje, v katerega lahko vključijo najboljše raziskovalne rezultate iz odprtokodnih orodij in jih uporabljajo za svoje namene. Primeri takšnih so na primer LangChain,[1] Hugging Face Enterprise Hub[2] in podobni. Za manj vešče

1 https://www.langchain.com
2 https://huggingface.co/enterprise

uporabnike so namenjene t.i. *low-code/no-code* platforme, ki omogočajo, da lahko s pomočjo vizualnih gradnikov zgradijo aplikacijo za lastne namene.[3] Takšne platforme pa so navadno preveč splošne, zaprte in ne omogočajo uporabe najnovejših raziskovalnih izsledkov. Nekje vmes obstajajo splošna ogrodja za obdelavo naravnega jezika, ki se razvijajo že vsaj od leta 2002, odkar je bilo ogrodje UIMA (Ferrucci in Lally, 2004) preneseno 6000-krat in orodje GATE (Cunningham, 2002) več kot 400.000-krat. Uporabniki obeh orodij so različnih profilov – raziskovalci, jezikoslovci, gospodarstvo ali razvijalci programske opreme. Za računalniško manj vešče uporabnike takšna orodja ponujajo grafični vmesnik, ki omogoča enostavno gradnjo cevovodov za obdelavo naravnega jezika.

Namen razdelka je pregledati ogrodja, ki omogočajo demokratizacijo uporabe metod umetne inteligence, natančneje metod obdelave naravnega jezika, pri čemer omogočajo raziskovalcem, da svoja orodja in modele enostavno vgradijo v takšna orodja in jih ponudijo v uporabo javnosti. Na podlagi pregleda predlagamo novo ogrodje ANGLEr – *A Next-Generation Natural Language Exploratory Framework*, ki bi raziskovalcem na področju obdelave naravnega jezika (ONJ) omogočilo enostavno implementacijo naprednih orodij v sistem, kjer bi lahko hitro pokazali praktično uporabo svojih metod in omogočili uporabo ostalim. Primerljiva ogrodja, ki so na voljo do sedaj, je bodisi težko splošno razširjati ali pa jih je težje uporabljati. V prispevku identificiramo ključne komponente takšnega ogrodja in predlagamo izboljšave glede na slabosti obstoječih ogrodij.

Glavni doprinosi predlaganega ogrodja ANGLEr so naslednji:

a) Razširljiva in porazdeljena arhitektura na osnovi vsebnikov Docker, ki omogoča enostavno uporabo in dodajanje orodij v sistem.
b) Splošen in odprt podatkovni model, ki omogoča shranjevanje podatkov in kompatibilnost z različnimi orodji.
c) Grafični vmesnik, ki pohitri proces izgradnje cevovoda obdelave naravnega jezika za uporabnike brez tehničnega znanja.

3 Primer no-code platforme za obdelavo besedil: https://monkeylearn.com

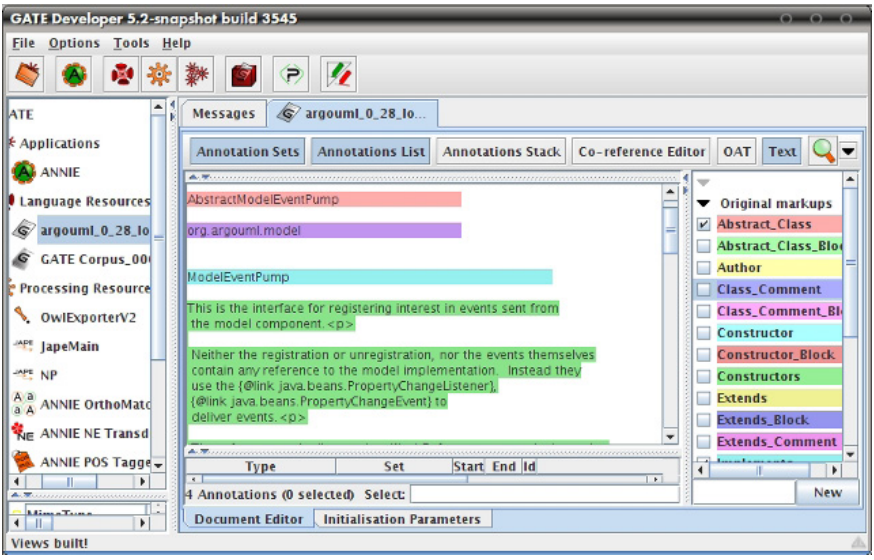
V nadaljevanju predstavimo obstoječa ogrodja in jih primerjamo med seboj (Razdelek 2). Posebej pregledamo uporabljene podatkovne modele in predlagamo lastnega (Razdelek 3). Na podlagi pregleda zasnujemo predlog novega splošnega ogrodja ANGLEr (Razdelek 4) in tudi predstavimo primer uporabe s pomočjo zaslonskih mask grafičnega vmesnika (Razdelek 5). Sklepne ugotovitve in ključna vprašanja za nadaljnji razvoj predstavimo v Razdelku 6.

2 Pregled obstoječih ogrodij

Na voljo je že kar nekaj razvitih ogrodij, ki omogočajo obdelavo besedil. Podrobneje smo pregledali obstoječa ogrodja in izbrali tista, ki so najbolj uporabna za uporabnike z omejenim tehničnim predznanjem. Analiza prednosti in slabosti posameznih nam je omogočila definirati seznam lastnosti, ki jih mora nasloviti novo ogrodje, kar prikazujemo v Tabeli 1. Orodja primerjamo predvsem glede na (a) grafični vmesnik, ki vpliva na uporabnost za nove uporabnike, (b) podatkovni model, ki vpliva na razširljivost, in (c) programski jezik ali okolje za razvoj novih vtičnikov, kar je pomembno za raziskovalce in razvijalce.

2.1 General Architecture for Text Engineering (GATE)

GATE (Cunningham, 2002) je eno izmed najstarejših in najbolj uporabljenih ogrodij za ONJ (Slika 1). Načrtovan je bil z namenom uporabe enotnega podatkovnega modela, ki naj bi podpiral širok nabor orodij za ONJ, vključno z grafičnim vmesnikom. GATE ponuja za delo in pripravo cevovodov za ONJ grafični vmesnik, ki pa je zastarel in ni bil posodobljen že več kot 10 let. Program je sicer enostavno namestiti, teče kot aplikacija Java na osebnem računalniku, vendar vmesnik ni intuitiven za nove uporabnike. Ogrodje omogoča razvoj dodatnih vtičnikov, ki pa morajo biti pripravljeni v programskem jeziku Java. Vsebuje tudi obstoječ nabor predpripravljenih vtičnikov, ki pa ne vsebujejo naprednejših metod. Zaradi vsega tega je ogrodje težko prilagoditi za uporabo novejših algoritmov.



Slika 1: Grafični vmesnik orodja GATE.

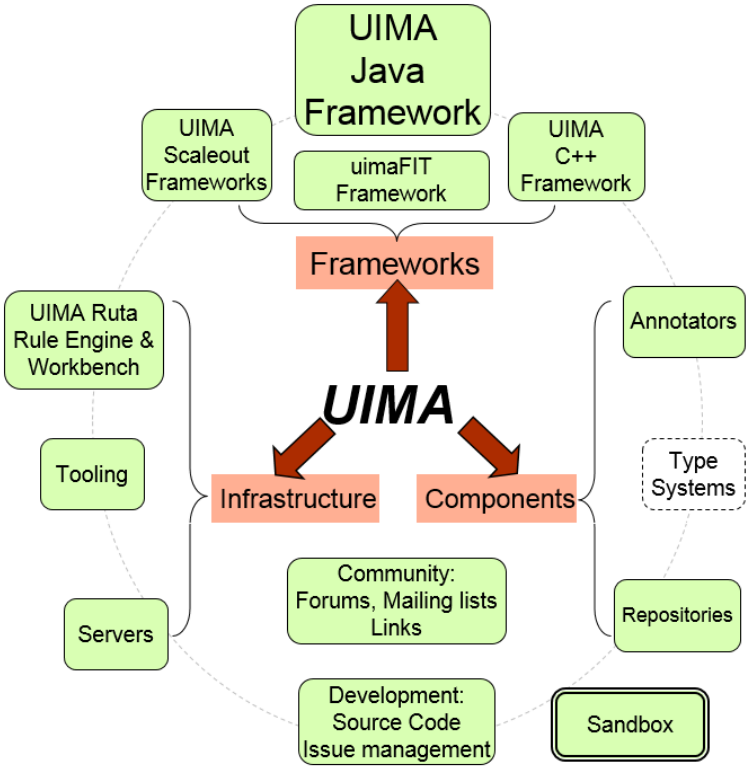
Orodje razvija Univerza Sheffield in podjetje OntoText. Skupaj ponujajo nadgrajeno ogrodje kot oblačno storitev in iskalnik GATE Mimir. Prav tako imajo objavljen seznam vtičnikov, ki so v tej storitvi na voljo.[4]

2.2 Unstructured Information Management Applications (UIMA)

UIMA (Ferrucci in Lally, 2004) je splošno ogrodje za ekstrakcijo informacij iz nestrukturiranih dokumentov, kot so besedila, slike, e-pošta in podobno (Slika 2). Je odprtokodni Apache projekt s standardizirano implementacijo tehničnega standarda UIMA, ki ga je pripravila standardizacijska organizacija OASIS. Vključuje tudi splošen podatkovni model, ki je sposoben hraniti podatke vseh razvitih komponent. Ogrodje ponuja tudi grafični vmesnik za nekatere splošne komponente. Vmesnik ni del enotne aplikacije in ne omogoča enostavnega načina kombiniranja orodij med seboj. Primarni način uporabe ogrodja še vedno zahteva urejanje XML opisov za označevanje

4 https://cloud.gate.ac.uk/shopfront

dokumentov, kar omejuje uporabnost za nove uporabnike in upočasnjuje razvoj. Nove komponente so lahko razvite v programskem jeziku C++ ali Java. Ogrodje pa omogoča, da so aplikacije UIMA razvite kot ločene komponente, kot na primer »identifikacija jezika« → »jezikovnoodvisno razčlenjevanje« → »prepoznavanje stavkov« → »prepoznavanje imenskih entitet.« Vsaka izmed komponent mora implementirati določene vmesnike ogrodja in definirati samoopisne metapodatke s pomočjo datotek XML. Namen ogrodja je, da upravlja s komponentami in med njimi posreduje podatke. Vsaka izmed komponent lahko teče na ločenem strežniku kot spletna storitev ali je replicirana na gruči strežnikov.



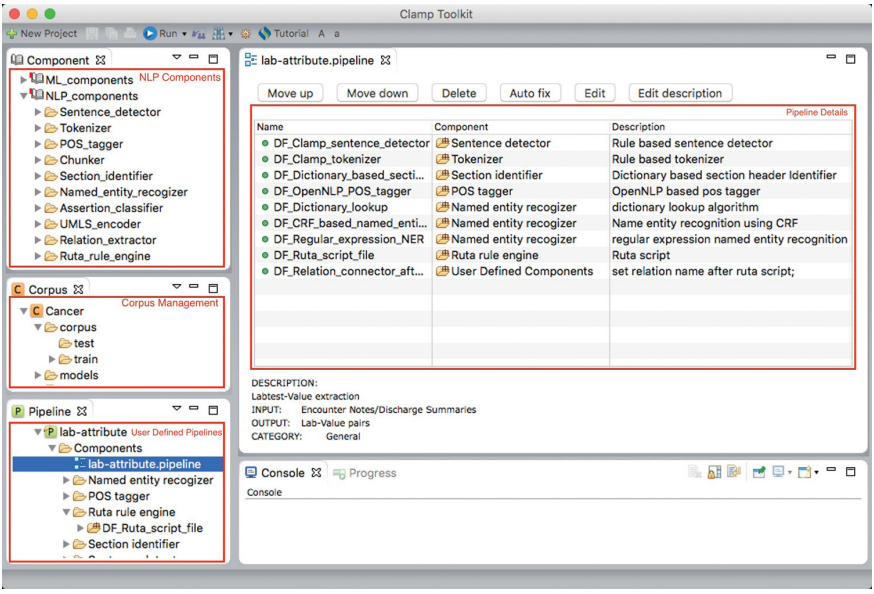
Slika 2: Tipi projektov v okolju UIMA.

Ogrodje UIMA omogoča konfiguracijo in zagon cevovodov za komponente za označevanje. Uporabniki lahko razvijajo svoje lastne

komponente ali nastavljajo in uporabljajo obstoječe. Nekaj komponent je na voljo kot del glavnega projekta, večina preostalih pa je dostopnih preko različnih repozitorijev na internetu. Platforma GitHub beleži več kot 900 repozitorijev, ki uporabljajo UIMA Java SDK. Novejše izvedbe med komponentami podpirajo tudi komunikacijo REST. Pomembno je omeniti, da je bil sistem IBM Watson, ki je osvojil tekmovanje Jeopardy v letu 2013, razvit na osnovi arhitekture UIMA.

2.2.1 Clinical Language Annotation, Modeling, and Processing (CLAMP)

Skupina raziskovalcev na področju obdelave biomedicinskih besedil (Soysal idr., 2018) je razvila orodje CLAMP, ki temelji na ogrodju UIMA. Orodje vključuje 10 namenskih UIMA komponent, kot na primer za prepoznavanje stavkov, kratic, imenskih entitet, povezav s shemo UMLS (Bodenreider, 2004) ali orodje za izvajanje pravil. Poleg tega so razvili tudi grafični vmesnik, ki temelji na ogrodju Eclipse (Slika 3). Orodje za njihove namene omogoča tudi označevanje besedil in

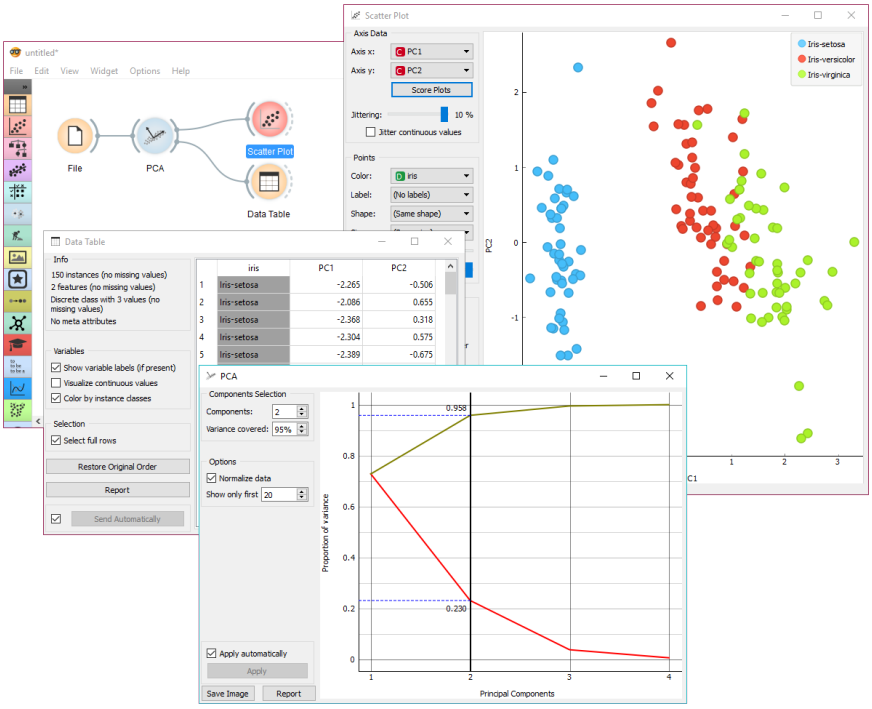


Slika 3: Prikaz grafičnega vmesnika CLAMP.

učenje modelov strojnega učenja. Ogrodje grafičnega vmesnika je zastarelo, težko razširljivo in po naših izkušnjah neintuitivno. Kljub temu je za začetnika bolj primerno kot vmesnik GATE. Celoten projekt je odprtokoden, vendar ga uporablja le manjša množica raziskovalcev na ožjem domenskem področju in tako ni prilagojen za splošno uporabo.

2.3 Orange - Data Mining Fruitful and Fun

Orodje Orange Data Mining (Demšar idr., 2013) se aktivno razvija in redno posodablja na Univerzi v Ljubljani in je eno izmed večkrat nagrajenih orodij za demokratizacijo umetne inteligence (Slika 4). Prihodnje izdaje predvidevajo tudi oblačno verzijo.



Slika 4: Primer uporabniškega vmesnika Orange.

Glavna lastnost orodja po presoji razvijalcev je intuitiven uporabniški vmesnik, primeren tudi za netehnične uporabnike. Orodje

ponuja velik nabor vtičnikov za naloge strojnega učenja in vizualizacije podatkov. Poleg tega lahko razvijalci uporabijo Orange kot programsko knjižnico. Orange je primarno namenjeno obdelavi relacijskih podatkov za klasično strojno učenje. Poleg tega pa ponuja dve razširitvi za obdelavo besedil (Razdelek 2.3.1 in 2.3.2). Največji problem pri ONJ v Orangu je tabelarična predstavitev podatkov, ki ni najbolj primerna za obdelavo besedil. Po drugi strani pa takšna predstavitev omogoča neposredno interoperabilnost z množico obstoječih orodij. Razvijalci lahko tudi razvijejo svoj vtičnik, vendar so pri tem precej vezani na programski jezik Python, specifično verzijo paketa Orange in arhitekturne značilnosti paketa, namesto šibko sklopljene komunikacije. V primeru šibke sklopljenosti bi lahko bili posamezni deli ogrodja razviti povsem ločeno med seboj, pri čemer bi jih povezoval le način medsebojne komunikacije, ki bi ga lahko vsaka komponenta implementirala na svoj način.

2.3.1 Vtičnik Orange text mining

Vtičnik Orange text mining (Slika 5) se lahko namesti neposredno preko uporabniškega vmesnika Orange. Vtičnik poleg tabelaričnega formata implementira podatkovna tipa Dokument in Korpus, vendar se zdi, da je tabelarični format še vedno glavni tip predstavitve podatkov. Na voljo pa je sicer tudi komponenta za pretvorbo podatkov med tipi. Vtičnik ponuja nekaj pogosto uporabljenih metod za predprocesiranje besedil (tokenizacija, lematizacija, filtriranje besed ipd.). Poleg tega vsebuje tudi nekaj naprednejših orodij, kot so *similarity hash* (pretvorba dokumentov v podobnostne vektorje), analiza sentimenta, določanje čustev v čivkih in podobno.[5]

5 https://orange3-text.readthedocs.io/en/latest

Slika 5: Vtičniki Orange text mining (levo) in Textable (desno).

2.3.2 Textable

Textable je ločena veja projekta Orange (Slika 5), ki jo razvija podjetje LangTech v sodelovanju z oddelkom za jezik in informacijske znanosti Univerze v Lausanne. Zdi se, da se ne posodablja redno.[6] Orodje definira uporabo tekstovnih datotek in ima podporo za datoteke tipa JSON in spletne URL vire. Verzija 3 podpira tudi različne nivoje besedilnih segmentov. Več različnih podatkovnih tipov sicer omogoča implementacijo več različnih algoritmov, vendar pa se zaradi tega izgubi velika mera kompatibilnosti med orodji, kar nas lahko omejuje pri gradnji kompleksnejših cevovodov za analizo besedil. S stališča podpore analizam orodje ponuja osnovno obdelavo besedil

6 http://textable.io

(npr. osnovno predprocesiranje, konkordance, kolokacije, matrike dokument-termin, lematizacijo ali oblikoslovno označevanje), povezavo z zunanjimi knjižnicami (npr. NLTK, Pattern, GenSim) in uvoz vsebin (npr. HTML, CSV, XML).

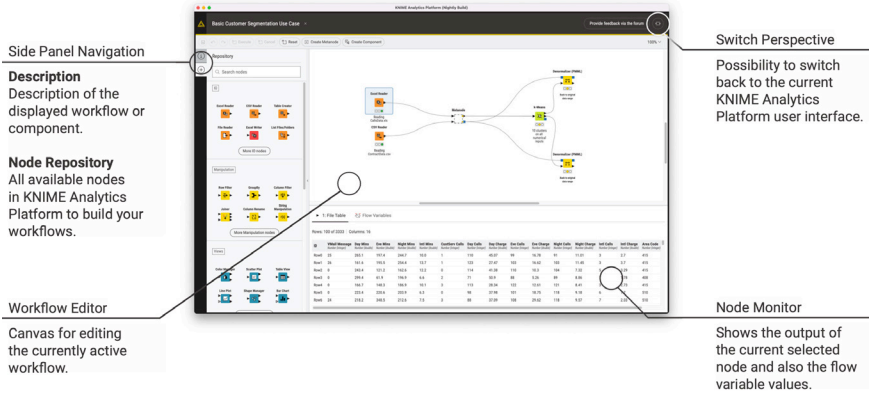
2.4 KNIME Analytics Platform

Platforma za podatkovne vede KNIME[7] je odprtokodno ogrodje, ki uporabnikom omogoča analizo in vizualizacijo podatkov brez programiranja. Njen *low/no-code* grafični vmesnik (Slika 6) ponuja množico orodij za začetnike in bolj napredne uporabnike. Platformo sicer razvija podjetje, ki prodaja tudi plačljive napredne storitve (npr. delo v oblaku, zasebni prostor, specializirane vtičnike, spletne storitve za integracijo z drugimi sistemi in podporo).

Platforma se zdi podobna orodju Orange, le da komponente komunicirajo preko spletnih storitev. Tržnica komponent vsebuje več tisoč orodij, ki jih je možno namestiti enostavno s *povleci-spusti* neposredno iz spletnega brskalnika. Komponente vsebujejo tudi najbolj napredne implementacije algoritmov (npr. uporaba modelov BERT ali pogovornih velikih jezikovnih modelov). Ravno tako kot v Orange lahko nekdo shrani postopek analize (angl. *workflow*) – tudi ti so na voljo na spletni tržnici, kjer jih je na voljo več kot 18 tisoč.

Platforma je na voljo kot namestitveni paket za več operacijskih sistemov in se namesti neposredno na osebni računalnik. Na voljo je tudi strežniška namestitev za napredne uporabnike. V primerjavi z novejšimi tehnologijami, platforma v ozadju uporablja ogrodje Eclipse, ki mu je potrebno slediti tudi pri razvoju lastnih komponent in ga obvezno uporabljati kot razvojno okolje. Osnovni programski jezik je tako Java. Navodila za razvijalce jasno definirajo strukturo projekta in tudi podatkovni model. Ogrodje kot osnovni tip podatkov predvideva *BufferedDataTable*, ki je splošna predstavitev tabelaričnih podatkov, ki jih lahko razvijalec prilagodi po lastnih željah.

7 https://www.knime.com



Slika 6: Grafični vmesnik platforme KNIME.

2.5 Programske knjižnice za obdelavo naravnega jezika

Obstaja mnogo programskih knjižnic za ONJ. Knjižnice načeloma omogočajo uporabo metod ONJ neposredno iz programskih jezikov in ne ponujajo grafičnih vmesnikov. Nekatere knjižnice se razvijajo že dalj časa, zelo popularne pa so nastale v zadnjih nekaj letih. Namenjene so predvsem ekspertom/razvijalcem in niso primerne za začetnike. Kljub temu predstavljajo pomemben del ekosistema, zato omenjamo nekaj glavnih.

Ena izmed bolj splošnih in starejših knjižnic je OpenNLP,[8] ki podpira razvoj aplikacij v programskem jeziku Java. Podobno obstaja kar nekaj knjižnic za programski jezik Python, npr. NLTK (Bird in Loper, 2004) ali Gobbli (Nance in Baumgartner, 2021), katere namen je poenostaviti uporabo globokega učenja za ONJ. Precej popularna je Stanza (Qi idr., 2020), naslednica tradicionalne knjižnice Stanford Core NLP (Manning idr., 2014), ki se razvija na Univerzi Stanford in podpira več različnih jezikov. V gospodarstvu večinoma tradicionalno uporabljajo Spacy,[9] ki vsebuje tudi kar nekaj novejših prednaučenih modelov.

Novejše knjižnice se osredotočajo predvsem na uporabo globokih nevronskih mrež in (velikih) jezikovnih modelov. Huggingface[10]

8 https://opennlp.apache.org
9 https://spacy.io
10 https://huggingface.co

se je začela razvijati kot knjižnica in sedaj predstavlja ekosistem, ki ponuja korpuse, prednaučene modele, knjižnico, repozitorij z gostovanjem in enostavno ogrodje *Spaces* za osnovno grafično preskušanje modelov. Podobno se knjižnica LangChain[11] usmerja na velike jezikovne modele, večinoma prilagojene za pogovore, pri čemer ločen projekt LangFlow[12] zagotavlja tudi grafične komponente za razvite modele.

Pisanje programov je precej počasnejše kot konstrukcija cevovodov preko grafičnega vmesnika, ki ima na voljo predpripravljene komponente. To je ena izmed ključnih funkcionalnosti za večjo uporabnost našega predlaganega ogrodja. Podobno pa se kaže tudi pri novejših programskih knjižnicah, ki želijo razvijalcem omogočiti čimhitrejše prototipiranje rešitev z uporabniki in zato ponujajo komponente za določen tip problemov ONJ.

2.6 Primerjava pregledanih ogrodij

Tabela 1 prikazuje primerjavo pregledanih ogrodij glede na izbrane dimenzije.

Tabela 1: Primerjava različnih ogrodij za obdelavo naravnega jezika.

Ogrodje	**Grafični vmesnik**	**Enotni podatkovni model**	**Programski jezik vtičnikov**
GATE	Da (OS)	Da (presplošen)	Java
UIMA	Da (Omejeno)	Da	Java ali C++
Orange /z vtičniki	Da (OS)	Različen (transformacije na voljo)	Python
KNIME	Da (OS)	Da (tabelaričen)	Java (Eclipse)
Knjižnice ONJ	Ne	Različno	Odvisno od knjižnice
ANGLEr	**Da (Splet)**	**Da (verzionirano)**	**Poljuben (Docker)**

Edini tip orodij, ki je najbolj različen od naših zahtev, so programske knjižnice. Vsa ostala štiri ogrodja so na voljo kot aplikacije,

11 https://www.langchain.com/
12 https://github.com/logspace-ai/langflow

ki jih je potrebno namestiti na operacijski sistem, medtem ko mi predlagamo spletno aplikacijo, ki teče v vsebnikih Docker in ne zahteva posebne namestitve. GATE, UIMA in KNIME omogočajo spletne storitve, ki se jih lahko namesti v oblačno infrastrukturo, vendar do neke mere zahtevajo uporabo predvidenega programskega jezika za izdelavo razširitev. V našem primeru predlagamo definicije programskih vmesnikov in enotnega podatkovnega modela za komunikacijo med posameznimi komponentami. Takšna arhitektura je tudi zelo šibko sklopljena in razširljiva in omogoča, da lahko vsakdo uporablja poljubno tehnologijo za razvoj dodatnih vtičnikov. Orange in KNIME ponujata prijazen uporabniški vmesnik z velikim naborom izdelanih komponent. Oba ponujata tudi možnost shranjevanja in deljenje cevovodov. KNIME uporablja ogromna odprtokodna skupnost, ki v ogrodje vključuje tudi najnovejše algoritme. Oba projekta se izvajata pod okriljem organizacij, ki že dalj časa vzdržujeta in posodabljata ogrodji. Slednje je predvsem pomembno, saj je potrebno ogrodja ves čas posodabljati, da delujejo z aktualnimi sistemi, zaradi česar lahko takšni projekti ostajajo aktivni.

3 Podatkovni modeli

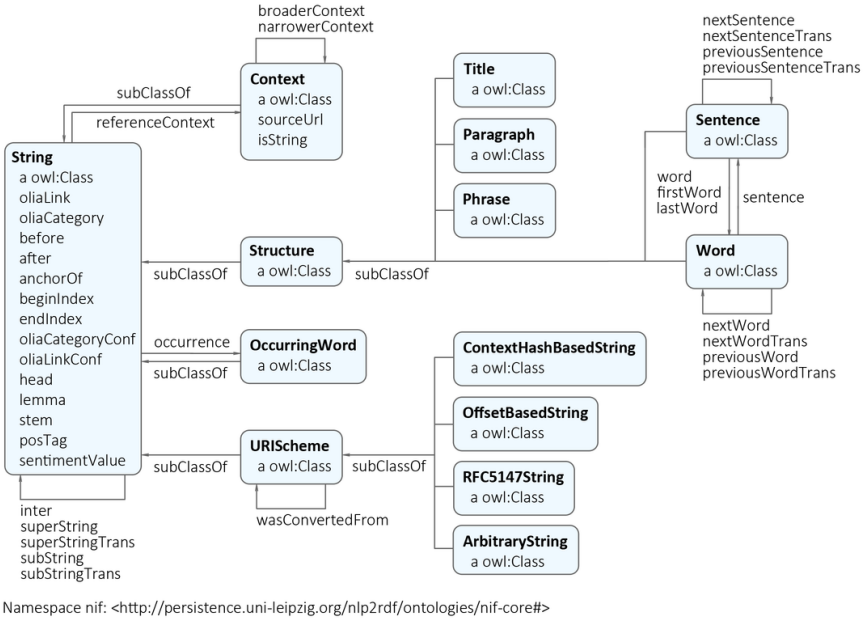
Podatkovni model omogoča predstavitev podatkov v določeni obliki. Podobno so definirane (ali standardizirane) predstavitve oznak v učnih korpusih (npr. TEI XML), ki omogočajo, da si lahko širša množica raziskovalcev med seboj deli podatke in jih razume. Podobno je pri razvoju programske opreme, saj želimo doseči, da bi algoritmi lahko »razumeli« vhode in izhode med seboj, brez da bi morali razvijalci prilagajati izhode predhodnih algoritmov v njihove ustreznice za vhode za druge algoritme. Zaradi tega morajo biti podatkovni modeli čimbolj splošni, razširljivi, razumljivi in enostavni za uporabo, kar pa je morda nemogoče doseči.

V nadaljevanju predstavimo podatkovne modele predhodno pregledanih ogrodij in standardiziran model NIF ter na podlagi ugotovitev zasnujemo podatkovni model ANGLEr. Poleg omenjenih obstaja še nekaj standardov, kot sta na primer CDA+GrAF (Meystre idr., 2012) ali

International Standard for a Linguistic Annotation Framework (Ide in Romary, 2004).[13] Prvi združuje dva standarda (ONJ in analiza grafov), drugi pa predstavlja mednarodno sprejet standard ISO. Ker nista širše uporabljena v praksi, jih ne omenjamo posebej.

3.1 Podatkovni model NLP Interchange Format (NIF)

Podatkovni model NLP Interchange Format (NIF; Hellmann idr., 2012) je nastal z namenom, da predstavi alternativo centraliziranim rešitvam (UIMA, GATE) in omogoči razvoj raznovrstnih, porazdeljenih in šibko sklopljenih aplikacij ONJ, ki sodelujejo med seboj. Verzija NIF 2.0 (Hellmann idr., 2013) definira semantično shemo v obliki ontologije RDF/OWL, s katero je možno opisovati posamezne označene dele besedila oz. prepoznane koncepte. Model se nanaša predvsem na fiksne predstavitve v besedilih, kot so naslov, odstavek, besedna zveza, stavek ali beseda (Slika 7).



Slika 7: Predstavitev podatkovnega modela (sheme ontologije) NIF 2.0.

13 https://www.cs.vassar.edu/~ide/papers/ISO+24612-2012.pdf

Jedro ontologije NIF predstavljajo viri *Strings as RDF*, katerih namen je, da označujejo izbrane dele besedila. Naslavljanje delov (tj. zaporedja črk) je implementirano v okviru definicije naslova URI posameznega objekta. Shemo je uporabljalo nekaj projektov na področju ekstrakcije informacij (npr. prepoznavanje imenskih entitet), standardizirana in vključena je bila v okviru projektov W3C ter uporabljena v nekaj delih izzven skupnosti (Sherif, 2014).

Podatkovni model se očitno ne razvija in se je nekako ustavil pri podpori za osnovna orodja.[14] To je možno zaradi več razlogov: (a) skupnost ni širše sprejela modela, (b) model je presplošen in ne omogoča širšega nabora predstavitve podatkov ONJ brez lastnih razširitev ali (c) raziskovalci ne poznajo semantičnih predstavitev RDF. Zagotovo bi moralo biti podprtih več predstavitev ONJ, vendar je verjetno na manjšo sprejetost najbolj vplivala »zima« semantičnega spleta. Raziskovalno področje semantičnega spleta namreč po 2013 ni več beležilo velikega zanimanja, kar se v zadnjih letih spreminja. S pojavom WikiData, smernic Evropske komisije glede podatkovnih prostorov in interoperabilnosti so opisi podatkov s pomočjo ontologij zelo uporabni za takšne naloge. Trenutno (v letu 2023) se prijavlja tudi akcija COST, ki bi nadaljevala razvoj semantičnih standardov za ONJ. Čeprav so funkcionalnosti semantičnega spleta zagotovo dodana vrednosti oznakam ONJ, lahko poleg oznak predstavljajo nepotrebno režijo. Zaradi tega pri podatkovnem modelu ANGLEr predlagamo predstavitev, ki se lahko enostavno poveže s semantično shemo.

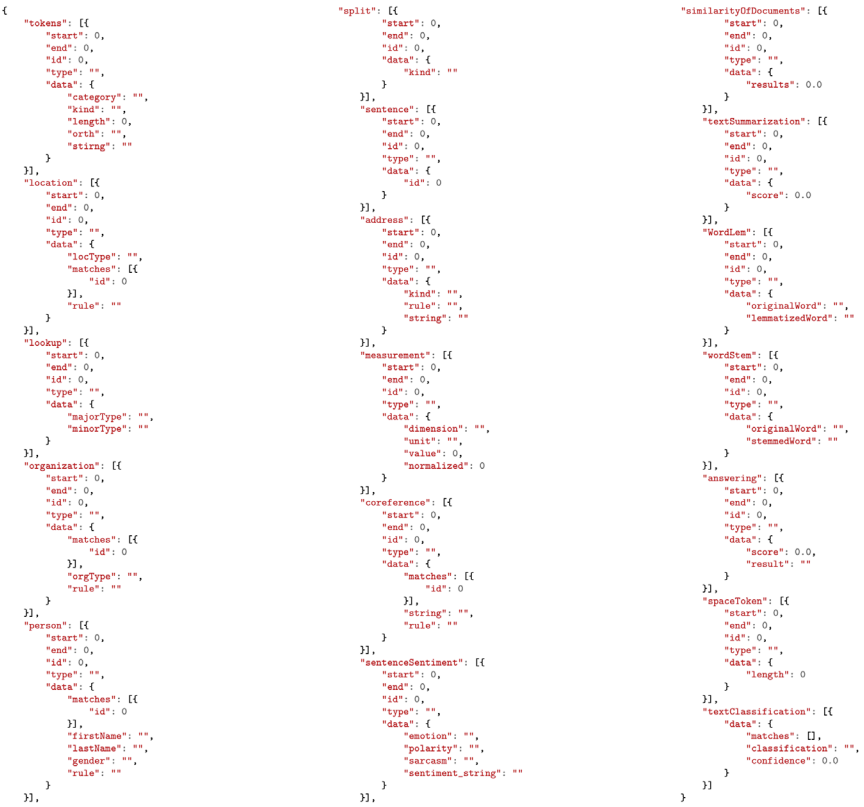
3.2 Podatkovni model GATE

Orodje GATE implementira podatkovni model v razredih programskega jezika Java, pri čemer je najvišje v hierarhiji tip razred Korpus (*Corpora*[15]). Primeri množice dokumentov, ki lahko predstavljajo korpus, so lahko tipov *TikaFormat*, *JsonDocuments*, *EmailDocuments*, *UIMADocument* ali *XMLDocument*. Vsak tip dokumenta mora omogočati tudi podporo za serializacijo v in iz zapisa GATE XML.

14 https://github.com/NLP2RDF
15 https://jenkins.gate.ac.uk/job/gate-core/javadoc/index.html

Predstavitve dokumentov so lahko pretvorjene v poljuben interni format, ki ga uporabljajo algoritmi v ogrodju GATE in se delijo na (a) vsebino, (b) oznake in (c) značilke. Vsaka oznaka, ki lahko vsebuje poljubno število značilk, je definirana z začetno točko, končno točko in tipom. Točka (*Node*) je definirana z unikatnim id-jem in odmikom v besedilu.

Poleg preddefiniranih predstavitev lahko vsakdo izdela svojo shemo[16] ali uporabi obstoječo. Slika 8 prikazuje preddefinirane tipe oznak. Menimo, da se veliko lastnosti med oznakami ponavlja in bi lahko bile organizirane bolj po skupinah/tipih in/ali hierarhično. Vsaka od oznak vsebuje tudi dodatne specifične atribute v obliki



```
{
    "tokens": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "category": "",
            "kind": "",
            "length": 0,
            "orth": "",
            "stirng": ""
        }
    }],
    "location": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "locType": "",
            "matches": [{
                "id": 0
            }],
            "rule": ""
        }
    }],
    "lookup": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "majorType": "",
            "minorType": ""
        }
    }],
    "organization": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "matches": [{
                "id": 0
            }],
            "orgType": "",
            "rule": ""
        }
    }],
    "person": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "matches": [{
                "id": 0
            }],
            "firstName": "",
            "lastName": "",
            "gender": "",
            "rule": ""
        }
    }],
    "split": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "data": {
            "kind": ""
        }
    }],
    "sentence": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "id": 0
        }
    }],
    "address": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "kind": "",
            "rule": "",
            "string": ""
        }
    }],
    "measurement": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "dimension": "",
            "unit": "",
            "value": 0,
            "normalized": 0
        }
    }],
    "coreference": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "matches": [{
                "id": 0
            }],
            "string": "",
            "rule": ""
        }
    }],
    "sentenceSentiment": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "emotion": "",
            "polarity": "",
            "sarcasm": "",
            "sentiment_string": ""
        }
    }],
    "similarityOfDocuments": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "results": 0.0
        }
    }],
    "textSummarization": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "score": 0.0
        }
    }],
    "WordLem": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "originalWord": "",
            "lemmatizedWord": ""
        }
    }],
    "wordStem": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "originalWord": "",
            "stemmedWord": ""
        }
    }],
    "answering": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "score": 0.0,
            "result": ""
        }
    }],
    "spaceToken": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "length": 0
        }
    }],
    "textClassification": [{
        "data": {
            "matches": [],
            "classification": "",
            "confidence": 0.0
        }
    }]
}
```


Slika 8: Predstavitev podatkovnega modela GATE v formatu JSON. Model je iz zapisa v razredih Java izluščil Nik Hrovat (2022).

16 https://gate.ac.uk/sale/tao/splitch5.html#x8-840005.3

slovarja (*data*), pri čemer so predvidene specifične oznake za vsak tip posebej, dodatne pa se lahko dodaja poljubno.

3.3 Podatkovni model UIMA

UIMA uporablja standardizirano predstavitev, ki jo je pripravila standardizacijska skupina OASIS. Zadnja verzija standarda je bila objavljena v letu 2008, pri čemer je UIMA 1.0 dosegel skladnost s standardom v letu 2009. V primerjavi z GATE je opis predstavitev precej splošen in zelo kompleksen za nekoga, ki npr. razvija orodje ONJ in bi želel, da je njegov algoritem kompatibilen s podatkovnim modelom. Poleg tega specifikacija definira tudi komunikacijski protokol in procese za sistem ONJ.

UIMA zelo specifično definira vse vidike ONJ (v formatu XML), kar je seveda zelo primerno za veliko ogrodje. Kljub temu so razvite komponente precej stare, pri čemer se nekateri projekti trudijo približati


```
{
    "tokens": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "name": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "personTitle": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "kind": ""
        }
    }],
    "government": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "email": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "sentence": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "location": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "kind": ""
        }
    }],
    "link": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "type": "",
            "string": ""
        }
    }],
    "coreference": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "matches": [{
                "id": 0
            }],
            "score": 0.0
        }
    }],
    "documentsSimilarity": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "documentId": 0,
            "result": 0.0
        }
    }],
    "summarization": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "sentences": []
        }
    }],
    "stemmer": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "end": 0,
        "data": {
            "original": "",
            "stemmed_word": ""
        }
    }],
    "sentenceSentiment": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "emotion_name": "",
            "sentiment_string": ""
        }
    }],
    "classification": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "topic": "",
            "confidance": 0.0
        }
    }],
    "lemma": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "original": "",
            "lemmatized_word": ""
        }
    }],
    "answer": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "result": 0.0,
            "sentence": ""
        }
    }]
}
```


Slika 9: Predstavitev podatkovnega modela UIMA v formatu JSON. Model je iz implementacije izluščil Nik Hrovat (2022).

UIMA bližje ciljnim uporabnikom (npr. uimaFIT[17]). Komponente so razvite ločeno in vsaka zahteva specifičen postopek namestitve. Verjetno so tudi zaradi tega (in podpore novejšim modelom) trenutno bolj popularne platforme, kot so HuggingFace, LangChain in podobne.

3.4 Podatkovni model Orange

Osnovni tip predstavitve podatkov v ogrodju Orange so datoteke tipa *.tab*, kar predstavlja tabelaričen format za standardne naloge strojnega učenja. Za namene ONJ vtičnik za delo z besedili uporablja enak format, kjer vsaka vrstica predstavlja dokument obdelave z različnimi lastnostmi, kot so besedilo, razred in podobno. Uporabnik lahko sicer v sistem uvozi tudi podatke iz preddefiniranih formatov, kot so *.txt*, *.docx*, *.odt*, *.pdf*, *.xml* ali *.conllu*. Korpus besedil pa se lahko interaktivno ustvari tudi v orodju. Enaka funkcionalnost je na voljo tudi v Textable. Poleg osnovnih vtičnikov ta omogoča tudi uvoz podatkov iz nekaterih javnih virov – The Guardian, NY Times, Pubmed, Twitter in Wikipedija.

Rezultati vtičnikov so vrnjeni kot značilke v dodatnih stolpcih. Na primer, vtičnik za vektorske vložitve doda stolpec [*embedding-length*] za podatke o vsakem dokumentu. Nekateri vtičniki pa znotraj posameznega stolpca definirajo lastne predstavitve. Na primer, vtičnik *vreča besed* ustvari stolpec, ki vključuje besede in njihove frekvence v sledeči obliki: »beseda1=3, beseda2=123, beseda3=66, ...« Za transformacijo podatkov se lahko izdela nov vtičnik, ki se ga lahko nato vključi v cevovod obdelave, kar omogoči uporabo ogromnega števila že razvitih vtičnikov na področju strojnega učenja.

Za izdelavo popolnoma drugačne predstavitve pa lahko razvijalci razširijo Python programski razred *Orange.data.Storage* in definirajo lastno predstavitev podatkov.

3.5 Podatkovni model KNIME

Podobno kot orodje Orange (Razdelek 3.4) tudi orodje KNIME definira tabelarično strukturo kot osnoven tip predstavitve

17 https://github.com/apache/uima-uimafit

podatkov.[18] Uporablja se *BufferedDataTable,* ki je razred v programskem jeziku Java. Vsak vtičnik lahko sprejme več takšnih tabel, ki niso nujno vse shranjene v pomnilniku, kar omogoči obdelavo večjih količin podatkov. Ogrodje ponuja nekaj tipov predstavitev podatkov (razred *DataCell*), vendar lahko vsakdo predstavitve razširi za namene lastne uporabe.

3.6 Podatkovni model Stanza

Skupina Stanford NLP Group ima dolgo tradicijo ponujanja orodij ONJ. Do nedavnega so večino svojih algoritmov ponujali preko ogrodja Stanford Core NLP (Manning idr., 2014), ki je kot osnovno predstavitev uporabljal hierarhično urejeno predstavitev v obliki slovarjev (*HashMap* v programskem jeziku Java). S pojavom širše uporabljanih knjižnic v programskem jeziku Python so svoje ogrodje napisali na novo – Stanza in predstavili enostavnejši podatkovni model.

Podatkovni model Stanza je zelo jasno predstavljen v njihovih navodilih za uporabo.[19] Podobno kot pri Core NLP je tudi tu model osnovan na slovarjih, ki lastnosti predstavljajo kot ključ-vrednost (Slika 10). Model se lahko enostavno razširja, vendar je jasno definiran za osnovne potrebe ekstrakcije informacij, kot je prepoznava imenskih entitet ali lematizacija.

```
// Document
{
    "text": "The raw text.",
    "sentences": [{Sentence1}, {Sentence2}, ...],
    "entities": [{Span1}, {Span2}, ...],
    "num_tokens": 51,
    "num_words": 23
}

//Sentence
{
    "doc": Document, //back-pointer
    "text":    "The raw text",
    "dependencies": [
        {"head w.": "go", "rel.": "subj", "dep. w.": "home"},
        ...
    ],
    "tokens": [{Token1}, {Token2}, ...],
    "words": [{Word1}, {Word2}, ...],
    "entities": [{Span1}, {Span2}, ...],
    "sentiment": "positive",
    "constituency":    {ParseTree}
}
```

```
//Token
{
    "id": {"start-idx": 1, "end-idx": 4},
    "text":    "The",
    "misc": "Custom annotation value",
    "words": [{Word1}, {Word2}, ...],
    "start_char": 24,
    "end_char": 43,
    "ner": "B-ORG"
}

//Word
{
    "id": 3,
    "text": "The",
    "lemma": "the",
    "upos": "NOUN",
    "xpos": "NNP",
    "feats": "Gender=Fem|Person=3",
    "head": 2, //syntactic head word
    "deprel": "nmod", //relation to head word
    "deps": "head-word + deprel",
    "misc": "custom value",
    "parent": {Token}
}
```

```
//Span
{
    "doc": {Document},
    "text": "The Batman",
    "tokens": [{Token1}, {Token2}, ...],
    "words": [{Word1}, {Word2}, ...],
    "type": "PERSON",
    "start_char": 21,
    "end_char": 43
}

//ParseTree
{
    "label": "Noun",
    "children": [{ParseTree1}, {ParseTree2}, {ParseTree3}, ...]
}
```

Slika 10: Predstavitev podatkovnega modela Stanza v formatu JSON.

18 https://docs.knime.com/latest/analytics_platform_new_node_quickstart_guide/index.html#_number_formatter_node_implementation

19 https://stanfordnlp.github.io/stanza/data_objects.html

3.7 Predlog podatkovnega modela ANGLEr

Glede podatkovnih modelov, ki smo jih predstavili v predhodnih razdelkih, menimo, da nekateri predstavijo zanimive rešitve, vendar niso splošni, razširljivi ali enostavni za uporabo. Na primer, GATE definira nekaj osnovnih tipov za osnovno ONJ, za naprednejše analize pa bi bilo v bistvu potrebno model narediti praktično od začetka v programskem jeziku Java. UIMA nasprotno sicer definira zelo splošno in kompleksno predstavitev na podlagi standardizacije, ki pa jo zaradi tega uporablja manj uporabnikov. Orange in KNIME pričakujeta uporabo tabelaričnega formata, ki je prisiljen način predstavitve podatkov zaradi zagotavljanja čim večje skladnosti z zgodovinskim razvojem vtičnikov. Stanza sicer predstavi jasen in razumljiv podatkovni model, vendar vključuje predstavitve le za osnovne metode ONJ.

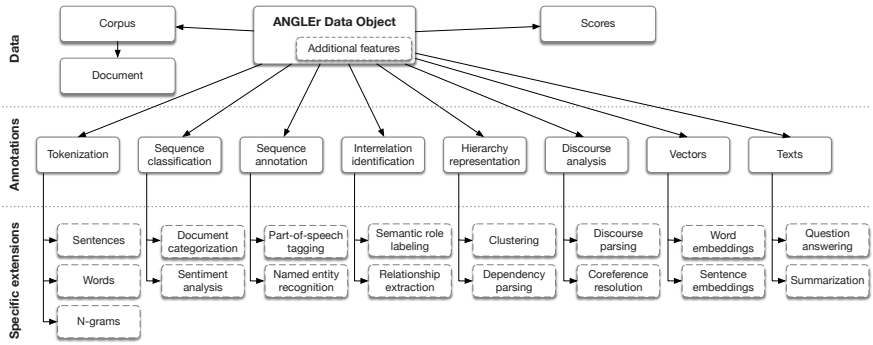
Cilj našega predloga je podpreti čim večje število orodij za analizo jezika. Poleg tega želimo pripraviti uporabniški vmesnik tako, da ga lahko razumejo tudi uporabniki, ki se ne spoznajo na princip delovanja programa. Uporabniški vmesnik mora biti sicer dovolj zmogljiv, da tudi naprednejšim uporabnikom omogoča enostavno in hitro prototipiranje. Tako lahko uporabniki hitro preizkusijo več različnih metod za analizo jezika in njihovih parametrov in takoj vidijo, kako spremembe vplivajo na pridobljene rezultate. Na podlagi pregleda obstoječih orodij smo ugotovili, da sta najbolj ključna elementa ogrodja za splošno ONJ podatkovni model in uporabniški vmesnik. Dober podatkovni model nam mora omogočati, da podpira predstavitve čim več metod ONJ, ki jih lahko zaradi tega enostavneje vključujemo v cevovode obdelav.

Menimo, da je pri snovanju podatkovnega modela potrebno definirati minimalne nabore predstavitev za posamezne naloge ONJ, ki jih sicer lahko specifične metode razširijo. Hierarhična struktura modela pa prispeva k lažji preglednosti, razumevanju in razširjanju modela. Glede na nam trenutno znane metode ONJ in uporabo v praksi, menimo, da bi zgornji nivo podatkovnega modela moral podpirati naslednje kategorije predstavitev podatkov:

- **Korpus**: Kot v večini modelov bi korpus moral vsebovati dokumente, ki so nižje-nivojska predstavitev podatkov različnih tipov, dolžin in metapodatkov. Vsak dokument pa bi vseboval določeno besedilo.
- **Razčlenjevanje**: Besedilni podatki so pogosto predstavljeni kot stavki, besede, besedne zveze, n-grami, ... Takšen tip bi moral podpirati vse različne členitve dokumentov.
- **Uvrščanje zaporedij**: Menimo, da je zaporedje splošen termin, ki predstavlja seznam členov, besed, stavkov, ... Primeri nalog, ki uporabljajo takšne predstavitve, so na primer analiza sentimenta ali kategorizacija besedil. Pri takšnih nalogah pa je ključno vedeti vsaj oznako uvrščanja.
- **Označevanje zaporedij**: Ta tip bi moral omogočati množico oznak za vsak del izbrane razčlenitve dokumentov. Primeri nalog so na primer oblikoslovno označevanje ali prepoznavanje imenskih entitet.
- **Prepoznavanje povezav**: Takšen tip bi moral omogočati hranjenje različnih tipov povezav med dvema ali več objekti. Uporabljal bi se na primer za podobnost med dokumenti, označevanju udeleženskih vlog ali prepoznavanju semantičnih relacij (»Janez« → »živi v« → »Boston«).
- **Analiza diskurza**: Tip bi moral omogočati skupno naslavljanje različnih predstavitev, kot je na primer zaporedje omenitev (tj. specifičen tip razčlenjevanja) za odkrivanje koreferenčnosti. Vsako zaporedje bi definiralo specifično oznako, ki bi veljala za izbrane objekte.
- **Hierarhična predstavitev**: Poleg osnovne predstavitve nekatera orodja ustvarjajo lastne hierarhične predstavitve znotraj izbranega besedila. Primer je na primer (plitko) označevanje drevesnic (angl. *dependency parsing*) ali gručenje podatkov.
- **Besedila**: Osnovna predstavitev podatkov, ki bi bila namenjena surovi predstavitvi ali generiranim rezultatom analiz. Dokument lahko sestoji iz različnih tipov besedil. Pri odgovarjanju na vprašanja so to na primer besedilo, vprašanje ali odgovor; pri prevodih besedila v različnih jezikih ali pri povzemanju različni povzetki.

- **Vektorji**: V času uporabe vektorskih predstavitev je potrebno zagotoviti, da so lahko različne razčlenitve besedil (tj. dokumenti, stavki, besede) predstavljeni kot vektorji.
- **Ocene**: Rezultati vrednotenj specifičnih algoritmov za namene hranjenja rezultatov analiz, vključno z metapodatki. Oznake, generirana besedila ali drugi rezultati v predstavitvi podatkov bi se referencirali na določen objekt ocen. Tako bi se omogočilo, da podatkovni model hrani osnovni korpus z oznakami in oznake več različnih algoritmov za iste naloge (npr. tri označevanja imenskih entitet treh različnih algoritmov ali njihovih različnih konfiguracij).
- **Dodatne lastnosti**: Vsak podatkovni tip bi moral poleg zahtevanih lastnosti omogočati še možnost hranjenja dodatnih lastnosti (ključ/vrednost). Takšne metapodatke bi lahko uporabljali algoritmi (npr. čas označevanja, ime algoritma). Poleg tega bi se po širši sprejetosti dodatnih lastnosti lahko le-te vključevalo kot obvezne za izpeljanke določenih tipov v naslednjih verzijah podatkovnega modela.

Na podlagi zgornjih definicij na Sliki 11 predstavljamo splošen podatkovni model ANGLEr. Krovni objekt vsebuje (a) korpus, (b) ocene in (c) visokonivojske objekte označevanj. Slednji objekti so hierarhični na način, da njihove izvedenke podedujejo obvezne lastnosti in dodatno uvajajo svoje. To omogoča interoperabilnost med različnimi



Slika 11: Hierarhija tipov podatkovnega modela ANGLEr.

metodami ONJ in različnimi verzijami podatkovnega modela. Kljub temu pa lahko katerokoli orodje obstoječim tipom dodaja še lastne metapodatke, ki so morda specifični le zanj (glej Dodatne lastnosti v zgornjem seznamu).

Slika 12 prikazuje praktičen primer strukture podatkovnega modela v formatu JSON. Različni deli so med seboj naslovljeni preko id-jev komponent. Nekatere komponente v podatkovnem modelu se lahko uporabijo neposredno, kot so definirane, ostale pa se lahko specializira glede na izbran algoritem. Na primer, predlogo za analizo diskurza se lahko razširi za namene odkrivanja koreferenčnosti ali analizo markerjev diskurza.

```
//ANGLEr Data Object
{
  "corpora": [{Corpus1}, {Corpus2}, ...],
  "scores": [{Scores1}, {Scores2}, ...],
  "annotations": [{Annotations1}, {Annotations2}, ...],
  "features": {"version": "1.0"} //Custom attributes
}

//Corpus
{
  "id": "1",
  "name": "Best NER-KB corpus",
  "documents": [{Document1}, {Document2}, ...],
  "features": {"url": "http://corpus-1.0.si"} //Custom attributes
}

//Document
{
  "id": "1-23", //Corpus id + Document id
  "text": "Best Story! Once upon a time, ...", //text used for analysis
  //Structured text parts that can be used by algorithms by key
  "text-parts": {
    "title": "Best Story!",
    "content": "Once upon a time, ...",
    "likes": 34
  }
  "features": {"length": 320, "separator": " "} //Custom attributes
}

//Scores
{
  "predicted_annotations_id": 42,
  "true_annotations_id": 31,
  "scores": {
    "F1": 0.45,
    "P": 0.88,
    "R": 0.91
  },
  "features": {"time": 2353242362} //Custom attributes
}
//Tokenization template
{
  "id": 33, //Used for algorithm input selection
  "annotation-type": "TOKENIZATION",
  "algorithm": "rule-based-1",
  "type": "SENTENCE", //WORDS, NGRAMS, WORD-PARTS, ...
  "documents": [
    {"doc_id": "1-23",
      "tokens": [{"id": 1,  "start_idx": 0, "end_idx": 342,
      "text": "Tralala, hopsasa."}, ...]
    }, ...
  ],
  "features": {"time": 2353242362} //Custom attributes
}
//Sequence classification template
{
  "id": 36, //Used for algorithm input selection
  "annotation-type": "SEQUENCE-CLASSIFICATION",
  "algorithm": "Best NN categorizer",
  "corpus_id": "1",
  "sequences": [
    {"id": 1, "class": "POSITIVE"}, ...
  ],
  "features": {"time": 2353242362} //Custom attributes
}
//Sequence annotation template
{
  "id": 39, //Used for algorithm input selection
  "annotation-type": "SEQUENCE-ANNOTATION",
  "algorithm": "CRF annotator",
  "tokenization_id": "1",
  "annotations": [
    {"doc_id": "1-23", "tags": ["O", "ORG", "O", ...]}, ...
  ],
  "features": {"annotation_time": "20sec"} //Custom attributes
}
```

```
//Interrelation identification template
{
  "id": 41, //Used for algorithm input selection
  "annotation-type": "INTERRELATION-IDENTIFICATION",
  "algorithm": "Intraverse relation extractor",
  "tokenization_id": "3",
  "relations": [
    {"relation": "employed_at", ...} //This object is specified on lower levels
  ],
  "features": {"annotation_time": "20sec"} //Custom attributes
}
//Hierarchy representation template
{
  "id": 43, //Used for algorithm input selection
  "annotation-type": "HIERARCHY-REPRESENTATION",
  "algorithm": "Agglomerative clusterer",
  "tokenization_id": null, //One of the following set only!
  "corpus_id": "1"
  "hierarchies": [
    {"parent_id": "34", "descendants": [{"id": "44", "relation": "nsubj"}, ...]}
    //This objects are specified on lower levels
  ],
  "features": {"max-hierarchy": 13} //Custom attributes
}
//Discourse analysis template
{
  "id": 45, //Used for algorithm input selection
  "annotation-type": "DISCOURSE-ANALYSIS",
  "algorithm": "Coref resolver SkipCor v2",
  "tokenization_id": 1, //These are sentences, mentions, ...
  "chains": [
    {"id1": "34", "id2": "45", "type": "anaphora"} //This object is specified on lower levels
  ],
  "features": {"singletons-num": 61} //Custom attributes
}
//Vectors template
{
  "id": 47, //Used for algorithm input selection
  "annotation-type": "VECTORS",
  "algorithm": "Doc2Vec",
  "tokenization_id": null, //One of the following set only!
  "corpus_id": "1"
  "vectors": [
    {"id": "78", "vec": [2.000, 3.241, 4.267, 5.987, ...]}
  ],
  "features": {"vector-dim": 350} //Custom attributes
}
//Texts template
{
  "id": 49, //Used for algorithm input selection
  "annotation-type": "TEXTS",
  "algorithm": "AnglerSummarizer",
  "corpus_id": "1",
  "doc-text-part-inputs": {
    "question": "question-part",
    "content": "text-part"
  }, //This object is specified on lower levels
  "texts": [
    {"doc-id": "78", "value": "No."}
  ],
  "features": null //Custom attributes
}
```

Slika 12: Predstavitev uporabe predloga podatkovnega modela ANGLEr v formatu JSON.

3.7.1 Verzioniranje podatkovnega modela

Hierarhična predstavitev podatkovnega modela izboljšuje obratno združljivost v primeru dodajanja novih tipov v model. V takem primeru so nove metode v primerjavi z obstoječimi združljive preko starševskih predstavitev podatkov. Prav tako se lahko zagotovi, da je nek objekt na nižjem nivoju lahko obdelan z algoritmom, ki deluje na višjenivojski predstavitvi.

Podatkovni model mora obstajati neodvisno od programskega paketa in biti na voljo v svojem odprtokodnem repozitoriju z jasno zgodovino sprememb. Tako se ga lahko uporablja v različnih programskih jezikih ali pa ga povzamejo tudi druga orodja. Verzioniranje mora slediti smernicam SemVer.[20]

4 Predlog arhitekture ogrodja ANGLEr

Med pregledom obstoječih ogrodij smo odkrili dobre in slabe prakse, o katerih smo lahko sklepali glede na sprejetost orodja v praksi. Namen arhitekture ANGLEr je zagotoviti razširljivo in skalabilno ogrodje, v katerega bi lahko napredni razvijalci (in raziskovalci) enostavno integrirali svoja orodja. To pomeni s čim manj branja dokumentacije o delovanju ostalih delov ogrodja. Netehnični uporabniki pa morajo imeti možnost enostavne namestitve in uporabe ogrodja (lahko tudi preko oblačne storitve). Poleg verzioniranega podatkovnega modela (Razdelek 3.7) menimo, da je pomembno zagotoviti (a) razširljivo arhitekturo, ki je šibko sklopljena preko programskih vmesnikov API, in (b) uporabniški vmesnik na podlagi vtičnikov.

Arhitektura sistema mora biti zasnovana tako, da bo mogoče podpreti vsa orodja, ki jih potrebujemo za analizo besedila. Poleg tega želimo arhitekturo narediti dovolj splošno, da bo kasneje mogoče dodati tudi nova orodja, ki si jih trenutno še nismo zamislili.

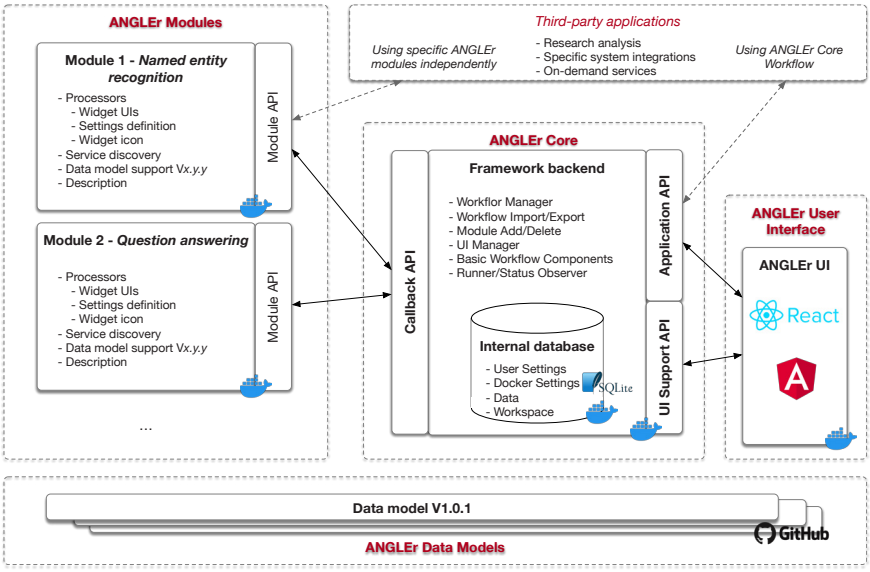
Orodje bo na osebnem računalniku (ali v oblaku) teklo kot storitev s spletnim vmesnikom.

Slika 13 prikazuje visokonivojsko predstavitev arhitekture ogrodja ANGLEr. Arhitektura sestoji iz naslednjih osnovnih gradnikov:

20 https://semver.org/

(a) moduli ANGLEr, (b) ANGLEr Core, (c) ANGLEr UI in (d) podatkovni modeli ANGLEr. S terminološkega vidika je pomembno razumeti predvsem koncepta ANGLEr modulov:

- ***Processor***: To je algoritem, ki sprejema in vrača podatke preko specifične verzije podatkovnega modela ANGLEr. V cevovodu/delotoku predstavlja en korak oz. komponento na grafičnem vmesniku.
- ***Module***: Modul predstavlja programski paket na osnovi Docker vsebnikov, ki vključuje enega ali več procesorjev (tj. implementacij specifičnih algoritmov ONJ). Vsak izmed njih pa mora omogočati integracijo z osnovnim gradnikom ANGLEr Core (tj. definicijo nastavitev, grafičnega vmesnika, ...) preko programskega vmesnika *Module API* (Razdelek 4.1). Modul je lahko vzpostavljen na ločeni/oddaljeni infrastrukuri in ga lahko neposredno prek programskega vmesnika poleg ogrodja ANGLEr uporabljajo tudi druge aplikacije (*Third-party applications*).



Slika 13: Predlog visoko-nivojske arhitekture ogrodja ANGLEr.

Podatkovni modeli ANGLEr (Razdelek 3.7) so na voljo v ločenem repozitoriju. Vsaka namestitev ogrodja ANGLEr uporablja specifičen podatkovni model, ki je obratno združljiv s starejšimi verzijami.

Osrednja arhitekturna komponenta ANGLEr Core zagotavlja glavne funkcionalnosti, ki orkestrirajo ostale komponente med seboj. Implementira upravljanje z delotoki, nameščanje/odstranjevanje modulov, komunikacijo z okoljem Docker, shranjevanjem podatkov v interno podatkovno bazo in programske vmesnike za komunikacijo. Programski vmesnik *Callback API* omogoča komunikacijo z vsemi »nameščenimi« oz. identificiranimi komponentami za ONJ. Vmesnik *Application API* omogoča programsko upravljanje s celotnim ogrodjem, kot to omogoča Orange preko knjižnice Python. Tako bo lahko nekdo npr. preko grafičnega vmesnika pripravil delotok, ga shranil, nato pa ga bo lahko integriral v nek drug sistem, ki bo delotok zaganjal preko programskega vmesnika. Vmesnik *UI Support API* pa implementira akcije, ki so vezane na nastavitve grafičnega vmesnika.

Komponenta *ANGLEr user interface* zagotavlja grafični vmesnik za delo z orodjem. Kot vse komponente je tudi ta šibko sklopljena z ostalimi deli ogrodja, kar omogoča, da bi lahko nekdo za delo z ogrodjem razvil svoj lasten vmesnik. Tako se lahko v prihodnosti za ogrodje razvije več programskih vmesnikov z različnimi funkcionalnostmi (npr. za različne interesne skupine uporabnikov), če bi bilo to potrebno. Komponenta nudi osnovne grafične elemente za *ANGLEr Core*, kot so dostop do nastavitev, menijev, delo z delotoki (tj. zaganjanje, ustavljanje, shranjevanje, odpiranje obstoječih, ...) in vključevanje grafičnih vmesnikov posameznih modulov. Specifični uporabniški vmesniki posameznih modulov (npr. ikone komponent/procesorjev, njihove nastavitve, vizualizacije) so implementirane neposredno v posameznih modulih in le prikazane v skupnem uporabniškem vmesniku.

Orodja bodo delovala kot samostojni programi, ki bodo izpostavili REST vmesnik, preko katerega bo glavni program komuniciral z njimi. Na ta način je implementacija vsakega orodja lahko povsem neodvisna od implementacije ostalih orodij in glavnega programa.

Orodja bodo praviloma pognana znotraj docker vsebnikov, s čimer bomo zagotovili delovanje orodij na vseh podprtih sistemih. Arhitektura omogoča tudi uporabo orodij, ki so pognana na oddaljenih računalnikih, saj vsa komunikacija poteka preko spletnih storitev. To omogoča, da orodja, ki za svoje delovanje zahtevajo grafično kartico (ali drugo zmogljivo opremo), tečejo na oddaljenem strežniku z ustrezno strojno opremo, uporabljamo pa jih na svojem lokalnem računalniku. Na takšen način se lahko razvije tudi tržnica storitev, kjer bi razvijalci/raziskovalci nudili svoje algoritme skupnosti.

4.1 Programski vmesnik Module API

Naloga vmesnika REST *Module API* je, da povezuje glavni program z vsakim izmed vsebnikov, ki poganjajo orodja. ANGLEr orodja uporabljajo enoten *ANGLEr Data Object*, ki je primerek podatkovnega modela ANGLEr, zato je preko vmesnika dovolj le definirati »naslovni prostor« podatkov za vhode in izhode algoritmov (tj. *inputs* in *outputs* na Sliki 14). Vmesnik mora podpirati naslednje:

- Pridobitev podatkov o modulu in orodjih, ki jih ponuja vsebnik.
- Spletno točko za delo z orodjem (tj. zagon, ustavitev).
- Spletno točko, na katerem je na voljo spletni vmesnik za konfiguracijo orodja.
- Pridobitev stanja orodja, kar pomeni seznam nalog, ki tečejo, in njihov napredek.
- Pridobivanje dnevniških zapisov.

Slika 14 prikazuje zahtevana polja pri komunikaciji z moduli ANGLEr ob pridobivanju podatkov o modulu in orodjih (tj. »*service discovery*«), za točki */about* in */processors.* Točka */docs* pa mora vračati HTML dokumentacijo o modulu. Ti podatki omogočijo delu ANGLEr Core, da na podlagi podanega osnovnega naslova modula samodejno prepozna funkcionalnosti in implementirane dele modula, ki jih lahko vključi v osnoven grafični vmesnik. Moduli so tako lahko pripravljeni v popolnoma drugih tehnologijah, kot jih uporablja osnovno ogrodje ANGLEr. Vizualizacije posameznega modula so prikazane

kot vsebniki na upravljalski strani ogrodja ANGLEr, kar pomeni, da je tudi tehnologija razvoja grafičnih elementov lahko popolnoma druga. Poleg tega lahko razvijalci uporabijo kako ogrodje za enostavno vizualizacijo svojih rezultatov, kot je na primer Gradio.[21]

```
// Endpoint: /about GET
{
  "UUID": "81a03d1a-0844-11ed-861d", //Identifier of the module.
  "name": "Best NER processors", //Name of the module.
  "version": "v1.2.9", //Module version.
  "data_model": "v1.0.1", //Version of the data model used.
  "desc": null, //(optional) Module description.
  "authors": "Jane Doe", //(optional) List of authors.
  "organisation": "University of Ljubljana", //(optional) Organisation of the authors.
  "url": "best-ner.github.io", //(optional) URL address of the page about the module.
  "docs_url": //(optional) A web page containing the documentation.
}

// Endpoint: /processors GET
[
  {
    "name": "CRF-based NER tagger" //Name of a processor.
    "short_name": "CRF-NER", //(optional) Short version of the name.
    "settings_endpoint": "/crf-ner/settings", //(optional) A module-relative address of a page containing processor settings.
    "ui_endpoint": "/ner-visualizations", //(optional) A module-relative address of a page for visualization.
    "icon": "/crf-ner/icon.ico", //A module-relative address of the icon to be used to represent the  processor.
    "category": "Semantic analysis", //A category of the processors menu that should  contain this processor (i.e., widget).
    "run_endpoint":  "/crf-ner/run", //Starting a processor. Data (i.e., current ANGLEr data object) is sent as payload.
    "status_endpoint":  "/crf-ner/status", //(optional) Returning statuses READY, PROCESSING (XX \%)
    "stop_endpoint":  "/crf-ner/stop", //Stopping/canceling current processing
    "inputs": ["TOKENIZATION", "WORDS"], //A list of ANGLEr data object types (from a hierarchy) that need to exist in the current data object for running this processor.
    "outputs": ["NER-ANNOTATION"], //A list of ANGLEr data object types (from a hierarchy, as high-level as possible) representing the processor's outputs.
    "docs_url": //(optional) A web page containing the documentation.
  },
  ... // Other available processors
]
```

Slika 14: Predlog definicije programskega vmesnika ANGLEr module API.

4.2 Arhitektura Docker

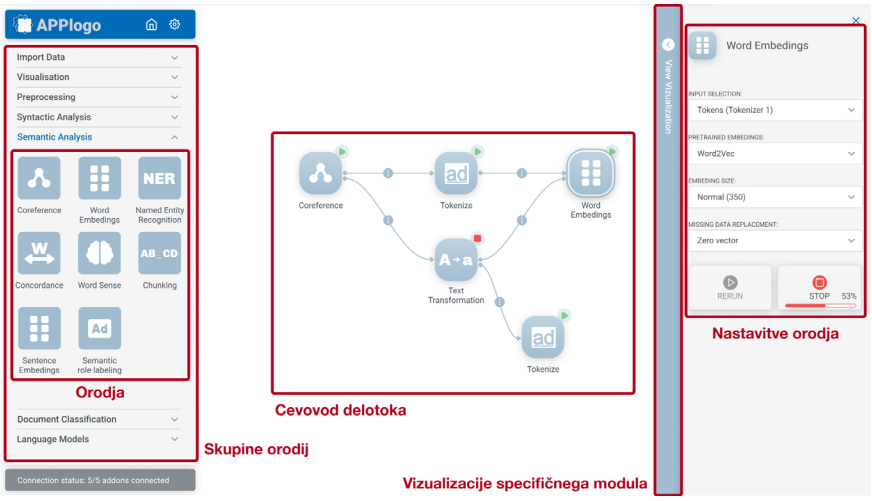
Uporaba arhitekture na osnovi tehnologije vsebnikov – Docker ni obvezna. Nekdo, ki bo razvijal svoje module, lahko na strežnikih storitve zaganja neposredno, brez vsebnikov, saj je potrebno za komunikacijo med gradniki zagotoviti le skladno implementacijo programskega vmesnika.

S stališča interoperabilnosti med sistemi, enostavne namestitve in zagona pa bodo vsi arhitekturni deli na voljo prek Docker slik. Docker slike lahko vsebujejo prednameščene vse odvisnosti, ni potrebe po posebni pripravi okolja za zagon in tudi razširjanje je enostavnejše, saj ostalim le delimo Docker sliko. V primerih uporabe modulov za več uporabnikov je možno Docker slike enostavno horizontalno in vertikalno razširjati. Osnoven sistem je načrtovan kot enouporabniški. Za ponujanje več uporabnikom pa lahko zanje tečejo različni primerki osnovnega ogrodja, ki pa vsi uporabljajo iste module.

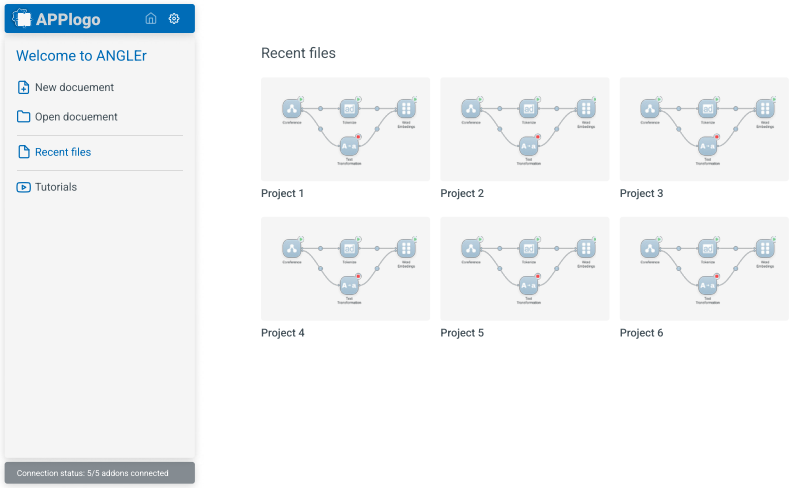
21 https://gradio.app

5 Predlog grafičnega vmesnika ANGLEr

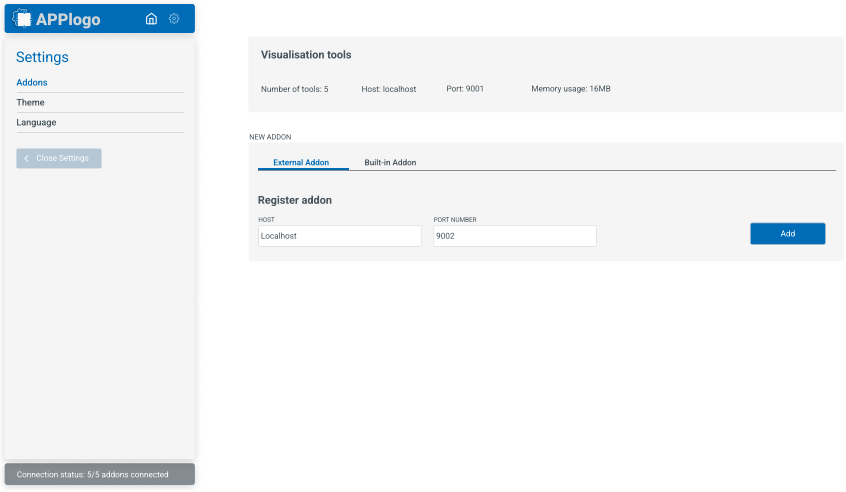
Grafični vmesnik mora omogočati poljubno povezovanje orodij s pomočjo kreiranja delotokov v obliki cevovodov procesiranja. Pomembno je definirati tudi, da se morajo komponente zaganjati zaporedno (razen ob vejitvah, kjer lahko analize tečejo vzporedno). Slika 15 prikazuje predlog splošnega grafičnega vmesnika orodja ANGLEr. Na levem delu je na voljo seznam vključenih orodij v sistem, ki jih lahko uporabnik povleče na delovno površino (tj. delotok) in povezuje med seboj v cevovod procesiranja. Na desni strani lahko upravlja z nastavitvami/parametri posameznih modulov in njihovimi vizualizacijami. Za boljšo ponazoritev možnosti je več primerov vmesnikov prikazanih na Slikah 16, 17, 18, 19 in 20.



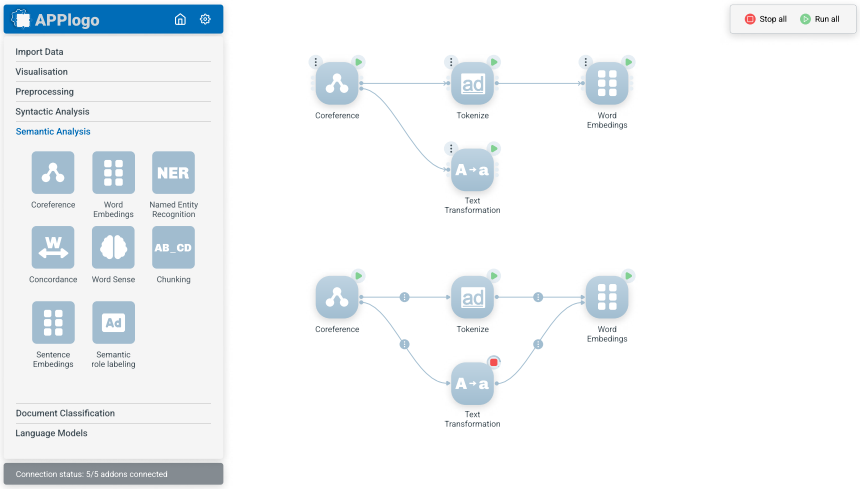
Slika 15: Predlog splošnega grafičnega vmesnika ogrodja ANGLEr.



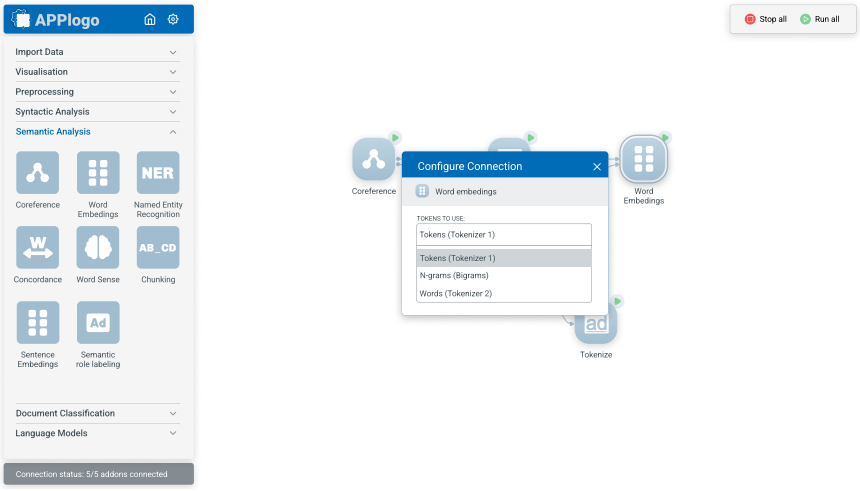
Slika 16: Predlog splošnega grafičnega vmesnika ogrodja ANGLEr. Začetni zaslon.



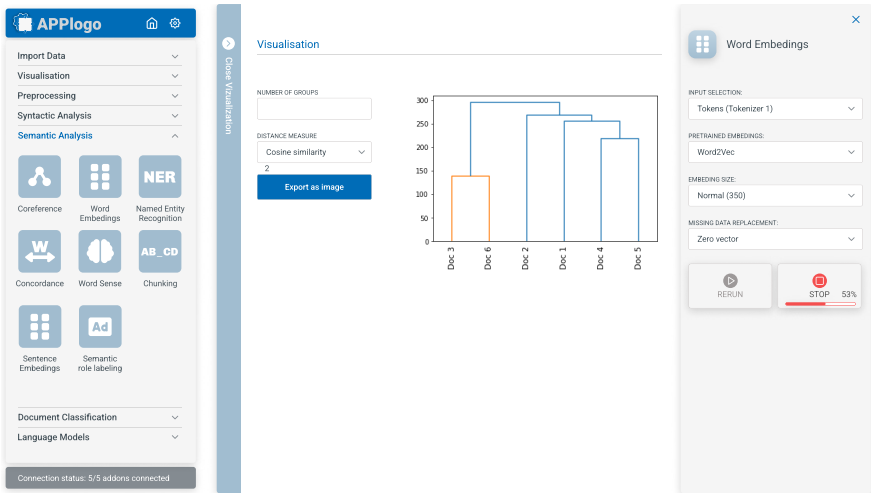
Slika 17: Predlog splošnega grafičnega vmesnika ogrodja ANGLEr. Glavne nastavitve za pregled in dodajanje modulov v sistem. Za dodajanje modula je potreben le URL naslov do njegovega programskega vmesnika REST.



Slika 18: Predlog splošnega grafičnega vmesnika ogrodja ANGLEr. Pregled delotoka, ki omogoča zaganjanje celotnega postopka, pregled posameznih procesorjev ali njihovo ločeno zaganjanje.



Slika 19: Predlog splošnega grafičnega vmesnika ogrodja ANGLEr. Povezovanje procesorjev med seboj in izbira vhodov glede na podatke v podatkovnem modelu.



Slika 20: Predlog splošnega grafičnega vmesnika ogrodja ANGLEr. Pregled vizualizacije izbranega procesorja.

6 Sklep

V tem prispevku smo pregledali obstoječa splošna ogrodja za obdelavo naravnega jezika in na podlagi tega pripravili načrt izgradnje novega splošnega in razširljivega podatkovnega modela ter ogrodja za analizo besedil – ANGLEr. Raziskovalni skupnosti in splošno zainteresirani javnosti bi takšno orodje za demokratizacijo obdelave naravnega jezika precej koristilo. Orodje bi omogočalo enostaven zagon najnovejših algoritmov s strani tehnično neveščih uporabnikov. Glede na pregledane uspešne primere bi bilo potrebno zagotoviti stalne posodobitve ogrodja in zagotavljanje primerov, novih modulov skupnosti, ki bi pri tem tudi sodelovala.

Menimo, da bi predlog izboljšal ponudbo na področju uporabe metod za obdelavo naravnega jezika. Ogrodje bi omogočilo tudi hitro prototipiranje in hiter dostop do prikaza uporabnosti metod za raziskovalce. V primerjavi z obstoječimi pristopi sta ključni izboljšavi, ki jih uvaja ANGLEr, (a) definicija enotnega, razširljivega, enostavnega in verzioniranega podatkovnega modela ter (b) vključevanje modulov le na podlagi jasno definiranega programskega

vmesnika API. Slednje omogoča, da so lahko nova orodja pripravljena v poljubni tehnologiji in da razvijalcem ni potrebno uporabljati predizbranih programskih jezikov, ogrodij ali poznati širše arhitekture ogrodja, kot je to potrebno pri najuspešnejših ogrodjih Orange in KNIME.

Ključna vprašanja, ki še dodatno poudarijo pomembnost izgradnje predlaganega ogrodja so naslednja:

- **Ali je potrebno predstaviti naloge obdelave naravnega jezika v celostnem podatkovnem modelu?** Obstaja več standardov za opisovanje podatkov in kar nekaj načinov hranjenja podatkov. Zaradi tega je pri razvijanju metod vedno potrebno prilagajati vhode in izhode, pri čemer ne obstajajo standardni postopki za transformacijo med predstavitvami. Veliko iniciativ za predstavitev podatkov je zamrlo, zato smo sledili uspešnim primerom in jih izboljšali, da je podatkovni model celosten in enostaven, kolikor je le možno.
- **Ali je potrebno (spet) razviti še eno ogrodje za obdelavo naravnega jezika?** Tehnologija se ves čas spreminja. Zaradi tega je šibko sklopljena arhitektura pravi odgovor na dolgoročno vzdrževanje sistema. Kljub temu pa je zelo pomembna tudi promocija ogrodja med zainteresiranimi deležniki. Trenutno se zdi spletni vmesnik in rešitev z vsebniki prava pot, vendar se lahko v prihodnosti spremeni tudi to. Srednja pot bi bila integracija orodij v obstoječe ogrodje (npr. Orange ali KNIME), vendar s tem postanemo zelo odvisni od ciljnega sistema, poleg tega pa je potrebno komponente ves čas prilagajati glede na razvoj osnovnega sistema.
- **Kdo so ciljni uporabniki in ali bi plačali za uporabo predlaganega sistema?** Trenutno veliko število raziskovalcev in jezikoslovcev uporablja orodja, ki jim za njihove namene omogočajo obdelavo besedil. Verjetno bi bil najboljši način slediti praksam GATE ali SketchEngine, pri čemer bi se začetni razvoj zgodil v okviru večjega raziskovalnega projekta. Celotno ogrodje bi seveda moralo ostati odprtokodno, tako da bi uporabniki lahko plačevali zgolj za storitev. Poleg tega pa bi lahko omogočili tržnico

orodij, saj bi podobno lahko raziskovalci na svojih strežnikih gostili svoja orodja in jih ponujali uporabnikom.

- **Kdo bi skrbel za dolgoročni razvoj takšnega ogrodja?** Kljub temu, da bi ogrodje uporabljala skupnost in razvijala odprtokodne module, bi moral nekdo skrbeti, posodabljati in usmerjati razvoj ogrodja. Glede na pregledana ogrodja je očitno, da »preživijo« le tista, za katera nekdo aktivno skrbi, organizira delavnice, izobraževanja (npr. Orange, KNIME, GATE). Možna »skrbnika« sta dveh tipov – raziskovalna skupina ali podjetje. Razvoj takšnega ogrodja lahko prevzame raziskovalna skupina, ki razvija tehnologije za obdelavo naravnega jezika, ima dovolj projektov, da financira tudi razvoj takšnega ogrodja, ki ga uporablja tudi za lastno promocijo. Podobno bi bilo v primeru podjetij, ki bi tržilo razvoj tehnologij za obdelavo naravnega jezika in bi ogrodje uporabljalo kot eno izmed svojih glavnih infrastrukturnih komponent. Glede na to, da bi bilo ogrodje odprtokodno, bi podjetje lahko vzporedno razvijalo tudi plačljive nadgradnje/module, ki bi bili tržno zanimivi.

Zahvale

Projekt Razvoj slovenščine v digitalnem okolju sta med leti 2020 in 2023 sofinancirali Republika Slovenija in Evropska unija iz Evropskega sklada za regionalni razvoj (Operacija se je izvajala v okviru Operativnega programa za izvajanje evropske kohezijske politike v obdobju 2014–2020).

Literatura

Apache (2010). *Apache OpenNLP.* http://opennlp.apache.org

Bird, S. & Loper, E. (2006). *NLTK: the natural language toolkit*. In Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions (pp. 69-72).

Bodenreider, O. (2004). *The unified medical language system (UMLS): integrating biomedical terminology*. Nucleic acids research, 32 (suppl_1), D267-D270.

Cunningham, H. (2002). *GATE, a general architecture for text engineering.* Computers and the Humanities, 36, 223-254.

Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevar, T., Milutinovič, M., ... & Zupan, B. (2013). *Orange: data mining toolbox in Python*. The Journal of machine Learning research, 14(1), 2349-2353.

Ferrucci, D., & Lally, A. (2004). *UIMA: an architectural approach to unstructured information processing in the corporate research environment*. Natural Language Engineering, 10(3-4), 327-348.

Hellmann, S., Lehmann, J., & Auer, S. (2012). *Linked-data aware uri schemes for referencing text fragments*. In Knowledge Engineering and Knowledge Management: 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings 18 (pp. 175-184). Springer Berlin Heidelberg.

Hellmann, S., Lehmann, J., Auer, S., & Brümmer, M. (2013). *Integrating NLP using linked data*. In The Semantic Web–ISWC 2013: 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II 12 (pp. 98-113).

Hrovat, N. (2022). Zasnova ogrodja za izvajanje metod za procesiranje naravnega jezika. [Diplomsko delo, Univerza v Ljubljani] Repozitorij Univerze v Ljubljani. https://repozitorij.uni-lj.si/IzpisGradiva.php?lang=slv&id=141460

Ide, N., & Romary, L. (2004). *International standard for a linguistic annotation framework.* Natural language engineering, 10(3-4), 211-225.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., & McClosky, D. (2014). *The Stanford CoreNLP natural language processing toolkit*. In Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations (pp. 55-60).

Meystre, S. M., Lee, S., Jung, C. Y., & Chevrier, R. D. (2012). *Common data model for natural language processing based on two existing standard information models: CDA+GrAF.* Journal of biomedical informatics, 45(4), 703-710.

Nance, J., & Baumgartner, P. (2021). *gobbli: A uniform interface to deep learning for text in Python*. Journal of Open Source Software, 6(62), 2395.

Qi, P., Zhang, Y., Zhang, Y., Bolton, J., Manning, C.D. (2020). *Stanza: A Python Natural Language Processing Toolkit for Many Human Languages*. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations (pp. 101-108).

Sherif, M. A., Coelho, S., Usbeck, R., Hellmann, S., Lehmann, J., Brümmer, M., & Both, A. (2014). *NIF4OGGD-NLP Interchange Format for Open German Governmental Data*. In LREC (pp. 3524-3528).

Soysal, E., Wang, J., Jiang, M., Wu, Y., Pakhomov, S., Liu, H., & Xu, H. (2018). *CLAMP–a toolkit for efficiently building customized clinical natural language processing pipelines*. Journal of the American Medical Informatics Association, 25(3), 331-336.