Laboratorij za tehnologije znanja (KTLab)
Zavod za elektroniku, mikrolektroniku, računalne i inteligentne sustave
Fakultet elektrotehnike i računarstva
Sveučilište u Zagrebu

Interni dokument

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1865

PROVJERNIK STILA U PISANOM TEKSTU

Davor Djaković

Zagreb, rujan 2010.

Sadržaj

Popis tablica

Popis slika

1. Uvod

Područje rada ovog diplomskog zadatka je računalna obrada prirodnog jezika (engl. *natural language processing*). To je podskup lingvistike i računarstva koji se bavi analizom prirodnog jezika uz upotrebu računala. Jedan od mnogih podskupova obrade prirodnog jezika je provjera ispravnosti pisanog teksta. Takva provjera može biti tek provjera pravopisa i gramatike, ali može biti i proširena na provjeru stila i semantike. Provjere se najčešće vrše na temelju skupova pravila, velikih tekstnih korpusa te heurističkih, stohastičkih i probabilističkih metoda analize. U ovom radu koristit ćemo tekstne korpuse i jednostavan heuristički algoritam.

Engleski jezik jedan je od najraširenijih jezika na svijetu i postao je neslužbeni svjetski standard. Njime govori, ili barem želi govoriti, većina svjetskog stanovništva, jer se ta potreba ukazala procesima globalizacije. Međutim, kako mnogima od njih taj jezik nije materinji, često im se događaju jezične pogreške. To je razlog postojanja potrebe za programima za provjeru pravopisa, gramatike i stila.

Cilj ovog rada izgradnja je programa za provjeru stila teksta pisanog na engleskom jeziku, u svrhu olakšavanja njegove upotrebe i učenja govornicima kojima nije materinji.

Kako se u radu bavimo stilom, počnimo od pitanja što je stil uopće. Oksfordski rječnik u svojoj prvog od šest definicija za imenicu *style* [1] definira stil kao *poseban postupak kojim se nešto čini/radi*, a specifično o pisanju (dakle jeziku) kaže:

- *način slikanja, pisanja, skladanja, građenja itd., osobina pojedinog razdoblja, mjesta, osobe ili pokreta;*
- *način korištenja jezika.*

Iz ovoga se može zaključiti da je stil poprilično subjektivan pojam i kao takav težak za ocjenjivanje. Programska rješenja za provjeru stila pisanog teksta su često utemeljena na skupovima pravila, koja mogu biti napisana od strane čovjeka ili izračunata od strane računala, ako se računalu pruže tekstovi s primjerima pogrešaka i njihovim ispravcima.

U ovom se radu orijentiramo na drugačiji pristup, koji će biti utemeljen na korištenju vrlo velikih skupova podataka (rečenice iz znanstvenih članaka ili skupine *n*-

grama iz književnih djela) koje će biti potrebno učinkovito pretražiti i pokušati naći zamjene rečenicama teksta kojeg provjeravamo. Takva metoda je heuristička i neće uvijek dati željene rezultate, ali to može biti popravljeno kombiniranjem heuristike sa skupovima pravila. Kako će rad ovog programa uvelike ovisiti o zadanom skupu podataka iz kojeg se računaju vjerojatnosti točnosti pojedinih rečenica, tako će i naglasak na tip stila koji program "smatra" ispravnim biti određen upravo tim skupom podataka.

Radit će se s dva skupa podataka, a to su baza stručnih tekstova i članaka akademske zajednice *Google Scholar* (http://scholar.google.com) te baza književnih djela nastala u *Projektu Gutenberg* (http://www.gutenberg.org). Očigledno, korištenje prvog skupa podataka uzrokovat će preferiranje znanstvenog stila, dok će korištenje drugog skupa uzrokovati preferiranje literarnog te vjerojatno pomalo zastarjelog stila.

Prije opisa samog programa, u poglavlju 2 osvrnut ćemo se na teorijsku pozadinu ispravljanja stila, tj. konstrukciju rečenice, pravopis, gramatiku i semantiku, zatim na neke osnovne pojmove u računalnoj obradi prirodnog jezika te slične radove. U poglavlju 3 bit će opisan algoritam provjere stila nastao za potrebe izrade ovog diplomskog rada, bez ulaženja u detalja implementacije samog algoritma. U poglavlju 4 bit će opisan nastanak programskog rješenja kroz sve korake njegove izrade, problemi s kojima se autor susreo, tehnologije koje se koriste, način na koji se pristupa podacima koje program koristi te organizacija tih podataka, implementacija algoritma opisanog u poglavlju 3, pregled napisanih razreda i njihove uloge te upute za instalaciju i korištenje. Konačno, u poglavlju 5 ćemo na temelju primjera vidjeti i što program može, a što ne može te zašto nešto može ili ne može. Kako je provjera stila zahtjevan problem, program je zapravo jedan veliki posao u nastanku, što znači da je otvoreno još puno mogućnosti za poboljšanja koje će biti opisane u Zaključku (poglavlje 6).

2. Teorijska pozadina

2. 1. Jezik i konstrukcija rečenice

Provjera stila pisanog teksta u svom specifičnom značenju obuhvaća samo provjeru načina na koji je tekst pisan, kako je i definirano u definiciji iz rječnika citiranoj u uvodu, ali naravno, prilikom same provjere je nemoguće ograničiti se samo na provjeru stila a da se prethodno ne obave sve ostale provjere "niže razine".

Npr., ako bismo provjeravali stil rečenice

This book are very interesting.

uvidjeli bismo da ova rečenica doista sadrži pogrešku, ali da ona nije stilskog tipa već gramatičkog; na imenicu u jednini (*book*) se nastavlja glagol *to be* u trećem licu množine (*are*), a trebao bi biti u trećem licu jednine (*is*).

Primjer iznad je, dakle, primjer gramatičke pogreške, ali to je samo jedna od razina s kojih možemo promatrati konstrukciju rečenice:

- **Pravopis** se bavi provjerom ispravnosti načina na koji su riječi pisane. To ponajprije znači da riječ koja je napisana mora postojati i imati značenje (npr. *languaige* je pogrešno, a *language* pravilno). Osim toga, riječ mora imati pravilan raspored velikih i malih slova (npr. vlastito ime *John* mora biti napisano s velikim početnim slovom).
- **Gramatika** propisuje pravilan način slaganja riječi u veće jezične cjeline (priloške oznake, imeničke i glagolske skupove...) te pravilnog slaganja tih skupova u jednostavne rečenice koje opet mogu, a i ne moraju biti dio većih složenih rečenica. Primjer gramatičke pogreške dan je iznad na ovoj stranici.
- **Stil**, za razliku od pravopisa i gramatike, nije jedinstveno propisan za čitavi jezik, već ovisi o zahtjevima pojedinog područja kojim se neki tekst bavi. Npr. dok je u umjetnosti u nekim slučajevima kompliciran stil s dugim rečenicama i neobičnim riječima i poželjan, u stručnim ili službenim tekstovima takav se način pisanja izbjegava. Jedan od zahtjeva stila u stručnim tekstovima može biti i nekorištenje skraćenih oblika glagola (*don't* umjesto *do not*). Stilski problemi

također mogu nastati i kod govornika koji pokušavaju govoriti jezikom koji im nije materinji, pa se vode nekim pravilima vlastitog jezika koja u jeziku kojim žele govoriti nisu važeća.

- **Semantika** se bavi promatranjem teksta na svim razinama te proučava pitanja smisla i značenja. Greške koje nisu ni pravopisne ni gramatičke ni stilske smatramo semantičkima. To znači da iako rečenica može biti gramatički i stilski ispravna, ona svejedno može biti besmislena, kao u ovom primjeru: *I am driving a fish.* Očigledno, sve riječi ove rečenice su pravilno napisane, gramatička konstrukcija je također ispravna, ali rečenica jednostavno nema smisla. Ovakve greške je teže otkriti računalnim sustavom jer to zahtijeva veliko znanje o činjenicama iz stvarnog svijeta te mogućnost razumijevanja teksta od strane računala.

2. 2. Pojavnica

Za ovaj rad je jako važan pojam pojavnice (engl. *token*) [2]. Pojavnica se odnosi na građevne elemente jezika niske razine (poput riječi ili pravopisnih znakova) od kojih se izgrađuju veće jezične cjeline. Važno je naglasiti da pojavnice nisu uvijek razdvojene razmakom, i to ne samo u slučaju pravopisnih znakova kada je to čak i rijetkost (npr. zarez se uvijek piše uz riječ koja mu prethodi), već i u slučaju nekih parova riječi. Najčešći primjer ovog drugog slučaja u engleskom jeziku su skraćeni oblici glagola poput skraćenog oblika glagola *to be*, koji npr. u izrazu *I am* može prijeći u *I'm*. Vidimo da zamjenica *I* i glagol *to be* u ovom slučaju nisu odvojeni razmakom, iako se nedvojbeno radi o dvije zasebne riječi.

2. 3. Označavanje vrste riječi

Proces označavanja vrste riječi (engl. *part-of-speech tagging*) je nakon rastavljanja teksta na rečenice i rečenica na riječi prvi korak u analizi rečenice. U tom procesu svakoj se pojedinoj riječi pridjeljuje oznaka koja označava njezinu vrstu. Primjeri takvih oznaka su *imenica*, *glagol*, *pridjev*... Neki računalni označivači imaju i mogućnost označavanja pojavnica koje nisu riječi, poput pravopisnih znakova. U tom slučaju, uz gore nabrojani tip oznaka, postoje i oznake poput *zarez*, *kraj rečenice*...

2. 4. *N*-gram

N-gram je definiran kao podniz *n* elemenata nekog niza. Specifično u obradi prirodnog jezika, *n*-gram često označava niz pojavnica koji je podniz većeg niza – rečenice. *N*-grami su važni u heurističkoj analizi teksta, gdje se koriste velike baze *n*-grama popisanih prema njihovoj učestalosti pojavljivanja u jeziku. Pomoću podataka iz tih baza sustavi za obradu prirodnog jezika mogu predlagati ispravke za krivo napisane riječi, prepoznavati jezik teksta, ocjenjivati vjerojatnost pojave nekog niza u jeziku itd. [3]

2. 5. Slični projekti

2. 5. 1. A Rule-Based Grammar and Style-Checker

Ovaj projekt nastao je kao diplomski rad Daniela Nabera [4] i kao što mu ime i govori, utemeljen je na pravilima. Program je napisan u Pythonu. Tekst se najprije rastavlja na rečenice pomoću gotovog rješenja, a zatim se, također pomoću gotovog rješenja, vrši označavanje riječi, što je vrlo slično pristupu u ovom radu. Pravila su podijeljena u dvije skupine, a to su pravila za provjeru gramatike i pravila za provjeru stila, što je potpuno različit pristup od pristupa ovog rada u kojem koristimo samo tekstne korpuse. Pravila su napisana u vanjskom XML-dokumentu koji se koristi tijekom rada programa. Sva pravila su pisana ručno od strane autora. Osim provjera izraženih pomoću pravila, u ovaj sustav ugrađeno je i nekoliko dodatnih provjera napisanih izravno u programskom jeziku poput ograničenja rečenice na maksimalno 30 riječi, provjere pravilnog korištenja neodređenog člana (*a*/*an*) i provjere pojavljivanja dviju istih riječi jedne do druge.

Pravila za provjeru stila ograničena su tek na nekoliko pravila poput onoga koje brani započinjanje rečenice veznikom *or*, jer, kako kaže autor, stil je ponajprije pitanje osobnih afiniteta i vrste teksta koju pišemo.

2. 5. 2. After the Deadline

Projekt *After the Deadline* rješenje je za provjeru pravopisa, stila i gramatike danog teksta napravljeno kao kombinacija jednog *web*-servisa i raznih klijenata [5, 6]. Servisu je moguće pristupati slanjem XML-upita, a rezultati se vraćaju na isti način. Ovakva organizacija omogućava pisanje raznih klijenata i široku dostupnost usluge, a prednost je i u tome što se korišteni jezični model tako može držati i održavati na poslužitelju, jer bi bio

prevelik za klijentske aplikacije (zahtijeva barem 1 GB radne memorije). Također, svako poboljšanje programa odražava se odmah na sve klijente koji servis koriste, tj. potrebno je vršiti izmjene jedino na poslužitelju, ne i na samim klijentskim programima.

Provjeru pravopisa izvršava u odnosu na kontekst, što znači da, za razliku od programa napravljenog u ovom diplomskom radu, ovo rješenje ima ugrađene i neke mehanizme razumijevanja semantike pisanog teksta. Zapravo, kako je u poglavlju 4.6.3 i opisano, program ovog diplomskog rada zasad uopće ne koristi provjeru pravopisa, iako je dotična metoda napisana u jednom od razreda.

Servis *After the Deadline* moguće je koristiti preko raznih sučelja i na razne načine: u internetskom pregledniku (postoje dodaci za *Firefox* i *Google Chrome*), na *blogu* (postoji dodatak za popularni CMS-servis *WordPress*), na vlastitom *web*-sjedištu (postoji dodatak za *TinyMCE*, *web*-kontrolu za uređivanje teksta) ili u programu za obradu teksta (postoji dodatak za *OpenOffice.org Writer*). Demonstracija rada dostupna je i izravno na URL-u http://www.polishmywriting.com/ (nije potrebna instalacija nikakvog dodatnog softvera). Za razliku od ove široke ponude, u projektu nastalom u ovom diplomskom radu koristimo tek vlastito *web*-sučelje, slično upravo navedenoj demonstraciji.

Jezični model nastao je koristeći tekstne korpuse nastale iz podataka *Wikipedie* i *Projekta Gutenberg*. Potonji izvor podataka korišten je i u projektu ovog diplomskog rada (uz *Google Scholar*). Za razliku od projekta opisanog u ovom radu koji koristi trigrame i 4-grame, jezični model projekta *After the Deadline* temelji se na bigramima. Osim već spomenutih izvora tekstova, u bazu je dodano i ponešto tekstova s raznih *blogova,* koji su selektirani tako što je nad njima izvršena provjera pravopisa te su odbačeni oni koji su imali prevelik broj pogrešaka. Čitavi korpus projekta ima 75 milijuna riječi. Usporedbe radi, korpus nastao na temelju *Projekta Gutenberg* u ovom diplomskom radu ima 77 milijuna trigrama. I *After the Deadline* koristi određeni broj trigrama, ali taj broj je, radi uštede memorijskog prostora, značajno manji od broja brigrama.

Opisani korpus nije jedina baza riječi projekta *After the Deadline*. Naime, koristeći neke javno dostupne liste riječi generiran je i jedan dodatni rječnik. Pri izvođenju provjere pravopisa koristi se rječnik koji je presjek generiranoga i skupa riječi sadržanih u iznad opisanom korpusu (bigrami i trigrami). To se radi kako bi se smanjila mogućnost pogreške.

Ispravljanje teksta vrši se kombinacijom provjere pravopisa (temeljeno na vjerojatnosnim metodama – "udaljenost" riječi, vjerojatnost pojave jedne riječi nakon druge i sl.) te provjere gramatike i stila pomoću skupa pravila. Razlika u odnosu na projekt ovog diplomskog rada je ta da u ovom projektu koristimo tek heurističku funkciju nad tekstim korpusima, bez provjere pravopisa te ne koristeći nikakav skup gramatičkih ili stilskih pravila.

3. Algoritam provjere stila

U ovom poglavlju dan je opis implementiranog algoritma za provjeru stila bez detalja njegove realizacije u programskom kodu. U svakom potpoglavlju bit će redom opisani koraci izvođenja algoritma. Implementirani algoritam pripada skupini heurističkih algoritama, a kao temelj koristi tekstne korpuse (alternativa tekstnim korpusima su skupovi pravila). Korištene formule nisu iterativno izračunate od strane računala, već se do njih došlo isprobavanjem na primjerima i podešavanjem u onom smjeru koji se autoru činio najprikladnijim. To znači da formule, iako dobre, vrlo vjerojatno nisu najbolje moguće. Bolji algoritam mogao bi biti izgrađen ako bismo u formule uveli faktore koje bismo mogli optimirati na skupu primjera za treniranje.

Algoritam je zamišljen na sljedeći način. Tekst se najprije rastavlja na rečenice. Zatim se svaka rečenica obrađuje posebno na način da se najprije generiraju *n*-grami od kojih se rečenica sastoji. Iz tih *n*-grama se izgrađuju nove rečenice alternativne originalnoj. Svaka od njih ocjenjuje se heurističkom ocjenom.

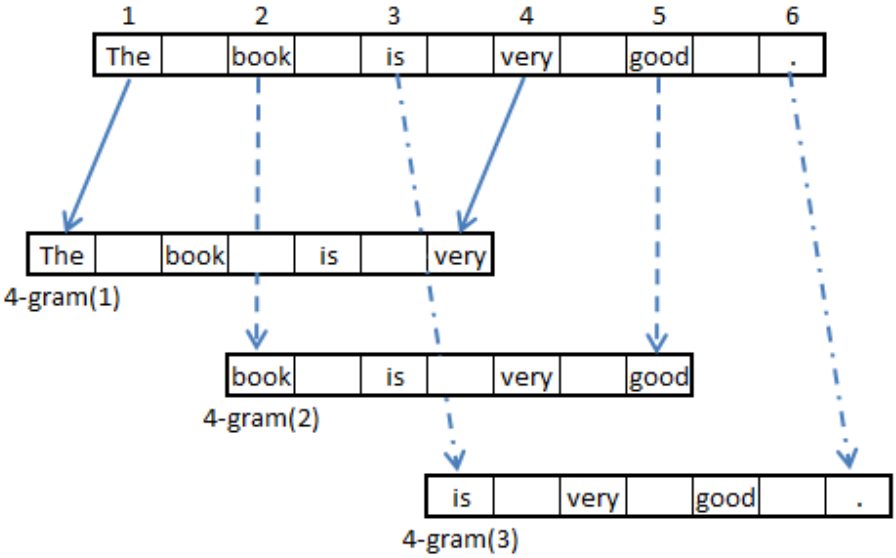
3. 1. Rastavljenje rečenica na *n*-grame

Duljina *n*-grama zadana je kao konstanta. Generiraju se tako da se uzme prvih *n* pojavnica rečenice i ta skupina postaje prvi *n*-gram, a svaki sljedeći *n*-gram sastoji se od podniza pojavnica za jedan pomaknutog udesno. Izraženo formulom, to bi bilo:

$$ngram(i) = znakovi(i, n),$$

gdje je *n* veličina *n*-grama, *i* pozicija početne riječi *n*-grama u rečenici te redni broj *n*-grama te rečenice, a funkcija *znakovi* vraća *n* riječi počevši od pozicije *i* u rečenici.

Svaki *n*-gram bit će posebno analiziran, a zatim će se ti pojedinačni rezultati iskoristiti za generiranje novih, moguće boljih, rečenica alternativnih originalnoj.



Slika 1: Primjer rastavljanja rečenice na *n*-grame

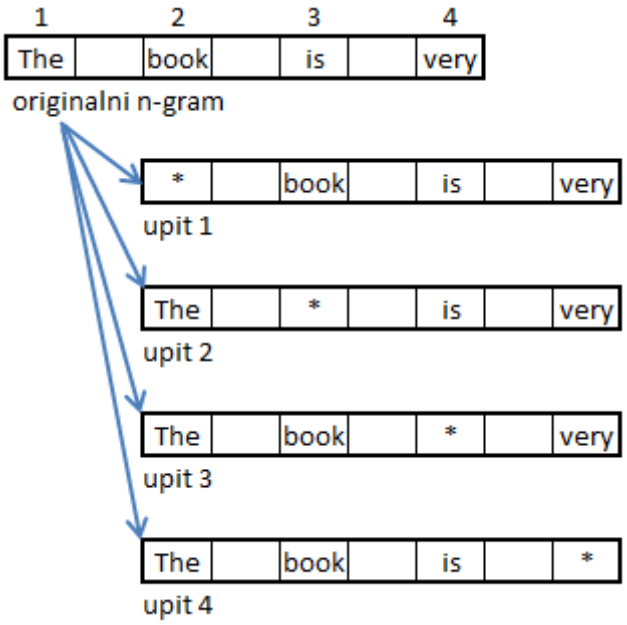
3. 2. Analiza *n*-grama

Prvi korak u analizi *n*-grama je označavanje riječi (engl. *POS-tagging*).

Drugi korak je generiranje upita koji će biti poslani izvoru podataka. Upiti su skupine pojavnica istovjetne skupu kakvog sadrži originalni *n*-gram, ali s najviše jednom pojavnicom označenom kao nepoznatom. U daljnjem tekstu to će biti označeno zvjezdicom, što je istovjetno programskoj implementaciji.

Zvjezdica označava da se za riječ na njezinom mjestu traže zamjene. Takav se upit predaje izvoru podataka koji za svaki upit pronalazi skupine *n*-grama u bazi dostupnih tekstova koji odgovaraju upitu, tj. one *n*-grame koji imaju sve riječi na odgovarajućim pozicijama jednake originalnom *n*-gramu, a umjesto zvjezdice mogu imati bilo koju pojavnicu.

Prije same predaje pojedinih upita neki od njih izbacuju se iz konačne liste. To se radi na temelju oznaka o vrstama riječi (engl. *part-of-speech tags*). Postoji 45 takvih oznaka (koriste se oznake biblioteke *OpenNLP* koja je opisana u poglavlju 4.2).



Slika 2: Primjer generiranja upita za 4-gram

Ideja je bila podijeliti pojavnice na one koje su vjerojatniji kandidati za zamjenu i na one koje su manje vjerojatni kandidati za zamjenu. Npr. govornicima kojima engleski jezik nije materinji puno se češće događaju pogreške u upotrebi prijedloga, određenih/neodređenih članova ili glagolskih vremena nego imenica, brojeva ili usklika.

Budući da biblioteka *OpenNLP* ne dijeli oznake ni na kakve skupine, autor je prema vlastitoj slobodnoj procjeni podijelio svih 45 oznaka na četiri skupine poredane po vjerojatnosti njihove prikladnosti za zamjenu. Te skupine su:

- pravopisni znakovi i znakovi za novac (najmanja vjerojatnost prikladnosti za zamjenu),
- pojavnice male vjerojatnosti prikladnosti za zamjenu,
- pojavnice umjerene vjerojatnosti prikladnosti za zamjenu,
- pojavnice velike vjerojatnosti prikladnosti za zamjenu.

Popis kategorija pojavnica dan je u Tablici 1. Neke oznake su ujedinjene pod zajedničkim imenom radi preglednosti (npr. imenice se ne dijele na imenice množine i jednine), a potpuni popis oznaka nalazi se u Dodatku A.

Tablica 1: Kategorije pojavnica prema vjerojatnosti prikladnosti za zamjenu

Pravopisni znakovi i znakovi za novac	Pojavnice male vjerojatnosti
" navodnici , zarez . ? ! oznaka kraja rečenice $ znak za dolar £ znak za funtu (otvorena zagrada) zatvorena zagrada ● oznaka člana liste	glavni broj strana riječ usklik imenica simbol čestica egzistencijalni *there* (*there is...*, *is there...*) posvojni sufiks (*'s*)
Pojavnice umjerene vjerojatnosti	Pojavnice velike vjerojatnosti
nemodalni glagol determinator predeterminator zamjenica	pridjev prilog veznik prijedlog modalni glagol *wh-riječ*[1]

Izbor upita će, dakle, izbaciti neke od generiranih upita; u gornjem primjeru *n*-grama (*the book is very*) to je upit *the * is very*, jer je pojavnica koja je zamjenjena zvjezdicom (*book*) imenica, a imenica se prema gornjoj tablici nalazi u kategoriji pojavnica male vjerojatnosti prikladnosti za zamjenu.

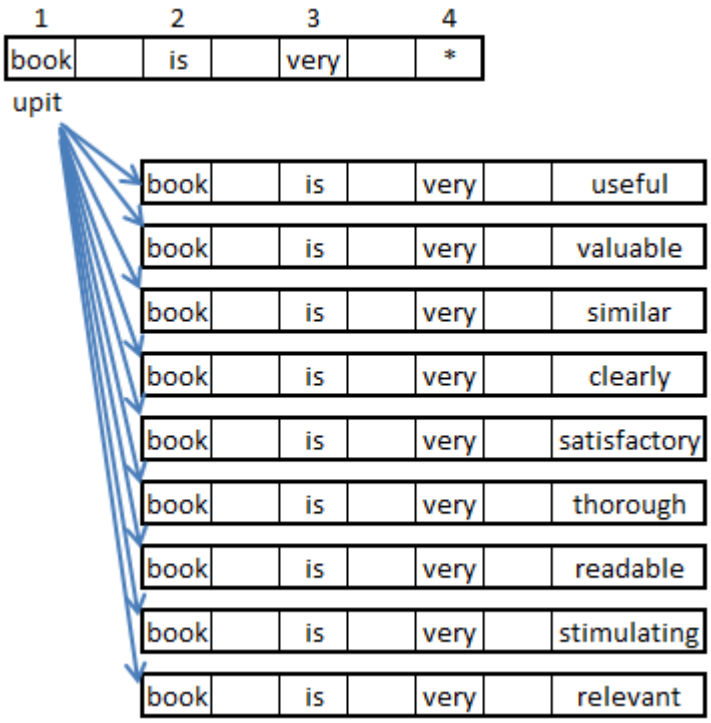
U tom bismo slučaju, dakle, iz zadanog 4-grama na kraju dobili sveukupno tri upita.

3. 3. Računanje alternativnih *n*-grama

Svi dobiveni upiti šalju se izvoru podataka koji pretražuje svoju bazu *n*-grama i za svaki od njih vraća one *n*-grame koji ga zadovoljavaju. Na Slici 3 vidimo primjere vraćenih *n*-grama za upit *book is very **.

Osim samih *n*-grama, izvor podataka nam vraća i učestalost pojavljivanja svakoga od njih, što će kasnije biti iskorišteno u ocjenjivanju rečenica koje će iz tih *n*-grama biti složene.

[1] Takozvane *wh-riječi* (engl. *wh-words*) u engleskom jeziku odnose se na skupinu upitnih i odnosnih riječi (ovisno o ulozi u pojedinoj uporabi) koje najčešće, ali ne nužno, započinju s *wh* (*what, why, where, which, who, how*).



Slika 3: Primjer skupine rezultirajućih *n*-grama za upit

3. 4. Generiranje alternativnih rečenica

Rečenice alternativne originalnoj generiraju se iz alternativnih *n*-grama dobivenih od izvora podataka. Algoritam kreće od prvog *n*-grama rečenice i za svaki njegov alternativni *n*-gram pomiče se na popis alternativnih *n*-grama prvog desnog susjeda. Tamo pronalazi one alternativne *n*-grame koji se mogu spojiti s alternativnim *n*-gramom prvog *n*-grama i tako izgrađuje određeni broj *(n+1)*-grama. Zatim se opet za svaki takav *(n+1)*-gram pomiče udesno i ponavlja postupak čime dobiva *(n+2)*-grame. To radi sve dok ne dođe do zadnjeg *n*-grama, kada izgrađuje određeni broj *(n+k)*-grama gdje je *k* broj pripadnih *n*-grama cijele rečenice, što zapravo znači da izgrađuje *n*-gram koji je duljine rečenice.

Svaki takav izgrađeni *n*-gram smatra se rečenicom alternativnom originalnoj, dakle njezinim mogućim ispravkom.

Pod spajanjem dvaju *n*-grama misli se na izgradnju *(n+1)*-grama koji se sastoji od prve riječi prvog *n*-grama, *n – 1* riječi zajedničkih prvom i drugom *n*-gramu te posljednje riječi drugog *n*-grama.

Npr. *The book is very* + *book is very relevant* = *The book is very relevant*. Moguće je, dakle, spojiti samo one *n*-grame kojima se preklapa *n – 1* riječi, i to takvih da su to završne riječi prvog i početne riječi drugog *n*-grama.

Napišimo gornji algoritam još jednom, u pseudokodu:

```
za svaki alternativni n-gram x prvog n-grama
  lista = pronađi_listu (drugi_ngram, x)
  konačna_lista.dodaj( x + lista )
sortiraj_prema_ocjenama (konačna_lista)

pronađi_listu (y, x)
  ako (y je zadnji n-gram)
    za svaki alternativni n-gram z n-grama y
      ako (x se preklapa sa z)
        konačna_lista.dodaj( z )
  inače
    za svaki alternativni n-gram z n-grama y
      ako (x se preklapa sa z)
        lista = pronađi_listu (susjedni_desni_ngram, z)
        konačna_lista.dodaj( z + lista )
  vrati konačna_lista
```

Iznad opisani algoritam zapravo je radi preglednosti pojednostavljena inačica implementiranog algoritma. Ono što je u gornjem opisu izostavljeno je činjenica da algoritam ne odustaje od spajanja ni u slučajevima kada pretraživanje neke liste *n*-grama za *n*-gramima koji se preklapaju s nekim lijevim susjednim alternativnim *n*-gramom ne vrati nikakve rezultate.

U tom slučaju algoritam će ignorirati alternativne *n*-grame onog *n*-grama koji trenutno obrađuje i jednostavno "nasilno" spojiti posljednju riječ originalnog *n*-grama s listom koju je dobio kao argument, te nastaviti pretraživati udesno (ako već nije na posljednjem *n*-gramu). Ovakva operacija, međutim, neće proći bez kazne za ukupnu ocjenu tako generirane rečenice, pa će algoritam u generiranu listu zabilježiti koliko puta se takvo nešto dogodilo, što će imati vrlo negativan utjecaj na konačnu ocjenu.

3. 5. Ocjenjivanje alternativnih rečenica

Iznad smo opisali generiranje alternativnih rečenica, ali to nam ne govori ništa o tome kako odlučiti koja od njih bi bila najbolji izbor za zamjenu originala i je li uopće potrebno mijenjati original.

Tu odluku donosimo na temelju ukupne heurističke ocjene svake rečenice. Ona se između ostalog, računa iz pojedinih ocjena *n*-grama iz kojih je izgrađena, pa je najprije potrebno opisati ocjenjivanje samih *n*-grama.

Kako je iznad opisano, svaki *n*-gram ima svoju listu alternativnih *n*-grama od kojih svaki ima određen broj pojavljivanja koji je dobiven od izvora podataka. Svaka takva lista se nakon dohvata sortira prema tom broju pojavljivanja, a ocjene svih *n*-grama računaju se skaliranjem cijele liste tako da maksimalna ocjena bude 100, dakle prema sljedećoj formuli:

$$ocjena(x) = 100 * \frac{broj\_pojavljivanja(x)}{maksimalni\_broj\_pojavljivanja}$$

Ocjene se skaliraju kako bi svaki dio rečenice u kasnijem računanju ukupne ocjene za rečenicu imao jednaku ulogu. Time se, dakle, sprječava da se stavlja nepotreban naglasak na one dijelove rečenice čiji alternativni *n*-grami imaju veće brojeve pojavljivanja, jer ono što želimo postići pri računanju ocjena rečenica nije "nadmetanje" između pojedinih dijelova rečenice, već samo "nadmetanje" između različitih inačica nekog dijela rečenice.

Prije računanja ukupne ocjene alternativne rečenice, računaju se tri faktora od kojih se ocjena sastoji. Prvi je **prosjek ocjena (*A*)** njezinih sastavnih *n*-grama, a preostala dva su **faktor zamijenjenih riječi (*B*)** i **faktor prekida (*C*)**.

Faktor zamijenjenih riječi (*B*) u formulu unosi smanjenje ocjene koje je proporcionalno broju riječi koje su u ovoj inačici rečenice različite u odnosu na originalnu rečenicu. Budući da je jedna zamijenjena riječ ionako minimum koji je potreban da bismo imali alternativnu rečenicu, formula je napisana tako da je minimalan broj zamijenjenih riječi za koje ovaj faktor uopće može imati utjecaj na formulu jednak dva. Ako stvaran broj zamijenjenih riječi označimo sa *x*, formula glasi:

$$B = \begin{cases} x = 0, & 1 \\ x > 0, & \left[1 - \left(\frac{x-1}{duljina\_rečenice}\right)\right]^2 \end{cases}$$

Što je to prekid opisano je pri kraju prethodnog potpoglavlja, a ovdje ćemo se osvrnuti na utjecaj broja prekida na ocjenu rečenice. Kako algoritam izgrađuje alternativnu rečenicu tako bilježi i eventualne prekide koji mu se pritom dogode, a kasnije se pomoću njihovog broja računa **faktor prekida (*C*)**:

$$C = \left(1 - \frac{broj\_prekida}{duljina\_rečenice}\right)^4$$

Sada možemo napisati i ukupnu formulu za računanje ocjene rečenice. Sažeti oblik:

$$ukupna\_ocjena = A * B * C$$

Puni oblik:

$$ukupna\_ocjena =$$

$$AVG(ocjena_{ngram}) * \left[1 - \left(\frac{br\_zamj\_riječi - 1}{duljina\_rečenice}\right)\right]^2 * \left(1 - \frac{broj\_prekida}{duljina\_rečenice}\right)^4$$

Drugi faktor napisan je samo za slučaj kada je broj zamijenjenih riječi veći od nule (inače je njegova vrijednost jednaka 1).

Ovime završavamo pregled algoritma ocjenjivanja pojedine alternativne rečenice. Nakon što su sve rečenice ocijenjene, preostaje samo sortirati ih prema tim ocjenama. U slučaju da se na vrhu te liste nalazi originalna rečenica, algoritam zaključuje da je rečenica ispravna, a u suprotnom algoritam zaključuje da je ispravna ona rečenica s najvišom ocjenom.

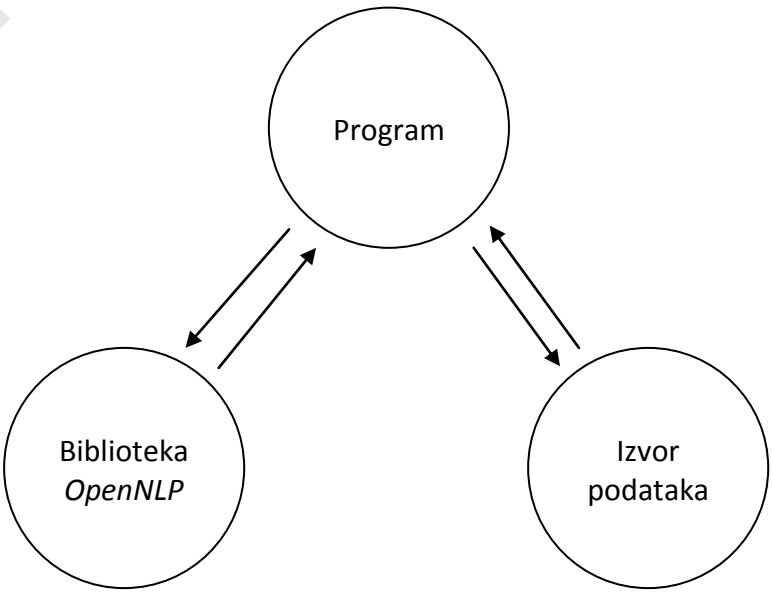
4. Programska izvedba

4. 1. Pregled

U ovom poglavlju opisana je kompletna struktura programa, uključena gotova rješenja koja se koriste pri obradi podataka i na sučelju te način organizacije podataka na kojima se program temelji i pristupa tim podacima.

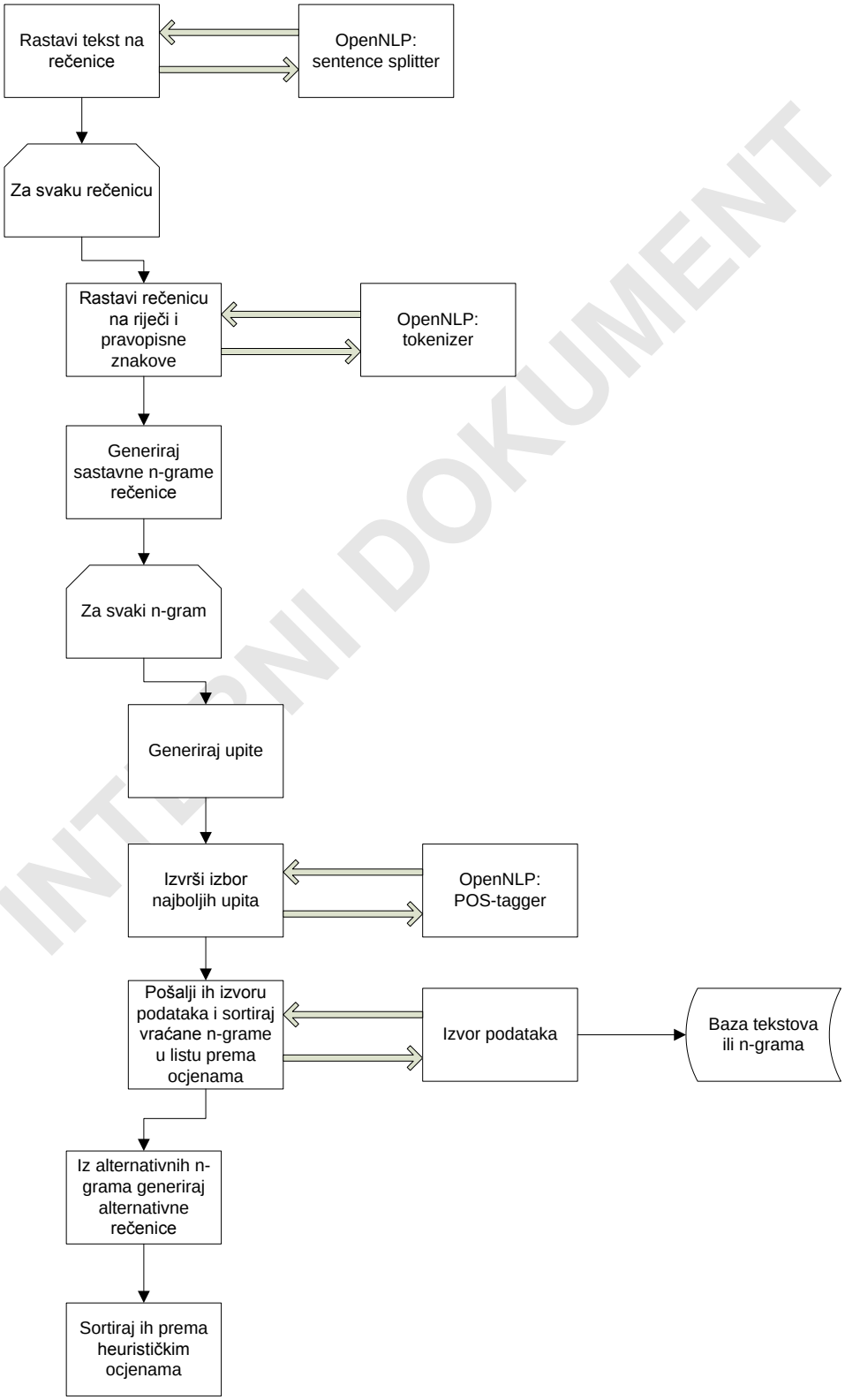
Program je zamišljen kao *web*-aplikacija i napisan je u C#-u kao dio ASP.NET projekta u Microsoft Visual Studiu 2008. Korisnik na *web*-sučelju unosi tekst čiji stil želi provjeriti. Tekst se zatim rastavlja na rečenice te se svaku rečenicu posebno obrađuje. Rečenica se rastavlja na *n*-grame kako je opisano u poglavlju 3.1. Slijedi generiranje upita objašnjeno u poglavlju 3.2 te slanje tih upita izvoru podataka. Postoje dva izvora podataka, implementirana pomoću dva različita razreda kojima se iz projekta pristupa preko unificiranog sučelja (*interface*) [7]. Jedan razred koristi podatke s Googleovog servisa *Google Scholar* [8], a drugi koristi bazu od oko 77 000 000 trigrama generiranih iz tekstova koji se nalaze u bazi *Project Gutenberg* [9].

Kada razred izvora podataka vrati sve pronađene *n*-grame s pripadajućim ocjenama, pristupa se generiranju i ocjenjivanju alternativnih rečenica (poglavlja 3.4 i 3.5) te, konačno, njihovom ispisu na sučelje gdje korisnik može odabrati pojedinu rečenicu i vidjeti njezine moguće ispravke poredane po heurističkim ocjenama.



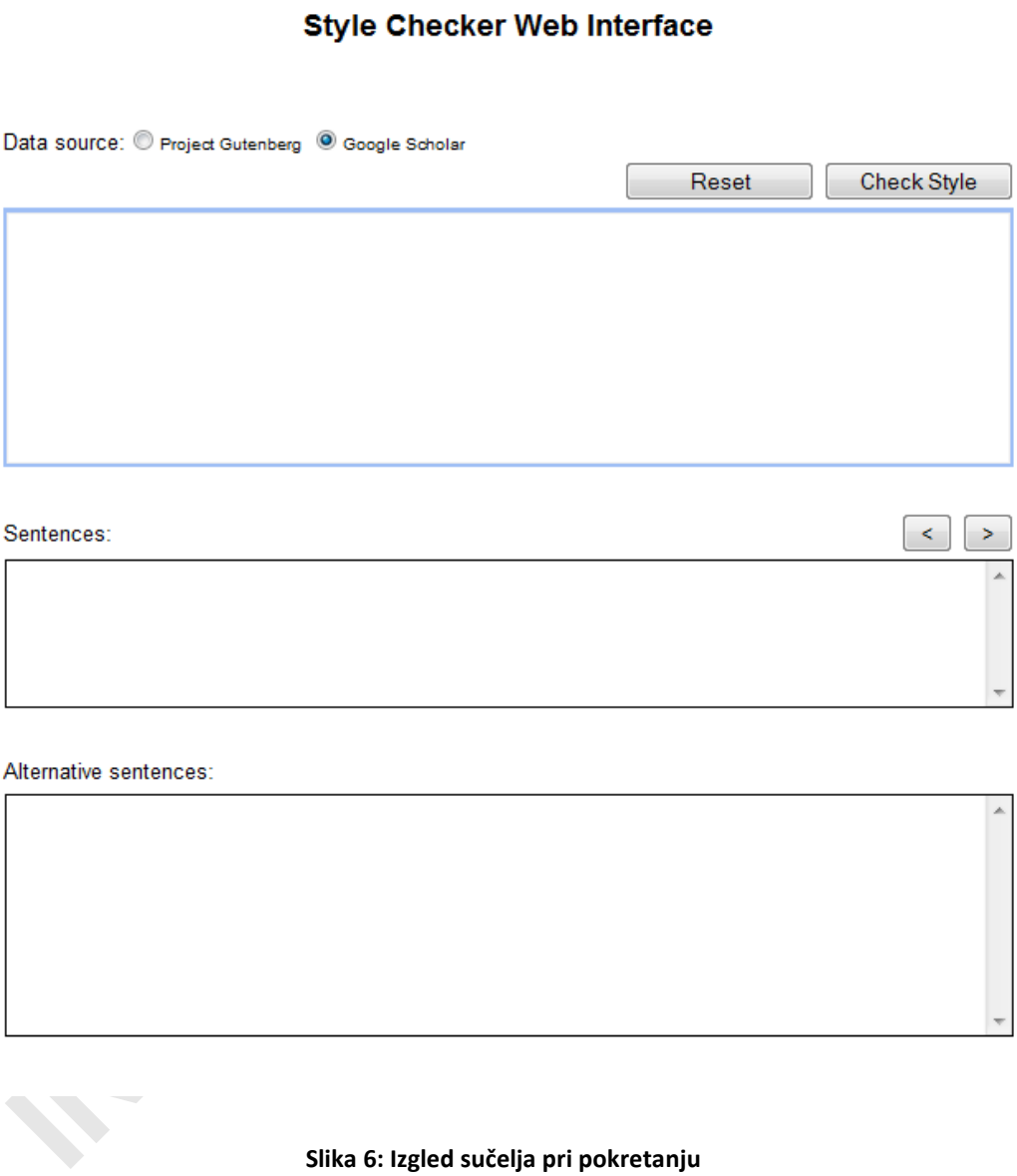
Slika 4: Osnovna organizacija projekta

Za poslove rastavljanja teksta na rečenice, rečenica na riječi te označavanja riječi oznakama vrste riječi koristi se biblioteka *OpenNLP*. Ovu najosnovniju organizaciju možemo vidjeti na Slici 4, a dijagram toka iznad opisanog procesa na Slici 5.



Slika 5: Dijagram toka rada programa

Izgled sučelja netom nakon pokretanja programa vidimo na Slici 6.



Slika 6: Izgled sučelja pri pokretanju

4. 2. Biblioteka *OpenNLP*

OpenNLP je biblioteka napisana u Javi, a sadrži alate za obradu prirodnog jezika. U ovom radu koristimo njezinu inačicu prevedenu u C# [10]. I originalna inačica i ova pisana u C#-u dostupne su pod LGPL-licencom. *OpenNLP* nije samo ime biblioteke, već i skupni naziv nekolicine projekata koji se bave obradom prirodnog jezika (NLP označava *natural language processing*). Alati koji se nalaze u biblioteci su rastavljač teksta na rečenice (engl. *sentence splitter*), tokenizator (engl. *tokenizer,* rastavlja tekst na pojavnice), označivač (engl. *POS-tagger*), plitki parser (engl. *chunker*, pronalazi sintatičke strukture

koje su dijelovi rečenice poput imeničkog skupa), parser (izgrađuje stablastu strukturu cijele rečenice) te pronalazač vlastitih imena (engl. *name finder*).

Odluke koje ova biblioteka donosi prilikom obrade teksta utemeljene su na modelima maksimalne entropije.

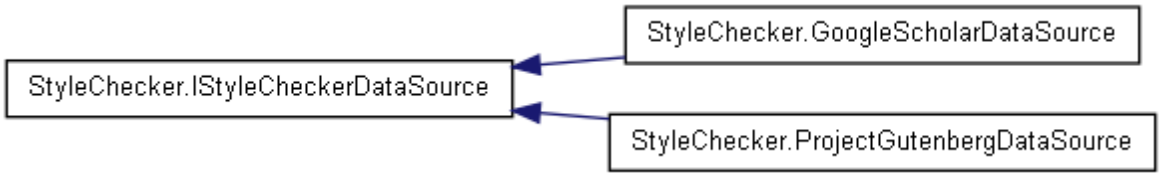
4. 3. Korišteni izvori podataka

Izvor podataka u ovom projektu zamišljen je kao bilo koji razred koji ima na raspolaganju nekakavu bazu tekstova, bilo lokalno bilo na Internetu i kojem je moguće slati upite poput:

- "kolika je učestalost pojavljivanja nekog *n*-grama?";
- "koji *n*-grami postoje u bazi, a da zadovoljavalju upite poput '*a b* * *c*'?", gdje *a*, *b* i *c* označavaju zadane riječi, a * označava bilo koju riječ.

Kako bi se povećala modularnost programa i olakšalo programiranje odlučeno je da se takvim razredima pristupa pomoću sučelja. Sučelje je nazvano *IstyleCheckerDataSource*, a ispod vidimo njegovu definiciju u kojoj imamo dvije metode koje odgovaraju gore definiranim upitima:

```
public interface IStyleCheckerDataSource
{
      int GetNumOccurences(TokenList text, List<string> tags);
      RequestResult GetResult(TokenList request, List<string>
      tags);
}
```



Slika 7: Dijagram nasljeđivanja za sučelje izvora podataka

Kako je spomenuto u poglavlju 4.1, koristimo dva izvora podataka. Prvi je implementiran razredom *GoogleScholarDataSource* i koristi podatke s *Google Scholara*, a drugi razredom *ProjectGutenbergDataSource* i koristi bazu *n*-grama s lokalnog diska.

4. 4. Google Scholar kao izvor podataka

Razred *GoogleScholarDataSource* prilikom rada programa izravno se spaja na *Google Scholar* koji se nalazi na URL-u http://scholar.google.com. S obzirom na akademski sadržaj, baza tekstova koju servis sadrži je prikladna za analizu stila stručnih i akademskih tekstova. Nažalost, za razliku od ostalih Googleovih servisa poput *Mapsa* ili *AdWordsa*, Google za ovaj servis nije razvio API (*Application Programming Interface*) koji bi programerima omogućio lakši pristup podacima koje servis sadrži [11], pa je za ovaj projekt bilo potrebno razviti metode koje će iz čistog HTML-a izvlačiti potrebne podatke.

Kao najveći nedostatak ovog izvora podataka pokazala se činjenica da Google nakon ne više od stotinu automatskih upita blokira IP-adresu klijenta koji ih šalje, što u praksi znači da programom možemo provjeriti ne više od desetak rečenica prije nego budemo blokirani.

Prije detaljnijeg opisa razreda *GoogleScholarDataSource*, pogledajmo njegovu deklaraciju s prikazom samo javnih članova:

```
public class GoogleScholarDataSource : IstyleCheckerDataSource
{
	public GoogleScholarDataSource(OpenNLPmethods onlp);
	public int GetNumOccurences(TokenList req, List<string> tags);
	public RequestResult GetResult(TokenList req, List<string> tags);
}
```

Metoda ***GetNumOccurences()*** za zadani tekst *req* vraća učestalost pojavljivanja tog izraza u bazi tekstova, dok metoda ***GetResult()*** vraća instancu razreda ***RequestResult***, u kojoj se nalazi lista pronađenih *n*-grama i učestalost pojavljivanja svakoga od njih.

Obje ove metode koriste dvije druge, privatne metode od kojih jedna služi za dohvat samog HTML-a, a kao parametar prima točan tekst upita koji se šalje *Google Scholaru*, a druga može parsirati taj HTML i iz njega saznati broj rezultata.

Ispod vidimo dio koda koji slaže točan niz znakova URL-a kojeg ćemo zatražiti s Interneta:

```
string strRequest =
	"http://scholar.google.com/scholar?start=" + 10 * pageNum
	+ "&q=" + s + "&hl=en&as_sdt=2000";
```

Do ovakvog oblika došlo se eksperimentalno tako što je autor jednostavno posjetio stranicu servisa i poslao neki upit, te pogledao kako izgleda adresa koja se pojavila u polju adrese internetskog preglednika. Uočeno je da parametar *start* određuje redni broj rezultata od kojeg želimo započeti prikaz, a kako je standardan broj rezultata koji se prikazuje jednak deset, očito je potrebno broj stranice koju želimo dohvatiti pomnožiti sa deset, što se i radi u gornjoj naredbi. Također je potrebno napomenuti da je ova naredba dio *for* petlje koja kao brojač koristi varijablu *pageNum* koja određuje koju stranicu rezultata trenutno dohvaćamo. Broj stranica koji se dohvaća je konstanta koja je u programskom kodu postavljena na četiri.

Iznimka od ovog broja stranica koje se dohvaćaju je slučaj kada nas zanima broj pojavljivanja gotovog izraza (ne tražimo alternativne *n*-grame pomoću upita koji sadrži zvjezdicu). U tom slučaju je broj stranica uvijek jedan, jer se već na prvoj stranici rezultata nalazi informacija o broju pojavljivalja *n*-grama, što će detaljnije biti prikazano kasnije.

Drugi parametar u adresi kojeg je potrebno ispuniti je sam tekst upita (u naredbi je to varijabla *s*). Ovaj tekst je potrebno staviti među dvostruke navodnike kako bi *Google Scholar* tražio točnu frazu, a ne i dokumente u kojima se riječi iz upita samo pojavljuju na raznim mjestima.

Treći dio niza znakova adrese je konstantan.

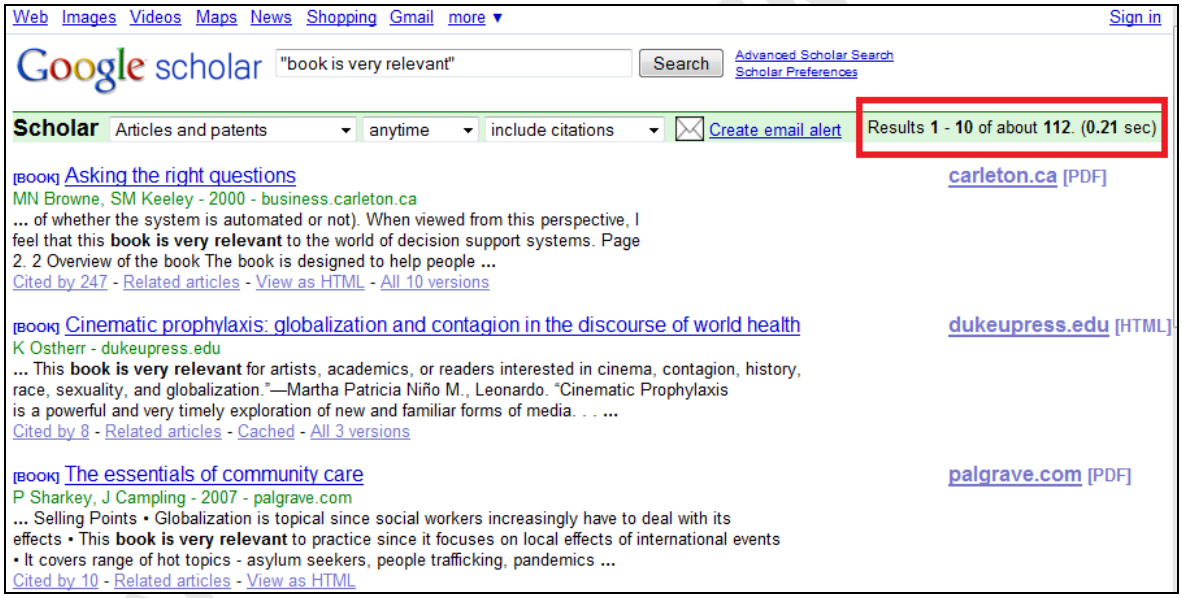
Slika 8: Sučelje Google Scholara – slanje upita

Slika sučelja iznad i upisanog 4-grama je analogna onome što radi upravo opisana metoda. Rezultati koje servis vrati bi u internetskom pregledniku izgledali kao na Slici 9.

Tamo je crvenim pravokutnikom označeno područje dokumenta u kojem se nalazi broj pojavljivanja zadanog *n*-grama. Taj dio koda u HTML-u izgleda ovako:

```
Results <b>1</b> - <b>10</b> of about <b>112</b>
```

Ideja za izvlačenje informacije koju tražimo (u ovom primjeru je to broj 112) bila je vrlo jednostavna. Najprije pronaći *string* koji tom broju prethodi (`Results <b>1</b> - <b>10</b> of about <b>`), a zatim *string* koji ga slijedi (prva pojava `</b>` nakon prvog *stringa*) te izdvojiti ono što ostane između. Veće brojeve *Google Scholar* prikazuje pomoću zareza (npr. 4,620,000), pa je prije pretvaranja izdvojenog *stringa* u *int* potrebno iz njega izbrisati sve zareze.



Slika 9: Rezultati upita poslanog Google Scholaru prikazani u internetskom pregledniku

Sad kada smo opisali ove dvije metode niže razine (dohvat HTML-a i dohvat broja rezultata iz tog HTML-a), opisat ćemo i dvije već spomenute javne metode više razine koje ih koriste.

4. 4. 1. Dohvat učestalosti pojavljivanja pojedinog *n*-grama

Ova metoda (***GetNumOccurences()***) jednostavno najprije poziva prvu metodu niže razine (dohvat HTML-a) uz parametar koji govori da se dohvaća samo jedna stranica s *Google Scholara* (više je nepotrebno, jer nas zanima samo broj pojavljivanja, ne i vraćeni dokumenti). Nakon toga poziva i drugu metodu, koja iz dobivenog HTML-a saznaje broj pojavljivanja.

4. 4. 2. Dohvat *n*-grama koji zadovoljavaju zadani upit

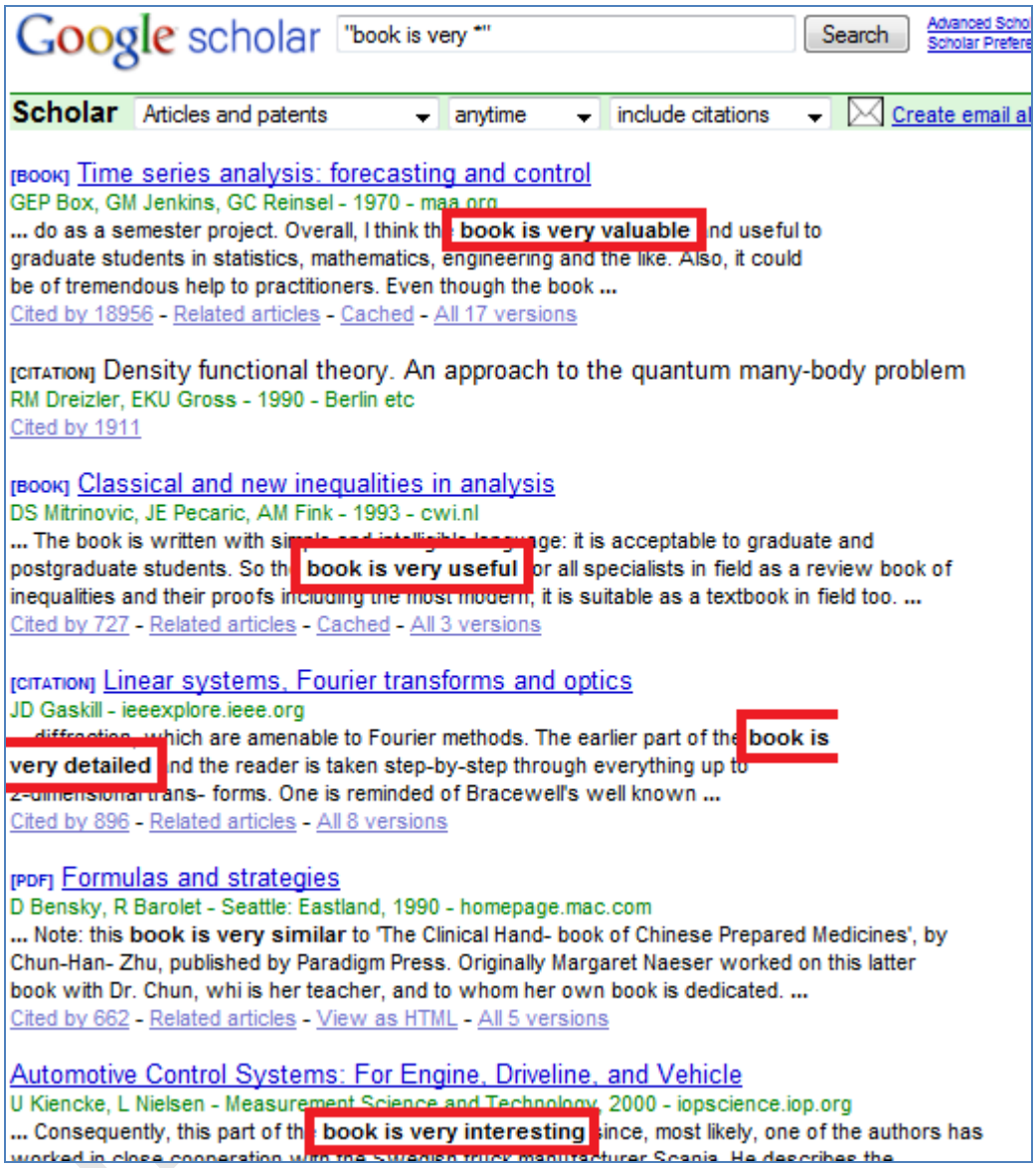
Dohvatom alternativnih *n*-grama za neki upit sa zvjezdicom bavi se metoda *GetResult()*. Ona najprije dohvaća konstantom zadani broj stranica HTML-a pomoću već opisane metode niže razine kojoj prosljeđuje primljeni upit, a onda u njemu pronalazi *n*-grame koji zadovoljavaju upit. Na Slici 10 vidimo kako bi izgledalo slanje upita *book is very ** preko standardnog sučelja *Google Scholara*.

Slika 10: Slanje upita sa zvjezdicom preko sučelja Google Scholara

Metoda niže razine dohvatit će nekoliko stranica rezultata u obliku HTML-a koje bi u internetskom pregledniku izgledale kao na Slici 11. Na toj slici je crvenim pravokutnicima označeno četiri od nekoliko alternativnih *n*-grama koji se mogu dobiti upitom *book si very **, a to su *book is very valuable, book is very useful, book is very detailed* i *book is very interesting*.

Izdvajanje ovih rezultata iz HTML-a započinje izbacivanjem svih HTML-oznaka tako da ostane samo tekst *web*-stranice. Ovo je vrlo jednostavno učiniti preko svojstva *body.innerText* razreda *HTMLDocumentClass*. Zatim je u tako dobivenom *stringu* potrebno pronaći sve pojave *n*-grama koji su traženog oblika.

Pretraga je implementirana pomoću regularnih izraza (razredi *Regex*, *MatchCollection* i *Match*) [12, 13]. Regularni izraz pomoću kojeg pretražujemo *string* izgrađuje se iz upita tako da se na mjestu na kojem se u upitu nalazi zvjezdica u izrazu nađe "`[^ ]+`", što označava bilo koji niz znakova koji nisu razmak. Tako bismo iz upita *book is very **, dobili regularni izraz "`book is very [^ ]+`".



Slika 11: Alternativni *n*-grami na stranici rezultata dobiveni upitom sa zvjezdicom

Ovakav regularni izraz nije savršen jer razmak nije univerzalni znak za odvajanje pojavnica, pa bi se u nekim vraćenim upitima moglo naći više od jedne pojavnice koje bismo dobili na mjestu zvjezdice. Moguće je, npr., dobiti ovakav *n*-gram: *book is very relevant.* (Ovdje je i točka dio *n*-grama.) Tu se, dakle, dogodilo da smo umjesto samo jedne pojavnice (*relevant*), dobili dvije (*relevant* i točku), jer se između zadnje riječi rečenice i završnog znaka rečenice (u ovom slučaju točke) ne piše razmak. Dobiveni *n*-gram je, naravno, za jednu pojavnicu veći od tražene duljine.

Taj se problem rješava tako da se provjeri duljina svakog vraćenog *n*-grama i ako je veća od duljine *n*-grama upita, onda se odbaci ili prva ili zadnja pojavnica, ovisno o tome

je li se pogreška dogodila na početku ili na kraju. Postavlja se pitanje što je s pogreškama koje se dogode negdje u sredini?

To je zasad riješeno tako da se u tom slučaju automatski odbaci zadnja pojavnica, a dalje se prepusti ostatku algoritma da takav *n*-gram eliminira iz upotrebe. To će se sigurno i dogoditi, jer će rečenice koje nastanu dijelom i iz tog *n*-grama imati značajno nižu ocjenu od ostalih te će završiti pri dnu liste, prema procesu ocjenjivanja opisanom u poglavlju 3.5.

Za svaki od ovih dobivenih *n*-grama je, prema prvotnoj ideji, kasnije u algoritmu bilo potrebno dohvatiti broj pojavljivanja koristeći metodu opisanu u prethodnom potpoglavlju (4.4.1). Međutim, kako je slanje svih tih dodatnih upita značajno usporavalo rad programa, autor je pokušao s nešto drugačijim pristupom.

Uočeno je da se neki *n*-grami na nekoliko prvih stranica rezultata *Google Scholara* mogu pojavljivati više puta, pa je taj broj pojavljivanja na stranicama rezultata iskorišten kao zamjena dohvatu broja pojavljivanja kojeg bi *Google Scholar* eksplicitno vratio ispisanog na stranici rezultata kad bi mu se kao upit predao svaki od tih *n*-grama pojedinačno. Ovako dobiveni brojevi skaliraju se tako da se najveći postavi na 100, a svi ostali proporcionalno njemu. Testiranja na nekoliko primjera pokazala su da tako izračunati brojevi mogu odstupati i do 30 % od onih dobivenih prvom metodom, ali to je cijena ubrzanja rada programa.

4. 5. Project Gutenberg kao izvor podataka

Razred *ProjectGutenbergDataSource* koristi bazu 77 000 000 trigrama koja je dobivena iz baze knjiga *Projekta Gutenberg* [9]. Ove *n*-grame izračunao je Prashanth Ellina te ih učinio dostupnima na svom *blogu.* Najprije je objavio komprimiranu datoteku s bigramima i trigramima knjiga iz svih jezika, veliku 624 MB [14], a zatim i datoteku koja sadrži samo engleske bigrame i trigrame veliku 493 MB [15]. Ta datoteka je uz neke veće prerade iskorištena kao izvor podataka u ovom projektu.

Jedan od nedostataka ove baze je ograničena duljina *n*-grama (maksimalna duljina je tri, što je i duljina koju pri obradi koristi čitavi program, ako je kao izvor podataka odabran *Project Gutenberg).*

Drugi nedostakak je taj da su u *n*-gramima kao pojavnice sadržane samo riječi, a ne i pravopisni znakovi, pa se svi upiti upućeni ovom izvoru podataka koji sadrže pravopisne znakove jednostavno ignoriraju.

Za razliku od ove, Googleova baza *n*-grama [16] sadrži i pravopisne znakove i *n*-grame većih duljina, ali ona tijekom izrade projekta autoru nije bila dostupna. Još jedna od prednosti Googleove baze je broj *n*-grama, jer sadrži npr. 977 000 000 trigrama u usporedbi sa samo 77 000 000 koliko sadrži korištena baza.

Dekomprimirana datoteka bigrama i trigrama *Projekta Gutenberg* velika je 2,41 GB, sortirana je po broju pojavljivanja pojedinog *n*-grama i formata je

```
x a b c,
```

gdje je *x* broj pojavljivanja *n*-grama, a *a, b* i *c* sastavne pojavnice. Isječak datoteke vidimo na Slici 12. Ovu je datoteku bilo potrebno obraditi tako da zadovolji zahtjeve projekta. Najvažniji zahtjev je bio sortiranje prema pojavnicama kako bi se omogućila binarna pretraga, jer bi sekvencijalno čitanje *n*-gram po *n*-gram jednostavno bilo neprihvatljivo sporo, ako ne i potpuno neupotrebljivo.

```
386 you may live
386 you not only
386 you to mr
386 you with this
386 your bow
386 youth at
386 zealous in
387 ' 8
387 ' de
387 ' replied she
387 ' said nicholas
387 60 feet
387 a 6
387 a book or
387 a british subject
387 a candlestick
```

Slika 12: Izgled originalne datoteke s *n*-gramima

Kako bismo detaljnije opisali ovaj zahtjev, raščlanit ćemo ga u nekoliko manjih.

Prvi je da imamo datoteku formata `a b c x` (sortiranu redom prema *a, b* i *c*) za slučaj kada želimo brzo pronaći trigram *a b c* i saznati njegov broj pojavljivanja *x*.

Drugi je da imamo datoteku formata `b c a x`, za slučaj kada pretražujemo *n*-grame prema upitu oblika ** b c*. U ovom slučaju nam je očigledno potrebno sortiranje prema *b* i *c*, jer je *a* nepoznata pojavnica na čijem mjestu prihvaćamo bilo koju pojavnicu.

Treći zahtjev je da imamo datoteku formata `a c b x`, za slučaj kada pretražujemo *n*-grame prema upitu oblika *a * c*. U ovom slučaju nam, analogno prethodnom, treba sortiranje prema *a* i *c*, jer je pojavnica na mjestu *b* nepoznata.

Četvrti zahtjev bi bio da imamo datoteku formata `a b c x`, za slučaj kada pretražujemo *n*-grame prema upitu oblika *a b **. Analogno dvama prethodnim slučajevima, treba nam sortiranje prema *a* i *b*, jer je *c* nepoznanica.

Vidimo da se četvrti i prvi zahtjev preklapaju, što znači da su nam potrebne ukupno tri datoteke koje treba generirati iz originalne. Sažetak iznad opisanih zahtjeva nalazi se u Tablici 2.

Tablica 2: Traženi oblici datoteke s *n*-gramima ovisno o tipu pretrage

Radnja	Zahtjevani oblik datoteke
Traženje *n*-grama *a b c*	`a b c x`
Upita oblika ** b c*	`b c a x`
Upit oblika *a * c*	`a c b x`
Upit oblika *a b **	`a b c x`

4. 5. 1. Generiranje potrebnih datoteka

U svrhu generiranja potrebnih datoteka napravljena je odvojena komandnolinijska C# aplikacija, s nekoliko procedura koje su redom izvršavane kako bi se dobile datoteke traženog oblika.

S obzirom na to da se u originalnoj datoteci osim trigrama, nalaze i bigrami, prvi korak bio je izbaciti bigrame. U tom istom koraku provedeno je i prebacivanje stupca s brojevima pojavljivanja sasvim desno, tako da je dobivena datoteka oblika:

```
a b c x,
```

ali koja je i dalje bila sortirana po *x*-u.

Jedan od problema s ovim podacima koji je uočen, je taj da su neki *n*-grami bili zapisani različito od konvencija biblioteke *OpenNLP,* tj. da su posvojni sufiksi (*'s*) bili zapisani zajedno s imenicom ili zamjenicom koja im prethodi te da su se skraćene forme glagola (*don't, he'll, I'm...*) također računale kao jedna pojavnica. Zato je sljedeći korak bio razdvajanje takvih "trigrama" koji su zapravo 4-grami, na dva zasebna trigrama. U Tablici 3 popisana su pravila provođenja takvih ispravaka. Primjer takvog nepravilnog "trigrama" je *he'll go to.* Prema trećem pravilu Tablice 3 bi takav *n*-gram bio razdvojen na *he 'll go* te na *'ll go to*. Oba ova trigrama "nasljeđuju" broj pojavljivanja *x* od 4-grama od kojeg su nastala.

Tablica 3: Pravila razdvajanja dvostrukih pojavnica

Dvostruka pojavnica	Odvojene pojavnice	
X's	X	's
Xn't	X	n't
X'll	X	'll
X're	X	're
X'm	X	'm
X've	X	've

Kako je u ovom procesu nastalo puno dvostrukih, pa i višestrukih *n*-grama, oni će biti očišćeni kasnije u procesu obrade na način da se za svaki *n*-gram koji se pojavljuje više puta ostavi samo ona pojava *n*-grama koja ima najviši *x* (broj pojavljivanja). Razlog zašto ih se nije čistilo odmah jest taj što je datoteka i dalje sortirana prama *x*-u, pa se višestruke pojave istog *n*-grama ne nalaze jedna do druge u datoteci, što je preduvjet za njihovo lagano pronalaženje.

Sljedeći korak je sortiranje prema ključu *a, b, c*. Ovo se nije moglo napraviti jednostavnim učitavanjem datoteke u listu i sortiranjem liste metodom *List.Sort()*, jer je datoteka prevelika. Zato je datoteka najprije rastavljena na manje datoteke od kojih je svaka sadržavala 10 000 000 linija. Svaka od njih je redom učitana u listu, sortirana te zapisana na disk, a zatim ih se spojilo u jednu veliku sortiranu datoteku sljedećim algoritmom:

```
za svaku datoteku datoteka
     linija = pročitaj_liniju(datoteka)
     lista.dodaj(linija)

dok (nisu sve datoteke zatvorene)
     l = prva linija sortirane liste
     zapiši(l, ciljna_datoteka)
     ako (izvorna_datoteka_od_l ima još linija)
          l = pročitaj(izvorna_datoteka)
     inače
          zatvori(izvorna_datoteka)
```

Gornji algoritam najprije pročita po jednu liniju iz svake manje datoteke i spremi ih u listu. Zatim pronalazi liniju liste koja je prva po redosljedu sortiranja i zapisuje je u ciljnu datoteku. Ako datoteka iz koje smo tu liniju pročitali ima još linija, čitamo sljedeću, stavljamo je na mjesto zapisane linije i pronalazimo novu prvu liniju liste te ponavljamo postupak dok ne pročitamo sve linije svih datoteka.

```
do anything unsupported 1
do anything until 44
do anything untoward 1
do anything unusual 2
do anything unjust 5
do anything up 1
do anything upon 3
do anything useful 14
do anything very 42
do anything vicious 2
```

Slika 13: Izgled datoteke s *n*-gramima nakon sortiranja po *a, b, c*

Nakon završetka sortiranja još samo očistimo višestruke *n*-grame i imamo sortiranu datoteku formata `a b c x`, koja je prva od tri tražene datoteke.

Preostale dvije datoteke dobit ćemo iz ove datoteke, tako da izvršimo odgovarajuće premještanje stupaca. To je realizirano u posebnoj metodi koja čita liniju po

liniju prve datoteke te svaku od njih zapisuje u dvije nove datoteke u izmjenjenim oblicima `b c a x` i `a c b x`.

Budući da je ideja bila pretraživati ove tri datoteke binarnom pretragom, pojavilo se i pitanje kako realizirati pretragu o obzirom na to da linije nisu iste duljine. Unatoč tome što postoji elegantno rješenje u vidu pozicioniranja na polovinu datoteke te vraćanja na početak linije traženjem prve pojave podniza "\r\n", uslijed nedostatka vremena autor se odlučio na proširivanje linija pomoću razmaka do veličine od 65 znakova. Po 20 znakova iskorišteno je za svaku od tri pojavnice, a dodatnih pet koristi se za broj pojavljivanja. Ako uračunamo i dva znaka za kraj linije ('\r' i '\n'), duljina je zapravo 67. Nedostatak ovog pristupa je značajno povećanje veličine datoteka, pa je tako pojedina datoteka s 1,49 GB narasla na 5,21 GB. U Tablici 4 je dan popis konačnih datoteka i pripadajućih formata.

Tablica 4: Popis konačnih datoteka s *n*-gramima

Datoteka	Format
ngrams123	`a b c x`
ngrams132	`a c b x`
ngrams231	`b c a x`

Na Slici 14 prikazan je dio datoteke *ngrams231*. Uočavamo da datoteka ima konstantnu duljinu linije, kao i da je sortirana po ključu *b, c, a*, što znači da je prvi trigram u popisu zapravo *walked impatiently about*, a ne *impatiently about walked* kako bi se moglo pomisliti na prvi pogled.

```
impatiently     about       walked      7
impatiently     about       walking     3
impatiently     above       eyes        3
impatiently     across      hand        4
impatiently     across      it          1
impatiently     across      line        2
impatiently     added       he          2
impatiently     added       she         2
impatiently     added       then        3
impatiently     addressed   bridget     1
impatiently     adopted     having      2
```

Slika 14: Izgled datoteke *ngrams231*

Budući da ove datoteke sadrže samo trigrame, u slučaju kada se kao izvor podataka koristi *ProjectGutenbergDataSource* sva obrada teksta radi se s trigramima, za razliku od standardno postavljene veličine *n*-grama u programu, koja je jednaka 4.

4. 5. 2. Pristup podacima iz datoteka

Kako je već spomenuto na početku poglavlja 4.3 razred *ProjectGutenbergDataSource* napravljen je prema predlošku sučelja izvora podataka *IstyleCheckerDataSource*. Deklaracija razreda s prikazanim samo javnim članovima izgleda ovako:

```
public class ProjectGutenbergDataSource : IstyleCheckerDataSource
{
	public int GetNumOccurences(TokenList req, List<string> tags);
	public RequestResult GetResult(TokenList req, List<string> tags);
	public static bool CheckWordSpelling(string word);
}
```

Metoda ***GetNumOccurences()*** dohvaća broj pojavljivanja zadanog trigrama *req* tako da binarnom pretragom pronađe taj trigram u datoteci *ngrams123*, iz pronađene linije izdvoji posljednje polje koje označava broj pojavljivanja, pretvori ga u *int* i vrati. Ako zadani trigram sadrži pravopisne znakove, tada se vraća broj –2 što je signal pozivnoj metodi da se pretraga ne može obaviti jer su trigrami sadržani u ovoj bazi ograničeni samo na riječi, kako je opisano u poglavlju 4.5. U slučaju da *n*-gram nije pronađen vraća se nula.

Metoda ***GetResult()*** vraća instancu razreda *RequestResult* koja sadrži pronađene *n*-grame koji zadovoljavaju upit *req* te pripadajuće brojeve pojavljivanja. Prva odluka koja se pri izvršavanju ove metode donosi je koju od tri dostupne datoteke s trigramima koristiti. Ovisno o mjestu na kojem se u upitu nalazi zvjezdica, otvara se datoteku *ngrams123*, *ngrams231* ili *ngrams132*. Odabrana datoteka se zatim pretražuje binarnom pretragom dok se ne pronađe prvi trigram koji zadovoljava zadani upit. Pronalaskom tog trigrama pretraga ne staje jer je sada potrebno pronaći i sve ostale zadovoljavajuće trigrame koji se nalaze u njegovom susjedstvu.

U prvoj fazi ostatka pretrage se čita trigram po trigram unatrag prema početku datoteke i svaki pročitani trigram se sprema u listu ako zadovoljava upit. Ako ne zadovoljava, pretraga se prekida i započinje se s drugom fazom. U njoj se provodi isti algoritam, samo se ide naprijed prema kraju datoteke počevši od prvog pronađenog trigrama.

Problem koji se pojavio pri ovakvoj pretrazi je pogrešna polazna pretpostavka autora da se svi trigrami koji zadovoljavalju neki upit nalaze u istoj skupini u datoteci, tj. da ne postoji trigram koji ne zadovoljava upit, a da se nalazi između dva trigrama koji ga zadovoljavaju. Iznenađujuće je da ta pretpostavka zapravo nije istinita unatoč tomu što su datoteke odgovarajuće sortirane. Uzrok su riječi koje sadrže apostrof. Naime, prema standardnom algoritmu sortiranja implemeniranom u platformi Microsoft .NET, sortiranje se očigledno provodi tako da se apostrofi najprije ignoriraju i *stringovi* se sortiraju kao da nemaju apostrofe, a tek ako su dva *stringa* jednaka, uzimaju se u obzir i apostrofi. Datoteke se pokušalo sortirati i na Linuxu (Ubuntu Linux 10.04) naredbom *sort* u komandnoj liniji, ali rezultati su bili isti. Takav algoritam dovodi do toga da imamo dijelove datoteke poput ovoga prikazanog na Slici 15.

julius	is	really	2
julius	is	reported	2
julius	is	sanguine	1
julius	is	schiller	2
'julius	is	somewhat	1
julius	is	strictly	2
julius	is	sure	2
julius	is	the	7

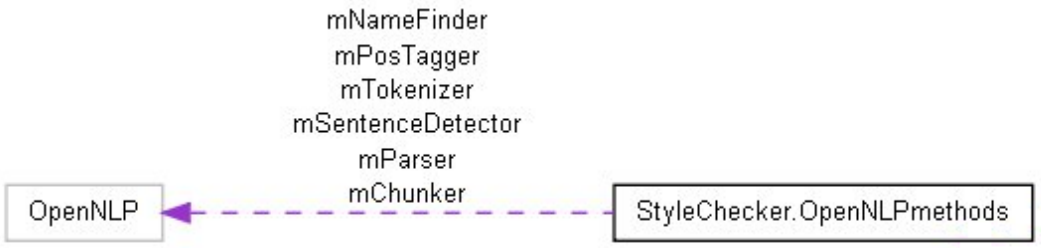
Slika 15: Primjer ignoriranja apostrofa pri sortiranju

Naravno, gore opisani algoritam pretrage će se na nekom primjeru poput ovog zaustaviti i neće pronaći druge trigrame koje slijede iza njega u pretrazi.

Problem je riješen na način da se svaki prekid pretrage dodatno testira tako da se provjeri sadrži li trigram kojim je pretraga završena apostrof. Ako je odgovor da, onda se ipak nastavlja sa sljedećim trigramom, a ovaj prethodni se jednostavno ne zapisuje u listu nađenih trigrama.

4. 6. Opis razreda i njihovih uloga

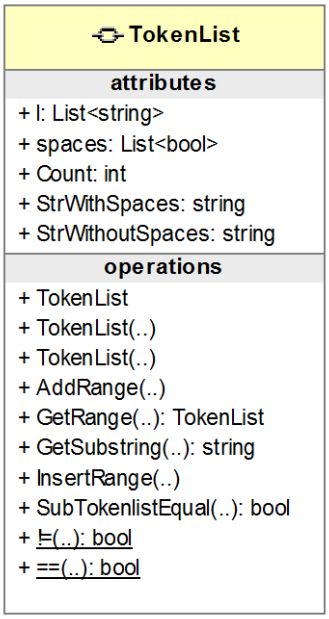
Budući da je algoritam provjere stila opisan u poglavlju 3, ovdje će biti opisani samo detalji njegove implementacije unutar programa. Cijeli program podijeljen je na pet datoteka s izvornim kodom. U datoteci *Constants.cs* definirane su sve konstante koje se se koriste; u datotekama *GoogleScholar.cs* i *ProjectGutenberg.cs* definirani su razredi izvora podataka koji su opisani u poglavlju 4.3; u datoteci *OpenNLPmethods.cs* definiran je razred *OpenNLPmethods* čije metode učahuruju neke metode biblioteke *OpenNLP* i tako olakšavaju njihovo korištenje, a u datoteci *Default.aspx* definirani su svi ostali razredi koji se koriste pri obradi teksta kao i razredi i metode za komunikaciju sa sučeljem, tj. korisnikom.



Slika 16: Dijagram pozivanja razreda *OpenNLPmethods* i biblioteke *OpenNLP*

4. 6. 1. Razred *TokenList*

Ovaj razred opisujemo prije svih ostalih jer ga koriste gotovo svi razredi. Potreba za postojanjem jednog ovakvog razreda postala je očitom pri samom početku izrade projekta. S obzirom na to da se u projektu bavimo analizom teksta, jasno je da *stringovi* ne mogu biti jedino rješenje za spremanje rečenica i *n*-grama. Kada se rečenice i *n*-grame sprema kao skupine pojavnica, onda je potrebno koristiti nekakvu listu. Razlog zašto ne koristimo običnu listu koju nudi *.NET framework* jest taj da je osim samih pojavnica potrebno spremati i informacije o tome nalazi li se između dvije susjedne pojavnice razmak.



Slika 17: Zaglavlje razreda *TokenList*

Razred *TokenList* je u tu svrhu napravljen da bude sličan razredu *List* uz dodatak da sprema i informacije o postojanju ili nepostojanju razmaka. Unutar njega nalaze se dvije liste; u jednoj su spremljene pojavnice, a druga je lista vrijednosti tipa *bool* koje govore nalazi li se ispred neke pojavnice razmak.

Instanca razreda može se inicijalizirati listom (u kojem slučaju se listi informacija o razmacima ne pridjeljuje nikakva vrijednost) ili drugim *TokenListom.*

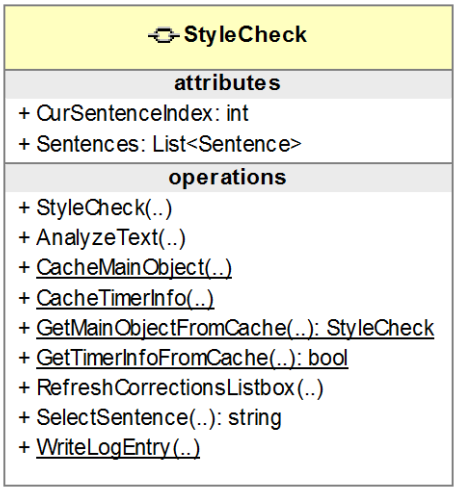
Iz *TokenLista* je moguće dobiti i *string* cijelog *n*-grama. U slučaju da želimo *string* u kojem su informacije o razmacima ignorirane (razmak se stavlja između svake dvije susjedne pojavnice) to činimo preko svojstva ***StrWithoutSpaces,*** a u suprotnom dohvaćamo svojstvo ***StrWithSpaces***.

Metoda ***GetSubstring()*** vraća samo dio *stringa* kojeg bi inače vratilo svojstvo *StrWithSpaces*. Implementirane su i metode ***AddRange()***, ***InsertRange()***, ***GetRange()*** ekvivalentne istoimenim metodama razreda *List,* kao i operatori ***==*** i ***!=*** te svojstvo ***Count***.

Tu je još i metoda ***SubTokenListEqual()*** koja uspoređuje dvije podliste zadanih *TokenLista*.

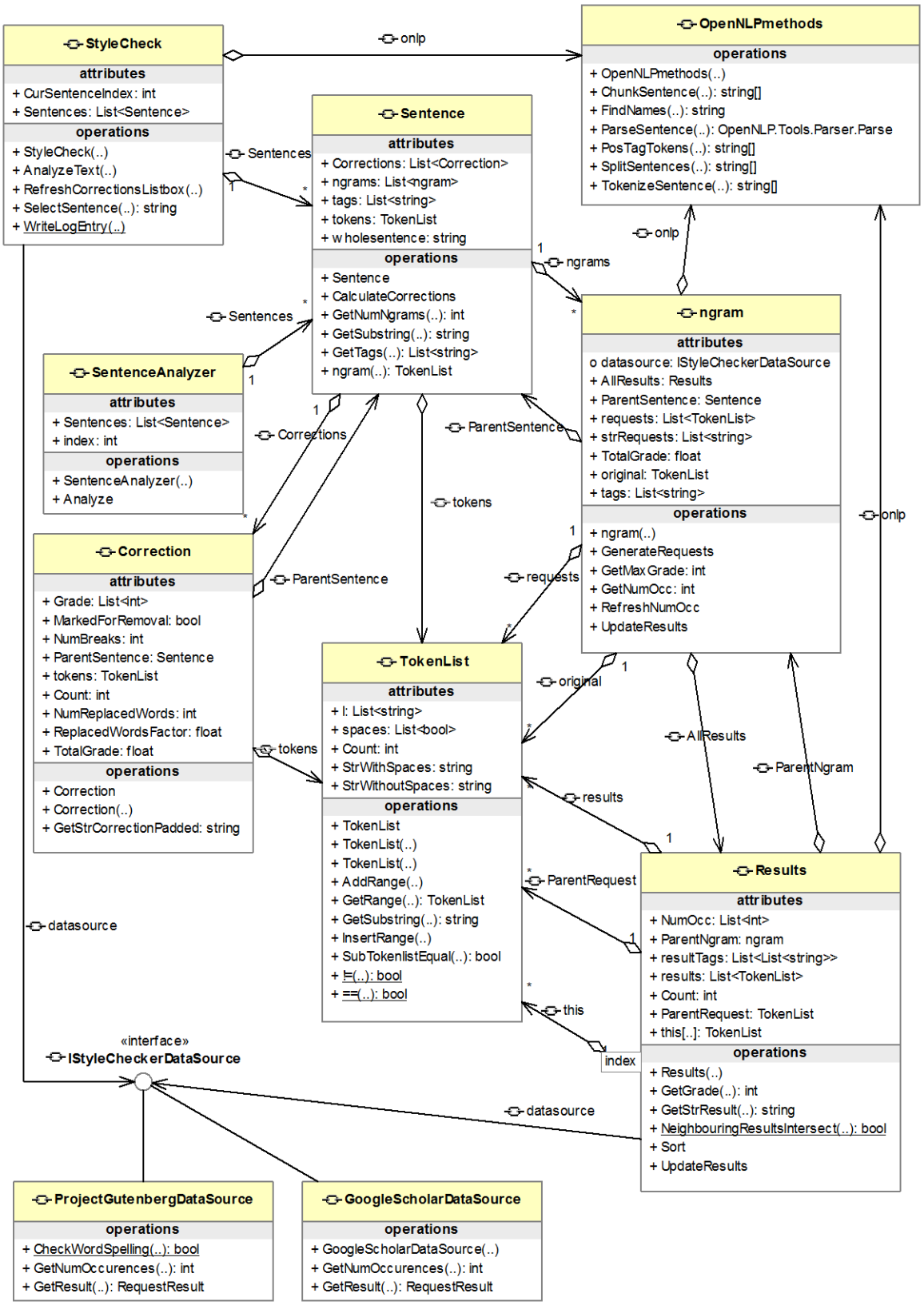
4. 6. 2. Razred *StyleCheck*

Razred *StyleCheck* nalazi se na najvišoj razini organizacijske hijerarhije projekta. On prima zahtjeve od korisnika preko sučelja, obrađuje zadani tekst pomoću ostalih razreda te ispisuje rezultate obrade. Upisom teksta u okvir nakon pokretanja aplikacije i odabirom gumba ***Check Style***, pokreće se metoda u razredu *_Default* koja je pridružena tom gumbu.



Slika 18: Zaglavlje razreda *StyleCheck*

_Default je razred čije je zaglavlje automatski generirano od strane razvojnog okruženja. Nasljeđuje razred ***System.Web.UI.Page*** koji sadrži osnovne elemente koji predstavljaju *web*-sučelje ASP.NET aplikacije.



Slika 19: Dijagram razreda

Metoda pridružena gumbu *Check Style* stvara instancu razreda *StyleCheck* te poziva njegovu metodu ***AnalyzeText()*** u kojoj se (pozivima ostalih razreda) događa veći dio cjelokupne obrade danog teksta. U ovom potpoglavlju će obrada biti opisana manje detaljno i s više razine, redom kojim se vrše pozivi metoda razreda niže razine, dok ćemo u nekoliko sljedećih potpoglavlja vidjeti detalje implementacije korištenih razreda ***Sentence*** (predstavlja rečenicu)*,* ***ngram*** (predstavlja *n*-gram)*,* ***Results*** (predstavlja listu alternativa nekog *n*-grama) te ***Correction*** (predstavlja pojedinačnu alternativnu rečenicu generiranu algoritmom provjere stila).

Prvi korak metode *AnalyzeText()* je instanciranje sučelja izvora podataka *IStyleCheckerDataSource*. Ovisno o tome je li na grafičkom sučelju odabran *Google Scholar* ili *Project Gutenberg* sučelju izvora podataka se pridružuje instanca razreda *GoogleScholarDataSource* ili *ProjectGutenbergDataSource.*

Zatim se pomoću metode *OpenNLPmethods.SplitSentences()* tekst rastavlja na rečenice. Pomoću svakog *stringa* rečenice stvara se instanca razreda *Sentence* te se sve takve instance spremaju u listu.

Svaka rečenica u listi obrađuje se zasebno te na sljedeći način: Najprije se metodom *OpenNLPmethods.Tokenize()* vrši tokenizacija, dobivene pojavnice se spremaju u *TokenList* sadržan u razredu *Sentence*, a potom se pojavnice i označavaju (metodom *OpenNLPmethods.PosTagTokens()*).

Vrši se provjera sadrži li rečenica više od 20 pojavnica, i ako je tako od obrade se odustaje. U suprotnom računa se broj *n*-grama koje rečenica sadrži, za svaki od njih stvara se instanca razreda *ngram* te se sve takve instance spremaju u listu *n*-grama sadržanu u razredu *Sentence* (prilikom stvaranja instance razreda *ngram,* njezina lista pojavnica nije eksplicitno stvorena, ali ju se može dohvatiti preko svojstva razreda *ngram*)*.* Nad svakom od njih dodatno se poziva metoda ***ngram.GenerateRequests()*** koja stvara listu upita sa zvjezdicama koji će kasnije biti poslani izvoru podataka. Svaka takva lista spremljena je u svoju instancu razreda *ngram.*

Ovime metoda *AnalyzeText()* završava svoj rad i kontrola se vraća korisniku, ali obrada teksta još nije gotova. Razlog zašto cijela obrada nije u ovoj metodi je sprječavanje

situacije u kojoj bi korisnik morao čekati da se sve rečenice obrade, prije nego vidi rezultate za prvu rečenicu.

Zato je uveden dodatni razred ***SentenceAnalyzer*** čija metoda ***AnalyzeSentence()*** vrši analizu samo jedne rečenice, a tu metodu se poziva preko instance razreda *Timer* koja "otkucava" svakih pola sekunde[2]. Pri svakom otkucaju *Timer* provjerava je li obrada trenutne rečenice gotova i, ako jest, poziva metodu *SentenceAnalyzer.AnalyzeSentence()* da obradi sljedeću.

Informacija o trenutno obrađivanoj rečenici kao i o tome je li obrada gotova čita se iz privremenog spremnika internetskog preglednika, gdje se i sprema od strane metode *AnalyzeText().* Opis ovog postupka dan je u pseudokodu:

```
ako (obrada prethodne rečenice gotova)
      izađi

spremi_u_cache(obrada_nije_gotova)
dohvati_iz_cachea(rečenica)
obradi(rečenica)
spremi_u_cache(rečenica)
ako (nema više rečenica)
      ugasi_timer
inače
      spremi_u_cache(obrada_gotova)
```

Sama obrada rečenice vrši se u dva koraka. Prvi je pozivanje metode ***ngram.UpdateResults()*** nad svakim *n*-gramom rečenice čime se dobivaju alternativni *n*-grami koristeći prije generirane upite. Drugi korak je pozivanje metode ***Sentence.CalculateCorrections()*** koja računa alternativne rečenice.

4. 6. 3. Razred *Sentence*

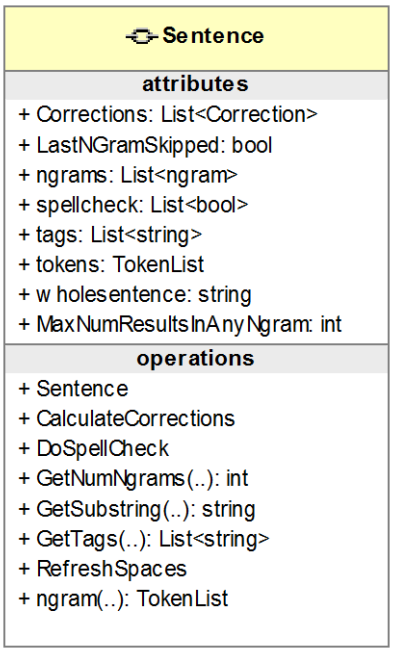
Razred *Sentence* predstavlja rečenicu. Informacije koje sadrži su *string* s tekstom čitave rečenice (*wholesentence*), lista pojavnica (*tokens*), lista oznaka pojavnica (*tags*), lista sadržanih *n*-grama (*ngrams*) i lista u kojoj se nakon obrade nalaze alternativne rečenice (*Corrections*). Opisat ćemo samo neke metode i svojstva čije značenje ili implementacija nisu očiti.

[2] Autor je ovaj problem pokušao riješiti i pomoću višedretvenosti, ali s obzirom na probleme na koje je naišao, a koji su vjerojatno posljedica neiskustva u radu s kombinacijom višedretvene aplikacije i *web*-sučelja, odlučeno je da se problem riješi pomoću razreda *Timer.*

Metoda ***GetNumNgrams()*** vraća broj *n*-grama koje rečenica sadrži. Taj broj ovisan je o konstanti koja određuje korištenu duljinu ngrama (*defaultNgramsize*) i računa se prema formuli:

$$duljina = broj\_pojavnica - n + 1,$$

gdje je *n* duljina *n*-grama. Ako je broj pojavnica manji od *n,* vraća se nula.



Slika 20: Zaglavlje razreda *Sentence*

Metoda ***ngram(int index, int n)*** vraća *n*-gram u obliku *TokenListe* koja sadrži pojavnice od pozicije *index* do pozicije *index + n – 1*.

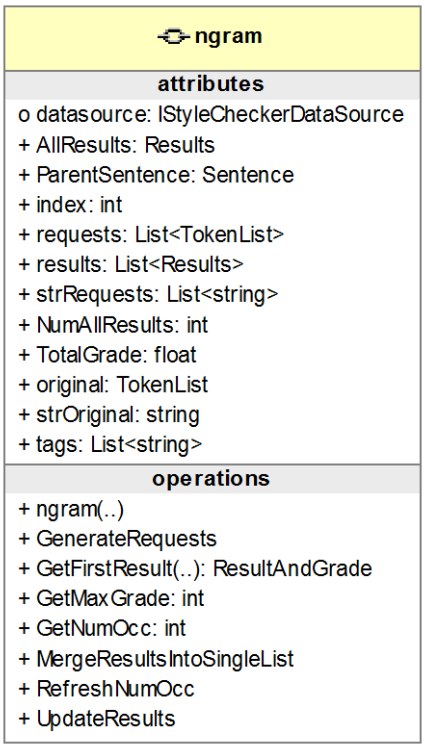
Metoda ***DoSpellCheck()*** svaku riječ rečenice traži u bazi trigrama u izvoru podataka *ProjectGutenbergDataSource,* i ovisno o tome da li je pronađena bilježi *true* ili *false* u listu *spellcheck*. Iako je implementirana, rezultati ove metode nisu iskorišteni, ali u nekoj poboljšanoj inačici projekta bi mogli biti.

Obrada jedne rečenice se vrši tako da se u metodi *SentenceAnalyzer.Analyze()* nad svakim njezinim *n*-gramom poziva metoda *ngram.UpdateResults()*, koja dohvaća alternativne *n*-grame iz izvora podataka i zapisuje ih u listu (ovo će detaljnije biti opisano u sljedećem potpoglavlju). Nakon što su alternativni *n*-grami spremljeni, nad rečenicnom se poziva metoda ***Sentence.CalculateCorrections()*** koja računa alternativne rečenice i njihove ocjene.

Ona najprije za svaki *n*-gram ujedinjuje njegove alternativne *n*-grame u jednu veliku listu, jer su ti *n*-grami najprije spremljeni u zasebne liste od kojih svaka odgovara jednom upitu sa zvjezdicom. Kada je to učinjeno, poziva se privatna metoda koja je rekurzivna implementacija algoritma generiranja rečenica iz *n*-grama opisanog u poglavlju 3.4.

Dobivene rečenice sortiraju se prema ocjeni (proces ocjenjivanja opisan je u poglavlju 4.6.6 koje se bavi razredom *Correction*) i time završava obrada jedne rečenice.

4. 6. 4. Razred *ngram*



Slika 21: Zaglavlje razreda *ngram*

Razred *ngram* zamišljen je tako da predstavlja jedan *n*-gram. Osnovne informacije koje sadrži su roditeljska rečenica (*Sentence* ***ParentSentence***) te vlastiti redni broj u toj rečenici (*int* ***index***).

Upiti sa zvjezdicama koje generira metoda ***GenerateRequests()*** spremljeni su u dvije liste (*List<TokenList>* ***requests*** te *List<string>* ***strRequests***). Ta metoda provodi algoritam generiranja upita opisan u poglavlju 3.2. Najprije se, koristeći drugu rekurzivnu privatnu metodu, generiraju kombinacije upita u kojima je jedna od pojavnica zamijenjena zvjezdicom. Zatim se provodi izbor tih upita, tako da se izbacuju oni upiti u kojima zvjezdica zamjenjuje pojavnicu koja je u jednoj od prve dvije kategorije vjerojatnosti prikladnosti za zamjenu, opisane u Tablici 1 u poglavlju 3.2.

Ove kategorije spremljene su kao polja konstanti u razredu *Constants*:

```
//interpukcija i znakovi za novac ($=dolar, #=funta)
public static string[] tags0 = {
      "``", ",", "''", ".", "$", "#", "-LRB-", "-RRB-", "LS"
};

//"loše" riječi
public static string[] tags1 = {
      "CD", "FW", "UH", "NN", "NNP", "NNPS", "NNS", "SYM", "RP", "EX",
      "POS"
};

//"srednje" riječi
public static string[] tags2 = {
      "VB", "VBD", "VBG", "VBN", "VBP", "VBZ", "DT", "PDT", "PRP",
      "PRP$",
                        };
//"dobre" riječi
public static string[] tags3 = {
      "JJ", "JJR", "JJS", "RB", "RBR", "RBS", "CC", "IN", "MD", "WDT",
      "WP", "WP$", "WRB", "TO"
};
```

Za razliku od liste upita, lista pojavnica koje *n*-gram sadrži te *string* teksta *n*-grama nisu eksplicitno spremljeni u instancu razreda, već se dobivaju preko svojstava koja te

informacije dohvaćaju od roditeljske rečenice (*TokenList* ***original*** i *string* ***strOriginal***). Isto vrijedi i za listu oznaka pojavnica (*List<string>* **tags**). Primjer takvog dohvata vidimo u kodu svojstva *TokenList original*:

```
public TokenList original
{
   get
   {
      return ParentSentence.ngram(index,
             Constants.GetNgramSize(datasource));
   }
}
```

Metoda ***GetNumOcc()*** od izvora podataka dohvaća broj pojavljivanja teksta *n*-grama u bazi, koji se istovremeno vraća kao povratna vrijednost i sprema u privatnu varijablu *numOcc*. Ako se metoda pozove više od jednom, upit izvoru podataka slat će se samo prvi put, kako bi se izbjeglo nepotrebno višestruko slanje.

Svojstvo ***TotalGrade*** jednostavno vraća taj broj pojavljivanja. Ovo svojstvo poziva se iz instance roditeljske rečenice kada se računaju ocjene generiranih alternativnih rečenica.

Dohvat alternativnih *n*-grama provodi metoda ***UpdateResults()***. Ona stvara listu čiji svaki element predstavlja popis rezultata (alternativnih *n*-grama) dobivenih za jedan upit. Lista upita predstavljena je razredom *Results* (koji će biti opisan u sljedećem potpoglavlju), pa je cjelokupna lista alternativnih *n*-grama definirana kao *List<Results>* ***results***. Za svaki upit stvara se po jedna instanca razreda *Results*, dodaje ju se u listu *results* i nad njom poziva metoda *Results.UpdateResults()* koja iz izvora podataka dohvaća alternativne *n*-grame odgovarajućeg upita.

4. 6. 5. Razred *Results*

Svaka instanca razreda *Results* predstavlja skupinu alternativnih *n*-grama dobivenih od jednog upita sa zvjezdicom. *N*-grami su spremljeni u listi *Lis<TokenList>* ***results***, a pristupa im se preko *indexera*:

```
public TokenList this[int index]
{
    get { return results[index]; }
}
```

Ako *n*-gram želimo dohvatiti u obliku *stringa,* tada pozivamo metodu ***GetStrResult()***. Lista oznaka pojavnica pojedinih *n*-grama spremljena je u *List<string>* ***tags***, a željene oznake mogu se dohvatiti metodom ***GetResultTags().*** Referenca na originalni *n*-gram nalazi se u javnoj varijabli *ngram* ***ParentNgram***. Upit kojim su *n*-grami dobiveni može se dohvatiti svojstvom ***ParentRequest:***

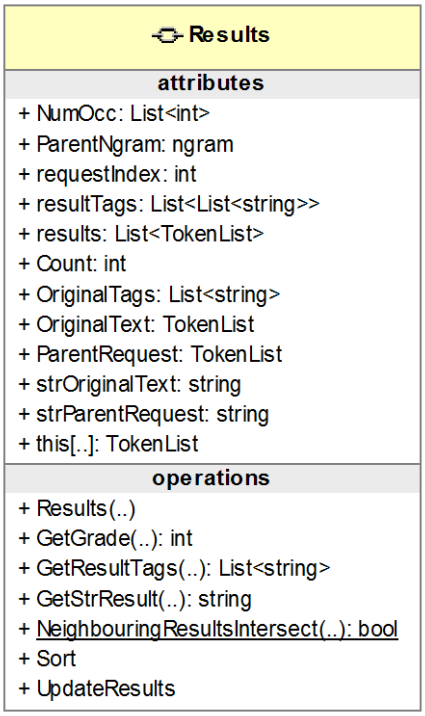
```
public TokenList ParentRequest
{
    get { return ParentNgram.requests[requestIndex]; }
}
```

Upit u obliku *stringa* dohvaća se sličnim svojstvom *string* ***strParentRequest***.

Moguće je dohvatiti i tekst originalnog *n*-grama (svojstva *TokenList* ***OriginalText*** i ***strOriginalText***), te njegove oznake (*List<string>* ***OriginalTags***).

Brojevi pojavljivanja pojedinih *n*-grama spremljeni su u listu *List<int>* ***NumOcc***, a dohvaća ih se metodom ***GetGrade().***

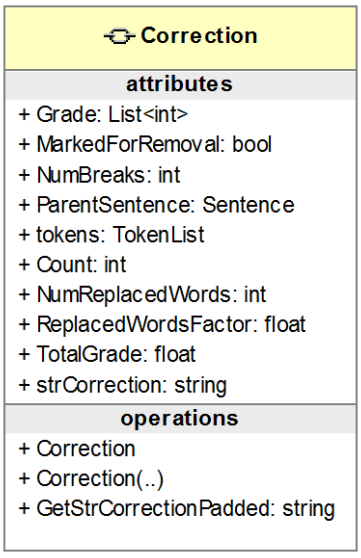
Prilikom prvog stvaranja instance razreda *Results* jedini dostupni podaci su roditeljski upit i originalni *n*-gram. Zadatak dohvaćanja alternativnih *n*-grama te njihovih ocjena ima metoda ***UpdateResults()***. Ona se spaja na izvor podataka te dohvaća sve pronađene alternativne *n*-grame i njihove ocjene slanjem roditeljskog upita sa zvjezdicom, a nakon toga ih sortira prema ocjenama.



Slika 22: Zaglavlje razreda *Results*

4. 6. 6. Razred *Correction*

Ovaj razred prestavlja jednu alternativnu rečenicu generiranu iz *n*-grama dobivenih od izvora podataka. Spremljene informacije su lista pojavnica (*TokenList* ***tokens***), referenca na originalnu rečenicu (*Sentence* ***ParentSentence***) i lista brojeva pojavljivanja *n*-grama od kojih je alternativna rečenica složena (*List<int>* ***Grade***). Tekst rečenice može se dohvatiti svojstvom *string* ***StrCorrection***.



Slika 23: Zaglavlje razreda *Correction*

Najvažnije svojstvo ovog razreda je *float* ***TotalGrade***. Ono vraća ukupnu heurističku ocjenu rečenice. Prije računanja ocjene potrebno je izračunati tri faktora od kojih se ocjena sastoji, kako je opisano u poglavlju 3.5. Svojstvo *TotalGrade* izgleda ovako:

```
public float TotalGrade {
    get {
        Grade.RemoveAll(GradeNotValid);
        float a = (float)(Grade.Average());
        float b = ReplacedWordsFactor;
        float c = (1.0F - (float)NumBreaks / Count);
        c = (float)Math.Pow(c, 4);
        return a * b * c;
    }
}
```

Najprije se izbacuju sve nevaljane ocjene (ocjene koje izvor podataka nije mogao dohvatiti označene su vrijednošću –2). Razlog postojanja nevaljanih ocjena opisan je u trećem odlomku poglavlja 4.5 i odnosi se samo na izvor podataka *ProjectGutenbergDataSource.* U drugoj naredbi računa se **prosjek ocjena** *n*-grama (faktor A). U trećoj naredbi dohvaća se **faktor broja zamijenjenih riječi**. On se računa u svojstvu *float* ***ReplacedWordsFactor*** prema formuli iz poglavlja 3.5. **Faktor broja prekida** (C) računa se u 4. i 5. naredbi te se sva tri faktora množe, čime se dobiva **ukupna ocjena** koja postaje povratna vrijednost svojstva ***TotalGrade***.

4. 7. Zadane konstante

Program koristi nekoliko konstanti, a sve su definirane u statičkom razredu ***Constants*** u datoteci *Constants.cs*. Slijedi opis pojedinih konstanti i njihovih vrijednosti kako su postavljene u konačnoj konfiguraciji programa:

- *GoogleScholarNumPages (int),* vrijednost **4** – odnosi se samo na izvor podataka *GoogleScholarDataSource* i određuje broj stranica rezultata koje se dohvaćaju pri traženju alternativnih *n*-grama;
- *defaultNgramsize (int)*, vrijednost **4** – određuje duljinu *n*-grama s kojom se radi tijekom cijelog postupka obrade teksta (ova konstanta se zanemaruje ako je kao izvor izabran *ProjectGutenbergDataSource*, jer u tom slučaju na raspolaganju imamo samo trigrame);

- *MaxResultsPerNgram (int),* vrijednost **30** – određuje koliki je maksimalan broj alternativnih *n*-grama nekog *n*-grama koje predajemo algoritmu generiranja rečenica. Ova konstanta služi sprječavanju prevelikog opterećenja algoritma;
- *MaxSentenceSize (int),* vrijednost **20** – određuje maksimalnu duljinu rečenice (broj pojavnica).

Postoje još i polja koja kategoriziraju oznake vrste riječi u četiri skupine, ali to je već opisano u poglavlju 4.6.4 koje se bavi razredom *ngram.*

Program tijekom rada dodatno sprema i dnevnik izvođenja. Dnevnik se sprema u radnu mapu pod imenom *StyleCheck.log*. Primjer dnevnika nalazi se u Dodatku B.

4. 8. Instalacija i korištenje

Program je zamišljen tako ga da se koristi kao običnu *web*-stranicu, dakle posjetom u internetskom pregledniku. U polje za unos teksta upisuje se tekst za provjeru, koja započinje odabirom gumba *Check Style*. Zatim se u izborniku ispod odabire rečenica čije alternative se želi pogledati.

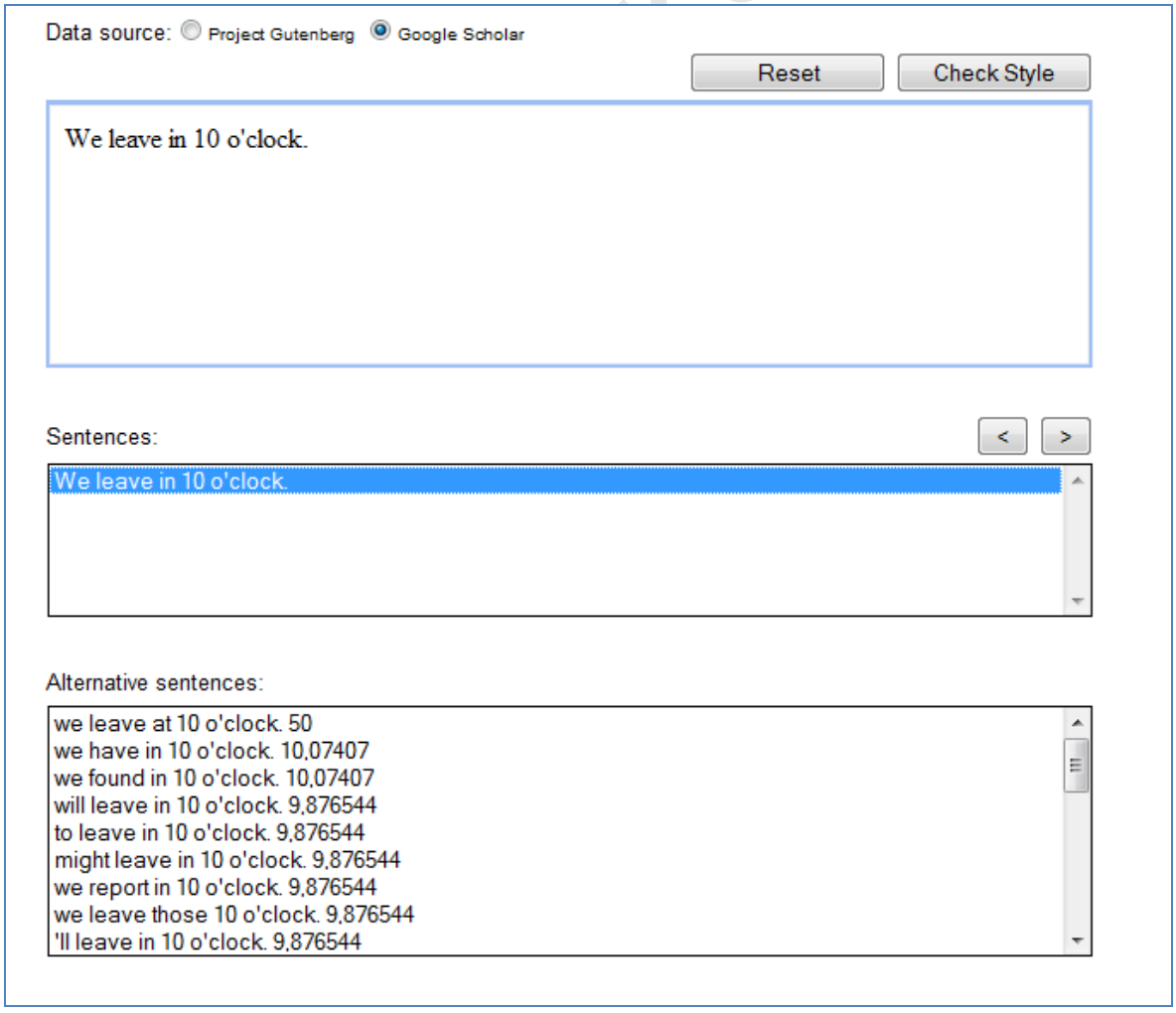
U slučaju da je program potrebno pokrenuti lokalno, to se može učiniti na dva načina. Prvi je učitavanje izvornog koda u Microsoft Visual Studio 2008 i odabir izvršavanja (*Run*). Ovime se automatski pokreće poslužitelj koji pokreće program te otvara sučelje u internetskom pregledniku. Drugi način je instalacija izvršne inačice programa u Microsoft Internet Information Services 4.0 ili noviji. Instalacija se vrši pokretanjem instalacijske datoteke *StyleCheckerWebSetup.msi* iz mape IzvrsniKod na CD-u.

Pri instalaciji u IIS potrebno je dodatno kopirati mapu s datotekama koje sadrže trigrame u radnu mapu programa. To znači da se datoteke moraju nalaziti u putanji "ProjGutenbergNgramFiles\..." relativno na radnu mapu. Na priloženom CD-u te datoteke nalaze se u mapi izvornog koda ("IzvorniKod\StyleChecker\ProjGutenbergNgramFiles\") u komprimiranom obliku, pa ih je prije upotrebe potrebno dekomprimirati. Naravno, ove korake vezane za datoteke s *n*-gramima potrebno je činiti samo u slučaju kada želimo koristiti *Project Gutenberg* kao izvor podataka.

5. Primjeri rada programa

U ovom poglavlju dano je nekoliko primjera ulaznih i izlaznih rezultata programa. Ulazne rezultate nećemo zadavati kao tekstove (iako je i to naravno moguće), već samo kao rečenice, jer ionako je obrada svake rečenice neovisna. Ono što će se iz tih primjera moći vidjeti jest da program prema trenutnoj konfiguraciji ne vrši zamjene prvih dviju kategorija pojavnica iz Tablice 1 (poglavlje 3.2), tj. preferira zamjenu nekih vrsta riječi (glagola, prijedloga, veznika...) nad nekim drugima (brojevi, imenice...).

Osim toga vidjet će se i da program u trenutnoj inačici nije u stanju pronaći nedostajuće riječi ili pravopisne znakove. To je zato što generiranje upita u kojima se zvjezdica umeće i između pojavnica, a ne samo umjesto njih nije implementirano.



Slika 24: Primjer izvršavanja

Potrebno je napomenuti i da se vrijeme dohvata rezultata za rečenicu duljine od pet do sedam pojavnica unatoč provedenim optimizacijama u slučaju *Google Scholara* kao izvora podataka, kreće u rasponu od deset do dvadeset sekundi. U slučaju kada koristimo *Project Gutenberg* to vrijeme je znatno manje.

Primjer izlaznih rezultata programa (s *Google Scholarom* kao izvorom podataka) vidimo na Slici 24. Vidimo ne samo rečenicu koja je odabrana kao najbolja, već i nekoliko sljedećih generiranih rečenica sortiranih prema ocjenama.

U tablicama ispod popisano je po nekoliko primjera s točnim i netočnim rezultatima za *Google Scholar*. Prikazane su samo rečenice koje je program rangirao kao najbolje, a ne i ostale kao na Slici 24.

Tablica 5: Točni primjeri

We leave in 10 o'clock. We leave **at** 10 o'clock.
All of these tools is driven by maximum entropy models. All of these tools **are** driven by maximum entropy models.
Provides information of the special constructs. Provides information **about** the special constructs.
This is the first step to solving this problem. This is the first step **in** solving this problem.

Tablica 6: Netočni primjeri

We invite you to using the service. We invite ***others*** to **using** the service. (Rečenica je trebala biti ispravljena u "We invite you to **use** the service.", ali to se nije dogodilo jer program "nije imao sreće". Naime, analizom dnevnika izvođenja utvrđeno je da se za upit "*to * the service"* glagol *use* nije našao na prve 4 stranice rezultata *Google Scholara.*)
We will go on concert next month. We will go on ***the*** next month. (Rečenica je trebala biti ispravljena u "We will go on **a** concert next month.", ali to se nije dogodilo jer program nema sposobnost ubacivanja nedostajućih riječi.)
The Internet is often often reffered to as the net. The Internet is ***not*** often reffered to as the net. (Rečenica je trebala biti ispravljena u "The Internet is often reffered to as the net.", ali to se nije dogodilo jer program nema sposobnost micanja suvišnih riječi.)

The sea is yellow.

The sea ***from*** yellow.

(Iako ovo nije pitanje stila već semantike, rečenica je očigledno trebala biti ispravljena u rečenicu poput "The sea is blue" ili "The sea is green". Razlog zašto promatramo ovaj primjer je to što se iz njega vidi da program ciljano nije mijenjao riječ *yellow* – zato što se radi o pridjevu, a pridjevi se prema Tablici 1 nalaze u četvrtoj (najnižoj) kategoriji riječi prikladnih za zamjenu.)

6. Zaključak

U ovom diplomskom radu bavili smo se jednim dijelom područja obrade prirodnog jezika (engl. *natural language processing*). Specifično, tema rada bila je izrada programa za provjeru stila teksta pisanog na engleskom jeziku. Takve provjere najčešće se vrše na temelju skupova pravila, velikih tekstnih korpusa te stohastičkih, probabilističkih i heurističkih metoda analize.

Motivaciju za izradu takvog programa nalazimo u činjenici da je engleski jedan od tri najraširenija svjetska jezika, a još više u činjenici da je postao neslužbeni standardni svjetski jezik. U situaciji u kojoj engleski govori jako veliki broj ljudi kojima taj jezik nije materinji, potrebno je pronaći načine za pisanje razumljivijih i pravilnijih tekstova, a korištenje programa za provjeru pravopisa, gramatike i stila jedan je od tih načina.

Cilj rada je, dakle, izgradnja programa za provjeru stila teksta pisanog na engleskom jeziku, u svrhu olakšavanja njegove upotrebe i učenja govornicima kojima nije materinji. Razvijeno je jedno jednostavno i donekle funkcionalno rješenje kojim se upravlja preko *web*-sučelja, što olakšava njegovu dostupnost zainteresiranim korisnicima. Koristi se metoda rastavljanja teksta na *n*-grame, traženja alternativnih *n*-grama u dostupnim izvorima podataka i računanja ispravljenih rečenica iz njih.

U programu se koristi jednostavan heuristički algoritam koja analizira dani tekst uz pomoću podataka tekstnih korpusa. Heuristika koja računa i ocjenjuje nove rečenice sastoji se od slaganja onih *n*-grama koje je moguće složiti zajedno u novu rečenicu te kombiniranja njihovih ocjena u ocjenu rečenice temeljem tri faktora koja se međusobno množe. Ti faktori nisu izračunati od strane računala već se do njih došlo isprobavanjem algoritma nad primjerima i podešavanjem od strane autora. Zato algoritam, iako donekle dobar, vrlo vjerojatno nije idealan. Algoritam bi se moglo poboljšati kad bi se u formule uveli neki faktori koje bi bilo moguće optimirati pomoću skupa primjera za treniranje.

Iako ne postiže potpunu funkcionalnost i ne daje idealne rezultate za svaku rečenicu, program je dobar temelj za neki evenutalni budući projekt, jer je otvoreno mnogo mogućnosti za poboljšanja, od kojih su neke dijelom implementirane, ali ne i iskorištene.

Ponajprije, tu je mogućnost provjere ispravnosti napisanih riječi (engl. *spell-checking*). Ova metoda implementirana je u izvor podataka *Projekta Gutenberg*, ali nije iskorištena.

Zatim, program nije u mogućnosti pronaći nedostajuće riječi ili znakove, ali to je moguće popraviti implementiranjem generiranja takve vrste upita.

Poboljšanje je moguće i u selekciji generiranih upita. Trenutno se upiti koji traže zamjene za neke vrste riječi strogo odbacuju, a svi ostali ostavljaju; to bi se moglo poboljšati uvođenjem algoritma koji ih ne bi tako strogo dijelio, već bi odluku donosio na temelju nekih vjerojatnosnih metoda.

Što se tiče konačnih rezultata, nad njima bi se također mogle vršiti dodatne provjere poput parsiranja dobivenih rečenica i korištenja rezultata parsera za povećavanje točnosti ocjene. Parser je dostupan u biblioteci koja se koristi u projektu. U biblioteci je dostupan i pronalazač vlastitih imena (engl. *name finder*) koji bi također mogao biti iskorišten (npr. ako želimo odbaciti one upite koji traže zamjenu za neko vlastito ime).

Kako su izvori podataka napisani potpuno odvojeno od ostatka programa, moguće je napraviti i nove i bolje izvore podataka. Npr. Googleova baza *n*-grama [16] zacijelo bi davala bolje rezultate od ove korištene u projektu, a i Microsoft razvija *web*-servis s *n*-gramima utemeljen na XML-u [17] (što je puno prikladnije od načina na koji pristupamo *Google Scholaru* u ovom projektu).

7. Literatura

1. Oxford Dictionaries. "style". S Interneta, pristupljeno 18. kolovoza 2010. (http://oxforddictionaries.com/)
2. Hrvatski nacionalni korpus. "Rječnik korpusne lingvistike". S Interneta, pristupljeno 22. kolovoza 2010. (http://www.hnk.ffzg.hr/bb/definicijekl.doc)
3. Wikipedia. "N-gram". S Interneta, pristupljeno 22. kolovoza 2010. (http://en.wikipedia.org/wiki/N-gram)
4. Naber. D. "A Rule-Based Style and Grammar Checker". Diplomski rad, Technische Fakultät, Universität Bielefeld, 2003.
5. After the Deadline (*web*-stranica istoimenog programa za provjeru pravopisa, gramatike i stila). S Interneta, pristupljeno 2. rujna 2010. (http://afterthedeadline.com/)
6. Raphael Mudge. "The Design of a Proofreading Software Service" (dokumentacija projekta *After the Deadline*). S Interneta, pristupljeno 3. rujna 2010. (http://aclweb.org/anthology-new/W/W10/W10-0404.pdf)
7. MSDN. "interface (C# Reference)". S Interneta, pristupljeno 26. kolovoza 2010. (http://msdn.microsoft.com/en-us/library/87d83y5b%28VS.80%29.aspx)
8. Google Scholar (*web*-stranica istoimene tražilice). S Interneta, pristupljeno 28. kolovoza 2010. (http://scholar.google.com/)
9. Project Gutenberg (*web*-stranica istoimene baze književnih djela). S Interneta, pristupljeno 28. kolovoza 2010. (http://www.gutenberg.org/)
10. Northedge R. "Statistical parsing of English sentences". S Interneta, pristupljeno 22. kolovoza 2010. (http://www.codeproject.com/KB/recipes/englishparsing.aspx)
11. Google Code (Googleova *web*-stranica posvećena razvijateljima aplikacija). S Interneta, pristupljeno 26. kolovoza 2010. (http://code.google.com/)
12. MSDN. "How to: Search Strings Using Regular Expressions (C# Programming Guide)". S Interneta, pristupljeno 26. kolovoza 2010. (http://msdn.microsoft.com/en-us/library/ms228595%28VS.80%29.aspx)
13. MSDN. "Regular Expression Language Elements". S Interneta, pristupljeno 26. kolovoza 2010. (http://msdn.microsoft.com/en-us/library/az24scfc%28v=VS.80%29.aspx)
14. Prashanth Ellina (*blog*). "N-gram data from Project Gutenberg". S Interneta, pristupljeno 30. srpnja 2010. (http://blog.prashanthellina.com/2008/05/04/n-gram-data-from-project-gutenberg/)
15. Prashanth Ellina (*blog*). "Project Gutenberg Ngram data: English only". S Interneta, pristupljeno 30. srpnja 2010. (http://blog.prashanthellina.com/2008/05/13/project-gutenberg-ngram-data-english-only/)
16. Official Google Research Blog. "All our N-gram are belong to you". S Interneta, pristupljeno 26. kolovoza 2010. (http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html)

17. Microsoft Web N-Gram Service (Public Beta). S Interneta, pristupljeno 31. kolovoza 2010. (http://web-ngram.research.microsoft.com/info/)

Dodatak A: Popis oznaka riječi biblioteke *OpenNLP*

``	lijevi navodnici
,	zarez
''	desni navodnici
.	oznaka kraja rečenice
$	znak za dolar
#	znak za funtu
-LRB-	otvorena zagrada
-RRB-	zatvorena zagrada
LS	oznaka člana liste
CD	glavni broj
FW	strana riječ
UH	usklik
NN	imenica, jednina ili nebrojiva
NNP	vlastita imenica, jednina
NNPS	vlastita imenica, množina
NNS	imenica, množina
SYM	simbol
RP	čestica
EX	egzistencijalni *there* (*there is...*, *is there…*)
POS	posvojni sufiks (*'s*)
VB	glagol, osnovni oblik
VBD	glagol, prošlo vrijeme
VBG	glagolska imenica/glagolski pridjev (*gerund/present participle*)
VBN	glagolski pridjev trpni (*past participle*)
VBZ	glagol, prezent trećeg lica jednine
VBP	glagol, prezent jednine, ne u trećem licu
DT	determinator (nalazi se ispred imenice, npr. broj, član, posvojna zamjenica...)
PDT	predeterminator (nalazi se ispred determinatora, npr. *both, all*)
PRP	osobna zamjenica
PRP$	posvojna zamjenica
JJ	pridjev
JJR	pridjev, komparativ
JJS	pridjev, superlativ
RB	prilog
RBR	prilog, komparativ
RBS	prilog, superlativ
CC	veznik nezavisno složene rečenice (*and, but, for, nor, or, so, yet*)
IN	prijedlog/veznik zavisno složene rečenice
MD	glagol, modalni (*may, can, must, should, need*)
WDT	determinator iz skupine *wh-riječi* [3]
WP	zamjenica iz skupine *wh-riječi*
WP$	posvojna zamjenica iz skupine *wh-riječi*
WRB	prilog iz skupine *wh-riječi*
TO	*to*

[3] Takozvane *wh-riječi* (engl. *wh-words*) u engleskom jeziku odnose se na skupinu upitnih i odnosnih riječi (ovisno o ulozi u pojedinoj uporabi) koje najčešće, ali ne nužno, započinju s *wh* (*what, why, where, which, who, how*).

Dodatak B: Primjer dnevnika izvođenja

Ovo je primjer uspješnog ispravljanja. Radi se o tekstu "We leave in 10 o'clock." Rečenica je pogrešna i potrebno je ispraviti *in* u *at*. Kod generiranja upita vidimo primjer odbacivanja upita (odbacuje se upit u kojem se zvjezdica nalazi umjesto broja 10). Generirani dnevnik je ručno uređen radi preglednijeg prikaza na stranici formata A4.

```
Razdvajam na recenice...
Pronadjeno 1 recenica.
Obradjujem recenicu (faza 1):
  We leave in 10 o'clock.
  Recenica ima 3 ngrama.
  Obradjujem ngram (faza 1):
    We leave in 10
    Generiram upite...
      * leave in 10
      We * in 10
      We leave * 10
      We leave in *
    Izbacujem upite koji imaju manje sanse biti korisni...
    Preostali upiti:
      * leave in 10
      We * in 10
      We leave * 10
  Obradjujem ngram (faza 1):
    leave in 10 o'clock
    Generiram upite...
      * in 10 o'clock
      leave * 10 o'clock
      leave in * o'clock
      leave in 10 *
    Izbacujem upite koji imaju manje sanse biti korisni...
    Preostali upiti:
      * in 10 o'clock
      leave * 10 o'clock
  Obradjujem ngram (faza 1):
    in 10 o'clock.
    Generiram upite...
      * 10 o'clock.
      in * o'clock.
      in 10 *.
      in 10 o'clock*
    Izbacujem upite koji imaju manje sanse biti korisni...
    Preostali upiti:
      * 10 o'clock.
Obradjujem recenicu (faza 2)
  We leave in 10 o'clock.
  Obradjujem ngram (faza 2):
    We leave in 10
    Racunam rezultate...
      za upit * leave in 10
        can leave in 10
        to leave in 10
        then leave in 10
        to leave in 10
        and leave in 10
        must leave in 10
        to leave in 10
        will leave in 10
        ? leave in 10
        might leave in 10
        'll leave in 10
        to leave in 10
        to leave in 10
        to leave in 10
        to leave in 10
        parental leave in 10
        Cistim dvostruke rezultate... preostali:
          ? leave in 10, koliko=7
          to leave in 10, koliko=1
```

```
        might leave in 10, koliko=1
        parental leave in 10, koliko=1
        'll leave in 10, koliko=1
        will leave in 10, koliko=1
        then leave in 10, koliko=1
        can leave in 10, koliko=1
        must leave in 10, koliko=1
        and leave in 10, koliko=1
    za upit We * in 10
      we sequenced in 10
      we investigated in 10
      we report in 10
      we proceeded in 10
      we noted in 10
      we confirmed in 10
      we observed in 10
      we fulled in 10
      we reported in 10
      we obtain in 10
      we found in 10
      we have in 10
      we be in 10
      we evaluated , in
      Cistim dvostruke rezultate... preostali:
        we obtain in 10, koliko=1
        we reported in 10, koliko=1
        we fulled in 10, koliko=1
        we found in 10, koliko=1
        we evaluated , in, koliko=1
        we be in 10, koliko=1
        we have in 10, koliko=1
        we report in 10, koliko=1
        we investigated in 10, koliko=1
        we sequenced in 10, koliko=1
        we proceeded in 10, koliko=1
        we observed in 10, koliko=1
        we confirmed in 10, koliko=1
        we noted in 10, koliko=1
      we obtain in 10, NumOcc = 33
      we reported in 10, NumOcc = 28
      we fulled in 10, NumOcc = 0
      we found in 10, NumOcc = 115
      we evaluated , in, NumOcc = 5150
      we be in 10, NumOcc = 32
      we have in 10, NumOcc = 104
      we report in 10, NumOcc = 0
      we investigated in 10, NumOcc = 18
      we sequenced in 10, NumOcc = 0
    za upit We leave * 10
      we leave out 10
      we leave romans 10
      we leave the 10
      we leave a 10
      we leave distal 10
      we leave those 10
      we leave a 10
      we leave at 10
      we leave at 10
      Cistim dvostruke rezultate... preostali:
        we leave out 10, koliko=2
        we leave a 10, koliko=2
        we leave the 10, koliko=1
        we leave romans 10, koliko=1
        we leave distal 10, koliko=1
        we leave at 10, koliko=1
        we leave those 10, koliko=1
Obradjujem ngram (faza 2):
  leave in 10 o'clock
  Racunam rezultate...
    za upit * in 10 o'clock
      obtained in 10 o'clock
      counts in 10 o'clock
      globe in 10 o'clock
      noted in 10 o'clock
      crack in 10 o'clock
      halo in 10 o'clock
      made in 10 o'clock
```

```
        } in 10 o'clock
        are in 10 o'clock
        figured in 10 o'clock
        noted in 10 o'clock
        strategy in 10 o'clock
        strategy in 10 o'clock
        layer in 10 o'clock
        hear in 10 o'clock
        hand in 10 o'clock
        canopy in 10 o'clock
        Cistim dvostruke rezultate... preostali:
          hear in 10 o'clock, koliko=2
          canopy in 10 o'clock, koliko=2
          are in 10 o'clock, koliko=1
          noted in 10 o'clock, koliko=1
          figured in 10 o'clock, koliko=1
          strategy in 10 o'clock, koliko=1
          hand in 10 o'clock, koliko=1
          layer in 10 o'clock, koliko=1
          halo in 10 o'clock, koliko=1
          globe in 10 o'clock, koliko=1
          counts in 10 o'clock, koliko=1
          obtained in 10 o'clock, koliko=1
          } in 10 o'clock, koliko=1
          made in 10 o'clock, koliko=1
          crack in 10 o'clock, koliko=1
      za upit leave * 10 o'clock
        leave at 10 o'clock
        leave around 10 o'clock
        leave until 10 o'clock
        leave at 10 o'clock
        leave at 10 o'clock
        leave at 10 o'clock
        leave at 10 o'clock
        leave at 10 o'clock
        leave at 10 o'clock
        leave at 10 o'clock
        leave at 10 o'clock
        Cistim dvostruke rezultate... preostali:
          leave at 10 o'clock, koliko=9
          leave until 10 o'clock, koliko=1
          leave around 10 o'clock, koliko=1
Trazim ispravke...

Konacni poredak
---------------
we      leave   at      10      o'clock.        UkOcj=50        BrPrekida=0   BrZamRj=0
we      have    in      10      o'clock.        UkOcj=10,07407  BrPrekida=1   BrZamRj=0
we      found   in      10      o'clock.        UkOcj=10,07407  BrPrekida=1   BrZamRj=0
will    leave   in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
to      leave   in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
might   leave   in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      report  in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      leave   those   10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
'll     leave   in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
must    leave   in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we sequenced    in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
and     leave   in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
parental leave  in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
then    leave   in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we  reported    in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we investigated in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
?       leave   in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      leave   the     10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      obtain  in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      be      in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      leave   romans  10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      leave   distal  10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      fulled  in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      leave   a       10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we      leave   out     10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
can     leave   in      10      o'clock.        UkOcj=9,876544  BrPrekida=1   BrZamRj=0
we   evaluated ,       in      o'clock.        UkOcj=2,194788  BrPrekida=1   BrZamRj=2
```

Sažetak

Tema ovog rada je izrada aplikacije za provjeru stila teksta pisanog na engleskom jeziku. Takve aplikacije se najčešće temelje na skupovima pravila i tekstnim korpusima. Razvijena je *web*-aplikacija koja se temelji na heurističkoj analizi teksta pomoću dva tekstna korpusa (utemeljeni na *Projektu Gutenberg* i tražilici akademskih tekstova *Google Scholar*). Koraci obrade teksta su analiza pomoću gotove biblioteke za obradu prirodnog jezika, rastavljanje rečenica na *n*-grame, generiranje novih *n*-grama, njihovo spajanje u nove rečenice, ocjenjivanje rečenica. U radu su prikazani i primjeri rečenica te obrazložena postignuta učinkovitost.

Ključne riječi: obrada prirodnog jezika, korpusna lingvistika, *n*-gram, provjera stila, tekstni korpus, analiza teksta.

Abstract

The subject of this diploma thesis is a software style checker for English language. Style checking is usually performed using text corpuses or collections of rules. The resulting software is a web application which uses two text corpuses (the data from *Project Gutenberg* and *Google Scholar*) and heuristic analysis. The steps of style checking are analyzing the text using a natural language processing library, extracting the *n*-grams from sentences, generating new alternative *n*-grams, building new sentences from those n-grams, grading the sentences. The examples of corrected sentences are shown and the efficiency accomplished is explained.

Keywords: natural language processing, corpus linguistics, *n*-gram, style checking, text corpus, text analysis.