SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1682

WEB SERVIS ZA AUTOMATSKO INDEKSIRANJE DOKUMENATA

Ivan Vidović

Zagreb, 2007.

Zahvaljujem se prof.dr.sc. Bojani Dalbelo-Bašić
koja mi je omogućila sudjelovanje na projektu.
Također zahvaljujem na usmjeravanju
i uvijek dobrodošloj potpori.

Posebno se zahvaljujem kolegi Frani Šariću
na savjetima i bez čije pomoći
bi ovaj projekt ostao puno
skromnijih razmjera.

Veliko hvala kolegici Ivani Zorović
na ugodnom timskom radu i uspješno
završenom zajedničkom projektu.

Zahvaljujem se obitelji na
strpljenju, poticanju i
razumijevanju.

Sadržaj

1. Uvod

U današnje vrijeme suočeni smo s velikim brojem elektroničkih dokumenata. Broj dokumenata konstantno raste, a time i količina sadržaja zapisana u njima što otežava pronalazak relavantnih informacija potrebnih za klasifikaciju dokumenata u odgovarajuće kategorije kao i pretragu dokumenata po ključnim riječima. Ručna obrada dokumenta je iz tog razloga naporna i zahtijeva puno vremena.

Ideja i prvi pokušaji poluautomatskog i automatskog indeksiranja dokumenata javili su se krajem 50-tih godina prošlog stoljeća. Tada je to bila progresivna teorija koja je sada postala neizbježna jer ručna indeksacija dokumenata i traženje značajki u tekstu nisu dovoljno brze tehnike obrade teksta da bi popratile rast broja elektroničkih dokumenata.

Poluautomatsko i automatsko indeksiranje teksta učinkovito rješava problem spore i neučikovite indeksacije dokumenata (ljudima kojima je zadatak da iz dana u dan indeksiraju dokumente i svrstavaju ih u pripadajuću kategoriju dokumenata).

Proces poluautomatskog ili automatskog indeksiranja dokumenata započinje tako da se sustavu preda elektronički zapis dokumenta koji je predstavljen nizom znakova. Sustav na temelju danog mu teksta provodi leksičku i semantičku analizu teksta te korisniku daje popis deskriptora koji opisuju ulazni dokument.

Jedan od takvih sustava je eCadis sustav (*eng. Computer Aided Document Indexing System*). Sustav je razvijen na fakultetu Elektrotehnike i Računarstava uz pomoć stručnjaka s Filozofskog fakulteta te Hrvatske informacijsko-dokumentacijske referalne agencije (HIDRA). Prvo rješenje je bio poluautomatski indeksator koji je na temelju analize teksta predlagao korisniku niz riječi koje su statistički najbolje reprezentirale tekst, dok je korisnik iz tog morao odrediti koje riječi su najbitnije za taj dokument i kojoj kategoriji on pripada. Nakon toga napravljen je i automatski indeksator eCadis koji se koristi u HIDRI pri indeksaciji službenih dokumenata Republike Hrvatske.

U daljnjim poglavljima biti će detaljno opisan sustav eCadis te razmatrane web tehnologije pogodne za razvoj web verzije ovog automatskog indeksatora. Nakon odabira najpogodnije tehnologije, biti će opisan razvoj web eCadis sustava i detaljni opis implementiranih funkcionalnosti.

Web verzija ovog automatskog indeksatora trebala bi služiti širem krugu korisnika da im ukaže mogućnosti eCadis sustava i njegovu vrijednost i svrhu.

2. Dubinska analiza teksta i indeksiranje dokumenata pojmovnikom EUROVOC

2.1. Uvod u područje dubinske analize podataka i teksta

Dubinska analiza podataka (eng. data mining) može se definirati kao netrivijalna ekstrakcija dotad nepoznatih potencijalno korisnih informacija iz velikih skupova podataka. Pomnom analizom i određenim algoritmima prikupljaju se potencijalno bitne informacije iz velikih skupova podataka. Uglavnom ga koriste organizacije koje se bave poslovnom inteligencijom i financijskom analizom, ali, naravno, najviše u znanstvene i istraživačke svrhe.

Dubinska anliza teksta (eng. text mining) je varijanta dubinske analize podataka u kojoj podatke čini nestrukturirani tekst.

Prvi sustavi za automatske indeksacije teksta se pojavljuju u 60-ima. Do prije nekoliko godina, vrlo malo ljudi je uopće čulo o pojmu dubinska analiza podataka, iako je on rezultat evolucije područja koja imaju dugu povijest. Pojam kao takav uveden je relativno nedavno, početkom 90-ih.

Automatsko indeksiranje može se definirati kao grupa riječi ili fraza koje je računalo odabralo iz teksta s namjerom da olakša pronalaženje i dohvat informacija te da pruži uvid o vrsti informacija koje se nalaze u tekstu. [29]

Kvaliteteta izabranih riječi dobivena računalnom obradom sigurno nije bolja od čovjekove procjene jer su ljudi sposobni u potpunosti shvatiti smisao teksta, shvatiti smisao riječi ovisno o kontekstu u kojem su izrečene za razliku od računala koje to ne može.

2.2. EUROVOC pojmovnik

Pojmovnik Eurovoc (EUROpean VOCabulary) je hijerarhijski organizirani, multidisciplinarni pojmovnik [2] za indeksaciju dokumenata korišten u mnogim Europskim institucijama. Obuhvaća sve djelatnosti Europske unije u obliku strukturiranoga i kontroliranoga popisa naziva za sve važne pojmove koji se javljaju u najrazličitijim dokumentima EU. [4]

Eurovoc je preveden na 21 službeni jezik EU-e, a dodatno su ga prevele i neke druge zemlje, među njima i Hrvatska. [5] Hijerarhijski je organiziran, tako da su pojmovi unutar njega raspoređeni u 21 područje i 127 podpodručja.

Pojmovi ili termini u Eurovocu nazivaju se deskriptori (eng. descriptors) jer se odabirom određenih deskriptora opisuje dokument. Ponekad se neki pojam u tekstu nalazi u različitim oblicima pa zato određeni deskriptori imaju uz sebe vezane asocijate – različite termine kojima se opisuje isti pojam. Asocijati nisu zasebni elementi pojmovnika, već su uvijek vezani uz točno određeni deskriptor.

Osim samog popisa deskriptora, EUROVOC sadrži i razne dodatne informacije i relacije među pojmovima. Prve dvije razine hijerarhije pojmovnika EUROVOC detaljno su prikazane u dodatku 1. Dodatno se u EUROVOC-u nalaze informacije o srodnim deskriptorima (eng. related term - RT), užim (eng. narrowed term - NT) i širim (eng. broader term - BT) pojmovima i slično.

Pojava sinonima u dokumentu može biti zbunjujuća za sam proces klasifikacije. Jedna od glavnih svrha tezaurusa je rješavanje problema sinonima. Jedan od sinonima je odabran kao deskriptor (eng. descriptor, pojam koji opisuje tekst). Samo deskriptor se može koristiti pri klasifikaciji teksta, a ostali sinonimi prestavljaju asocijate tom deskriptoru. [28]

3. Sustav za strojno potpomognuto indeksiranje dokumenata (Pei/eCADIS)

3.1. Razvoj Pei/eCADIS sustava

Primarni cilj eCADIS sustava je bio olakšati zadaću indeksiranja službenih dokumenata Republike Hrvatske deskriptorima iz Eurovoc pojmovnika. Izrada sustava je završena u travnju 2006. godine. eCADIS sustav obavlja automatsko indeksiranje teksta i omogućuje brže, efikasnije i uniformnije indeksiranje dokumenata u odnosu na ručno indeksiranje. [1]

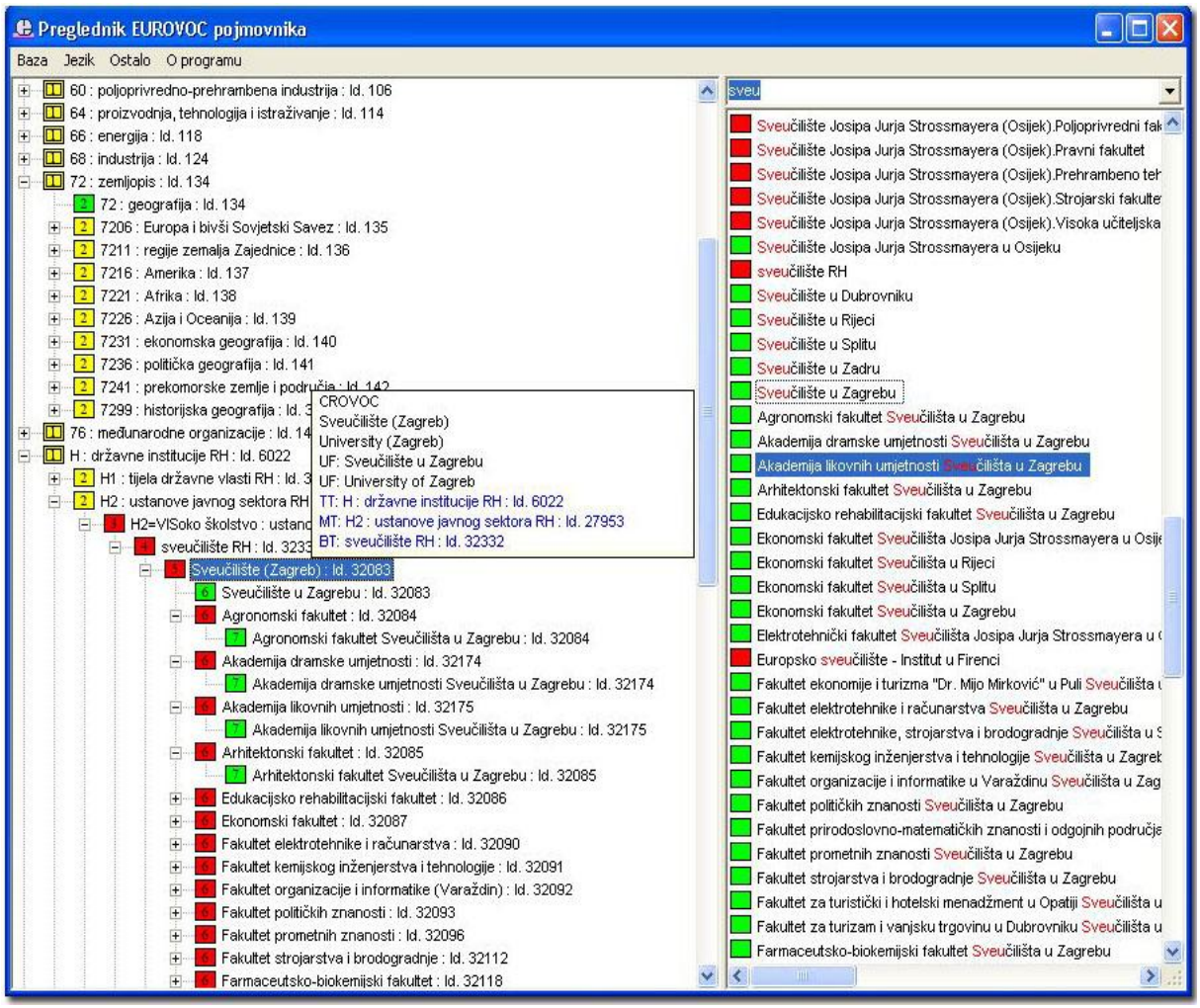
Prvi naziv je bio PEI – Poluautomatski Eurovoc Indeksator, da bi se naknadno, nakon uvođenja automatskog indeksiranja, preimenovao u eCadis sustav. Aplikacija je rađena u C++ programskom jeziku, no, u pozadini te aplikacije su nastajali algoritmi za obrađivanje teksta, indeksiranje. To je bio početak TMT – Text Mining Tools biblioteke. Iz cijelog projekta je proizašlo dosta seminara, članaka pa čak i jedna Rektorova nagrada.

Izdvajanjem algoritama iz PEI aplikacije, organizacijom koda i klasa te stvaranjem TMT biblioteke omogućen je lakši rad svim trenutnim i budućim sudionicima projekta. Ostvarena je modularnost, lakše nadograđivanje novih modula i naravno lakše korištenje postojećih modula. I na kraju, najbitniji pomak je mogućnost rada više timova/sudionika istovremeno i neovisno, na nadogradnji TMT biblioteke i različitih aplikacija koje koriste TMT biblioteku.

3.2. Struktura i funkcionalnost Pei/eCADIS sustava

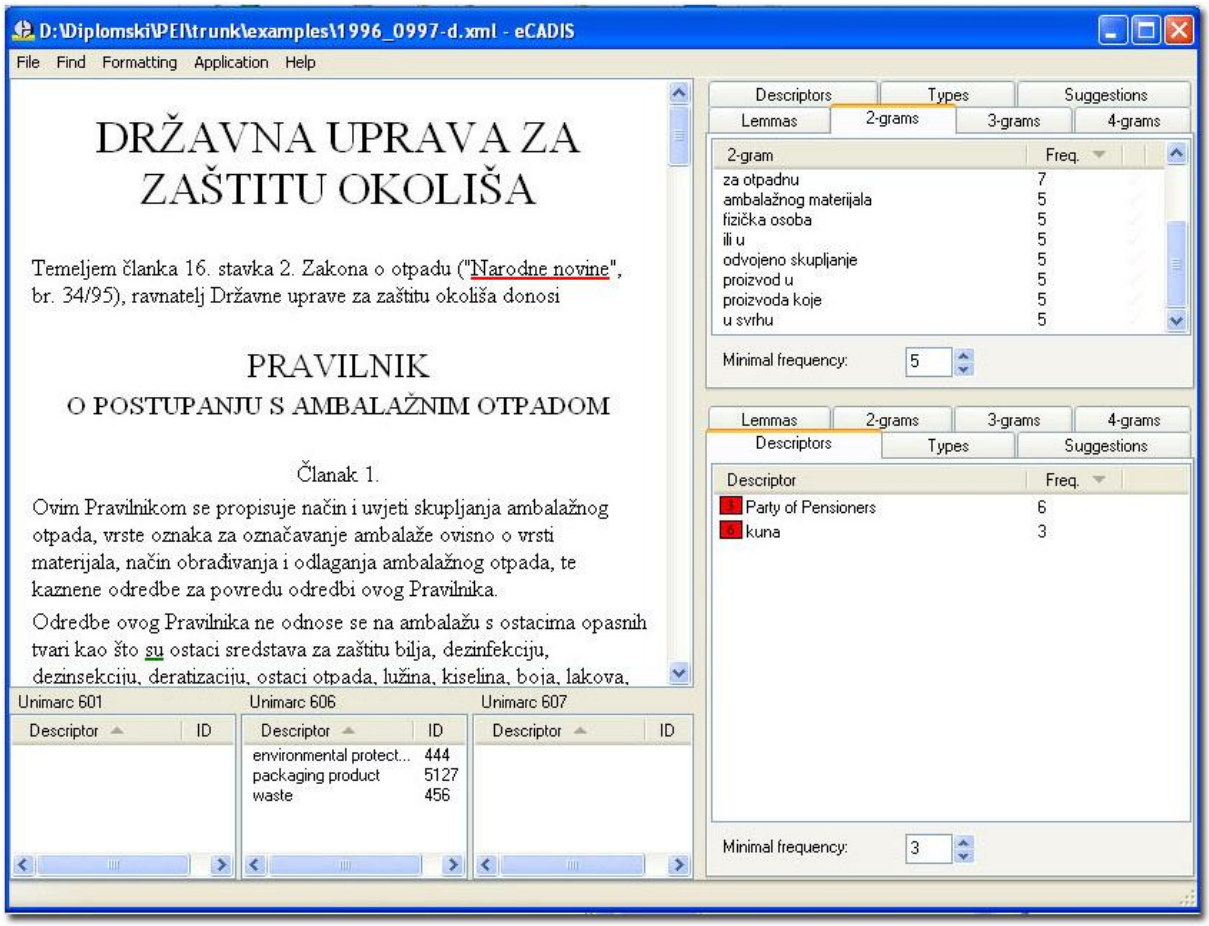
Ulaz u sustav je definiran datotekom zapisanom u XML formatu (eng. Extensible Markup Language) [3]. Zbog nemogućnosti podržavanja svih formata zapisa dokumenata odabaran je XML, standardizirani format zapisa dokumenata kojem je glavna svrha razmjena podataka u informacijskim sustavima i koji zbog svoje dobro definirane strukture zapisa podataka ima široku uporabu. [28]

PEI/eCADIS omogućava pregled EUROVOC pojmovnika. Hijerarhijski pogled na tezaurus u obliku stabla nalazi se s lijeve strane zasebnog prozora aplikacije, slika 3.1. S desne strane istog prozora moguća je pretraga po deskriptorima i asocijatima iz EUROVOC-a. Nađeni desktiptor, odnosno, asocijat može se zatim locirati unutar stabla. [6]



Slika 3.1 Korisničko sučelje PEI-a. Na lijevoj strani se prikazuje Eurovoc stablo, a na desnoj strani pretraživanje deskriptora i asocijata

Nakon učitavanja dokumenta kojeg treba indeksirati, aplikacija gradi internu stukturu podataka koja olakšava brzo i učinkovito pretraživanje dokumenta. Također, izvodi niz statističkih i leksičkih analiza, poput brojanja pojavljivanja različica u dokumentu, brojanja pojavljivanja lema, te pronalaženje N-grama, skupova od 2, 3 ili 4 riječi koje se često pojavljuju u skupini i kao takve nose dodatnu semantičku informaciju.



Slika 3.2 Korisničko sučelje PEI-a. Na lijevoj strani se prikazuje dokument predan na klasifikaciju, a na desnoj strani podaci dobiveni obradom algoritama implementiranih u sustav eCADIS. (Prikaz lema, n-grama, deskriptora i pojavnica).

Međusobna veza prozora za pregled EUROVOC pojmovnika i glavnog prozora za prikaz indeksiranog dokumenta, slika 3.2, omogućuje efikasnije indeksiranje samog dokumenta. Pronalaženjem odgovarajućeg deskriptora, korisnik unutar aplikacije stvara skupove 601, 602 i 607 koji opisuju indeksirani dokument. U trenutku kada su skupovi stvoreni po želji, omogućen je izvoz skupova u druge aplikacije. [6]

Vizualna rekonstrukcija dokumenta dobivena sustavom omogućuje korisniku praćenje rada sustava i rezultata dobivenih klasifikacijom. Na desnoj strani postoje dva preglednika. Oni nude korisniku uvid u dobivene rezultate. Rezultati koje korisnik može vidjeti su sljedeći:

- popis deskriptora i asocijata i pripadne frekvencije pojavljivanja unutar dokumenta,
- popis lema i pripadne frekvencije pojavljivanja unutar dokumenta,
- popis pojavnica i pripadne frekvencije pojavljivanja unutar dokumenta,

- popis n-grama (bigrami, trigrami i tetragrami) i pripadne frekvencije pojavljivanja unutar dokumenta. [28]

3.3. TMT biblioteka

Razvojem PEI aplikacije i izdvajanjem funkcionalnosti od sučelja je nastala TMT biblioteka/razvojno okruženje.

TMT razvojno okruženje je pisano u C++ programskom jeziku i omogućuje korištenje modernih „text mining“ metoda.

Funkcionalnosti TMT biblioteke su:

- Tokenizacija je proces rasčlanjivanja teksta na blokove (*eng*. tokens). Blokovi su sastavljeni od niza znakova kojima se kasnije pridjeljuje vrijednost procesom leksičke analize (proces rasčlanjivanja teksta na riječi, traženja početka i kraja riječi i uklanjanje formatiranja). Tim procesom su sve riječi u tekstu indentificirane i označene.

- Lematizacija je proces konvertiranja riječi iz svoje osnovne tekstualne forme u normaliziranu lingvističku formu – lemu.

- Eng. stemmer (od eng. stem - korijen) je program ili algoritam kojim se određuje korijen (najčešće leksički, tj. korijenski morfem) neke riječi. Na primjer, stemmer bi mogao naći da je korijen riječi osjećajnost – "osjeć", ili da je korijen riječi noga - "nog". Većina stemmera po određenim pravilima uklanja sufikse zadane riječi te često rezultat nije ispravan korijen, već samo aproksimacija korijena. Npr. stemmeri za engleski jezik javiti će "drie" umjesto "dry" za riječ "dries". [28]

Upute za korištenje te detaljniji opis funkcionalnosti koje omogućuje TMT biblioteka dostupan je na službenim stranicama textmining grupe. [6]

Iako se većina vremena i resursa/ljudi troši na nadogradnju TMT biblioteke i izradi novih algoritama i metoda indeksiranja dokumenata i raznih drugih nadogradnji, ipak je postojala potreba za nadogradnjom i PEI-a.

Nadogradnjom PEI aplikacije, odnosno izdvajanjem TMT biblioteke iz PEI-a i njenom nadogradnjom te uvođenjem automatkog indeksiranja, PEI je preimenovan u e-CADIS sustav kako se danas i zove. U ovo današnje vrijeme Interneta, dostupnosti Interneta te sve većim brzinama i mogućnostima, kao logičan sljedeći korak bio je napraviti WEB eCadis sustav.

4. Primjenjive web tehnologije za „web eCadis“ sustav

4.1. Uvod u problematiku

Naime, tema ovog rada nije proučavanje Interneta ni Weba, ali zahtjeva malu analizu razvoja Weba i problema koji postoje prilikom razvoja Web aplikacija, pogotovo prilikom implementacije rješenja u više različitih tehnologija, na više različitih platformi.

U početku razvoja Interneta nije se moglo ni slutiti do kakvih će revolucionarnih promjena doći, prvenstveno u komunikaciji. Početak razvoja je bio skromnih razmjera, no Internet je prošao dug put i prerastao u bitan medij koji se ravnopravno može nositi sa svojom puno starijom konkurencijom.

Za početak je potrebno napraviti razliku između Interneta i Weba. Ova dva pojma uglavnom se poistovjećuju, što je pogrešno. Internet kao pojam i kao tehnologija, medij, ili kako god ga želimo nazvati, je nastao prilično davno, sredinom prošlog stoljeća, no, Web (WWW) ima puno mlađu povijest i zapravo je servis koji funkcionira putem Interneta. Kraj 80-tih godina i početak 90-tih godina prošlog stoljeća se smatra kao početak WEB-a. Nastao je kao sustav za razmjenu dokumenata, da bi u iduća dva desetljeća evoluirao u nešto što njegovi autori nisu mogli ni zamisliti.

Iako je postojao i prije, u tih par godina (početak 90-tih) se dogodio ključan zaokret u razvoju WEB-a. Napravljen je prvi WEB server, prvi WEB browser te je prvi put napisan standard za HTML (koji je postojao još od 1980. godine). Nakon toga, razvoj WEB-a je krenuo strelovitom brzinom i kroz iduće desetljeće napravio pravu revoluciju u poslovnim i akademskim krugovima, a naročito društvenom životu običnih ljudi.

Kao sve nove i revolucionarne tehnologije, WEB je društvu pružio nešto više. Izbrisao je granice, kontinente, rasne razlike, postao je mjesto gdje svaki čovjek može izraziti sebe. Jednostavno, promijenio je način komunikacije. No, na pojavu WEB-a se može gledati i s druge strane. Budući je ljudima omogućena komunikacija putem Interneta, izgubljen je osjećaj prisnosti. Ljudi postaju otuđeniji, depresivniji, nemaju vremena jedni za druge zbog zaokupljenosti svojim kompjuterima, no, tema je to za neki drugi rad.

Razvoj telekomunikacijskih veza koji je Internet, a samim time i Web, učinio pristupačnijim te razvoj i nastanak novih Web tehnologija omogućili su da Web doživi popularnost kakvu danas ima. Razvoj WEB–a počeo je definiranjem HTTP-a (eng. Hypertext Transfer Protocol), komunikacijskog protokola koji se koristi za prijenos podataka putem Interneta i HTML-a (eng. Hypertext Markup Language), tekstualnog jezika kojim se daju upute browseru (program na klijent računalu koji “razumije” internet protokole, komunicira

preko njih sa server/klijent računalima razmjenjujući podatke) kako da prikaže određeni sadržaj.

Već na samom početku razvoja WEB-a nastali su problemi - „rat browsera“. Naime, komercijalizacijom Interneta počeli su nastajati različiti browseri, a kako nije bilo strogih definicija HTML-a, svaki od proizvođača počeo je uvoditi svoje nadogradnje. Iako su uvedeni standardi, HTML 1.0-4.0 i sl., nisu uspjeli prisiliti sve proizvođače da se drže svih pravila, tako da razvoj web aplikacija/stranica zadaje velike probleme/glavobolje onima koji ih razvijaju.

Prve WEB stranice sadržavale su obične tekstove te poneku sliku i pisane su u običnom i jednostavnom HTML-u. Stranice su bile opisnog karaktera, nisu omogućavale korisniku nikakvu interakciju, tek jednostavno gledanje i čitanje. Njihov statički karakter nije omogućavao ni autorima jednostavnu izmjenu. Nedugo nakon, uz napredak tehnologije, serveri su omogućili poziv komandno linijskih programa te prikazivanje njihovih rezultata unutar WEB stranica (eng. Common Gateway Interface - CGI). Sljedeći korak u razvoju su bili Appleti (mali programi koji bi se učitali i izvršavali na korisnikovom kompjuteru,ali bi komunicirali i dalje sa serverom) te paralelno s njima i skriptni jezici (PHP, CF, Perl, Python, Ruby ...) kao i serverske (JSP, ASP) te klijentske tehnologije (JavaScript, VBScript).

Pojam WEB 2.0 je nastao 2003. godine kao izraz za novu generaciju WEB stranica/aplikacija, WEB zajednica i WEB usluga. Izraz pretpostavlja novu verziju WEB-a, ali zapravo ne predstavlja novu verziju tehničke specifikacije WEB-a nego označava promjene koje su nastale u načinu razvoja WEB stanica/aplikacija i načinu kako krajnji korisnici koriste WEB. Označava početak WEB-a kao interaktivnog medija („platforma“).

Jedno od bitnijih značenja pojma WEB 2.0 jest da je svaka WEB stranica barem mala WEB aplikacija koja omogućuje jednostavnu klijentsku interakciju. Iako su sve tehnologije koje se koriste u razvoju WEB 2.0 aplikacija postojale i prije 2003. godine, pojam WEB 2.0 se ustalio u svijetu WEB-a.

Web aplikacije su postale sve sličnije „desktop“ aplikacijama po njihovom izgledu i funkcionalnostima, a napretkom u telekomunikacijskim infrastrukturama i brzini prijenosa podataka putem Interneta i po performansama, pa je takav pomak zaslužio i poseban naziv kojeg je i dobio, WEB 2.0.

Najveća zasluga pomaku pripada JavaScript jeziku, odnosno AJAX-u. AJAX je pojam koji je nastao slično kao i pojam WEB 2.0. Iza AJAX-a se ne krije neka nova tehnologija nego se samo ustalio dobar način korištenja starih tehnologija koji omogućava WEB aplikacijama da budu još sličnije „desktop“ aplikacijama, za koji se ustalila kratica AJAX (eng. „*Asynchronous JavaScript and XML*“).

4.2. Skriptni jezici

4.2.1. PHP

PHP je open-source skriptni jezik koji se najčešće koristi na serverima za dinamičko generiranje HTML koda. Sam naziv PHP je rekurzivni akronim koji znači "PHP: Hypertext Preprocessor".

PHP je skriptni jezik pomoću kojeg možete kreirati HTML stranicu na serveru prije nego što se ona, popunjena dinamičkim sadržajem, pošalje klijentu. Ovim načinom generiranja sadržaja klijent ne može vidjeti kod (skriptu) koji je generirao sadržaj koji gleda, već ima pristup čistom HTML kodu. Primjerice:

```
<html><head>
  <title>Primjer</title>
</head><body>
  <?php
   echo "Pozdrav svijete!";
  ?>
</body></html>
```

Ovaj primjer koda snimljen na serveru s podrškom za PHP i učitan u browseru ispisati će "Pozdrav svijete!", dakle, korisnik je dobio običan HTML kod.

PHP je jedna od najnaprednijih i najkorištenijih server-side skriptnih tehnologija koja je danas u upotrebi. On je po svojoj sintaksi poput mnogih drugih sličnih jezika, koristi čak i funkcije nekih drugih jezika kao što su C ili Perl. To znači da se jedna radnja može izvesti korištenjem više različitih funkcija.

Neke kritike upućivane PHP-u su generalne zamjerke svim skriptnim jezicima i jezicima s dinamičkim/slabim (eng. weak) tipovima podataka, no, ima i nekih koje su specifične za PHP:

- u PHP-u deklaracija varijabli nije obavezna, što može biti izvor raznih grešaka i sigurnosnih propusta,
- ugrađene funkcije nisu dosljedne po pitanju njihovog nazivlja ni po pitanju redosljeda argumenata među sličnim funkcijama (primjer nazivanja: strip_tags i html_entity_decode nasuprot stripslashes, htmlentities),
- funkcije nisu dosljedne u vraćanju rezultata - false, ali mogu vratiti i 0 ili "" ,

- broj ugrađenih funkcija je velik (preko 3000) što otežava razvoj, posebice jer dijele isti namespace,
- opcija "magic quotes" koja dodaje backslasheve u stringove može se uključiti ili isključiti u konfiguraciji pa kad se programeri oslanjaju na nju, mogu se javiti problemi na sistemima gdje je isključena,
- opcija "register_globals" automatski kreira varijable iz obrazaca što može postati sigurnosni rizik,
- konfigurabilnost PHP-a koja je istovremeno i prednost i nedostatak jer jedna skripta na jednom serveru može raditi, dok na dugom ne radi,
- stabilnost PHP-a mnogo ovisi o vanjskim bibliotekama funkcija.

4.2.2. Perl

Programski jezik čije je ime kratica od eng. "Practical Extraction and Report Language". Originalni autor Perla je Larry Wall, a prva inačica pojavila se 18. prosinca 1987. godine. Perl vuče svoje korijene iz drugih jezika kao što su primjerice C, sed, awk i Unix shell. Perl je danas ne samo programski jezik već i vrlo aktivna zajednica programera i korisnika. Odlikuje ga kvalitetan repozitorij gotovih programskih rješenja (CPAN - kratica od engl. "Comprehensive Perl Archive Network") što mu je ujedno i glavna prednost u odnosu na konkurentne jezike.

Perl je prema svojim karakteristikama objektno orijentirani interpretirani programski jezik opće namjene s naglaskom na funkcionalnost, proširivost te laganu krivulju učenja. Perl je bio jezik izbora za razvoj WWW aplikacija sredinom 90-ih godina. Od samih svojih početaka to je jezik Unix i Linux sistemskih administratora koji ga koriste u svakodnevnom radu prvenstveno za automatizaciju procesa. Danas postoji i čitav niz korisničkih komercijalnih aplikacija pisanih u Perlu.

Izvedbena okolina istovjetna je istoimenom prevoditelju koji ima dvostruku namjenu: prevođenje izvornog koda u međuoblik pogodan za neposredno izvođenje (eng. code compilation) te samo izvođenje koda (eng. code execution). Obje funkcije dostupne su u svakom trenutku, odnosno, Perl izvedbena okolina omogućava tvorenje novih Perl programa, odnosno, Perl funkcija u toku samog izvođenja. Time je Perl blizak i funkcionalnim programskim jezicima kao što su Smalltalk i Haskell.

Trenutna inačica Perla koja je danas u širokoj uporabi (Perl 5) ipak sadrži previše, već ponešto, zastarjelih programerskih tehnika, a zbog željene potpune kompatibilnosti sa starijim inačicama iz 80-ih godina pati i od izvjesnih nedostataka. Ovdje je potrebno napomenuti da je današnji stil programiranja u Perlu daleko od stila iz sredine 90-ih. Naglasak je prije svega na

korištenju gotovih, dobro testiranih, korisničkih biblioteka dostupnih preko već spomenute CPAN arhive. Premda je Perl na glasu kao "kriptičan" jezik "hackera", uz nešto profesionalne discipline te pridržavanja određenih pravila programiranja, moguće je pisati vrlo pregledne i uredne programe.

4.3. ASP.NET

Nešto malo prije popularizacije PHP-a, na Web scenu stupa Microsoft, izdavanjem Option Packa za Windows NT Server, u kojem unutar IIS-a 4.0 (eng. Internet Information Services) uvodi značajnu novost - prvu verziju svojeg Web programskog jezika nazvanog Active Server Pages ili skraćeno ASP. Dvije godine kasnije, izlaskom Windowsa 2000, izlazi i IIS 5.0 s ASP-om 3.0 što će, budućnost će pokazati, biti ujedno i posljednja inačica "klasičnog" ASP-a.

Već tada Microsoftov ASP po svojim mogućnostima uvelike zaostaje za PHP-om koji je još k tome pod Open Source licencom i vrti se na Linux/Unix-based operativnim sustavima (besplatna platforma). Upravo zbog toga Microsoft prestaje razvijati staru tehnologiju i okreće se izradi nove, koja će, vrijeme će pokazati, predstavljati revoluciju i svijetu Web-a, ali i informatičkom svijetu uopće.

Negdje početkom 2001. godine, Microsoft objavljuje osnovnu arhitekturu svoje nove tehnologije nazvanu .NET. Sredinom 2002. godine finaliziran je .NET Framework 1.0 i MS Visual Studio 2002. Od tog vremena potječe sveopća opčinjenost .NET-om koja traje i danas.

Osnovu .NET-a predstavlja svakako .NET Framework. Najjednostavnije rečeno, to je sustav koji nadograđuje mogućnosti samog operativnog sustava. Radi se o posebnoj infrastrukturi koja programerima nudi gotova rješenja i funkcionalnosti da bi ubrzala i pojednostavila razvoj aplikacija svih vrsta i oblika.

Najvažnija sastavnica .NET Frameworka zove se Common Language Runtime ili skraćeno CLR. CLR je softverski sustav u kojem se kôd izvršava. Kada korisnik pokrene aplikaciju pisanu za .NET Platformu, CLR ju izvršava kako bi joj osigurao stabilnost i funkcionalnost. Instrukcije u programu se u realnom vremenu prevode u izvorni strojni kôd koji razumije računalo. Za taj je posao zaslužan JIT-kompajler (eng. Just In Time). Upravo prevođenje u izvorni strojni kôd računala, omogućilo je .NET-u prelazak na druge operativne sustave kao što su Linux ili MacOS (putem pomoćnog third-party MONO sustava).

Kako kompajliranje zasigurno usporava izvršavanje aplikacija, ono će se izvršavati samo jednom, a njegov će se rezultat spremiti kako bi se kasnije mogao koristiti bez ponovnog kompajliranja.

Aplikacije za .NET platformu mogu se pisati u raznim programskim jezicima, gotovo svim poznatijim. CLR, međutim, ne poznaje niti jedan taj jezik - on dobiva naredbe isključivo u jeziku nazvanom Microsoft Intermediate Language (sraćeno MSIL), temeljen na pravilima koja se nazivaju Common Language Specifications (CLS). Stoga je jasno da mora postojati kompajler koji će programski jezik u kojem programer piše kôd prevesti u MSIL kako bi ga CLR razumio. Ovi kompajleri nazivaju se IL-kompajleri te su dostupni za velik broj programskih jezika. Microsoft iz izdao kompajlere za pet jezika: C#, J#, C++, Visual Basic i JScript, dok su se ostali proizvođači softvera potrudili oko brojnih drugih kao što su: Perl, Python, Cobol, Eiffel…

Kako se svi ovi jezici prvo pretvaraju u MSIL, sasvim je svejedno u kojemu od njih će se pisati aplikacije. Iz ovoga također proizlazi i druga velika mogućnost .NET-a - višejezično pisanje aplikacija. Tako sada više nije nužno da svi programeri koji rade na određenom projektu poznaju isti programski jezik, važno je samo da je podrška za njihov jezik dostupna u .NET-u, odnosno da postoji IL-kompajler za njihov jezik.

Mogućnosti koje CLR nudi su izuzetne, no, same po sebi nisu dovoljno uporabljive iz ljudskog aspekta. Upravo zbog toga u .NET Frameworku postoje setovi klasa koje omogućavaju brzo i jednostavno korištenje mogućnosti koje CLR nudi.

4.4. Java

Java je postala vrlo popularna zbog “write once - run anywhere” sintagme i prenosivog međukoda, koja omogućava da pružatelj usluge može promjeniti hardware sustava bez potrebe za ponovnim prevođenjem programa.

Izvođenje takvog koda je znatno brže od interpretiranih jezika, jer međukod je po strukturi blizak instrukcijama procesora, samim time i znatno jednostavniji od programa u skriptnim jezicima. Drugi čimbenik koji značajno utječe na bolje performanse izvršavanja međukoda je upotreba “just-in-time”prevodilaca. Ti prevodioci su sastavni dio Java izvršne okoline, iza vrijeme izvršavanja programa “u letu” prevode u izvršni kod (eng. “machine executable”) dijelove koda koji se često pozivaju ili uzrokuju usko grlo u izvođenju programa.

Činjenica da Java programski jezik zahtijeva da se programi izvršavaju unutar zaštićene okoline virtualnog stroja (eng. “virtual machine”) značajno poboljšava sigurnosne karakteristike Jave. Java izvršna okolina je vrlo robustna i tolerantna na pogreške. Ako program u Javi počini pogrešku, za vrijeme izvođenja baca se iznimka, ili točnije, objekt tipa Exception. Takav objekt se može koristiti u manipulaciji pogreškama kako bi cijeli sustav ostao funkcionalan.

Način na koji se to postiže je zatvaranje kritičnog dijela programskog koda unutar try bloka. Na kraju try bloka mora postojati i catch blok koji služi za hvatanje iznimke.

```
try{
/* kritični kod */
...
}
catch(Exception e) {
/* manipulacija pogreškom */
...
}
```

Također vrlo bitan razlog u odabiru Java programskog jezika leži u činjenici da je Java u potpunosti objektno orijentirani jezik. Dodatna sigurnost Jave je u činjenici da ne dopušta direktnu manipulaciju s memorijom, odnosno, postupak oslobađanja memorije je automatski, a sustav će uništiti objekt na toj memorijskoj lokaciji nakon što je siguran da taj objekt više nikome nije potreban.

U Javi nije moguće višestruko naslijeđivanje objekata kao što je to moguće u C++ jeziku. Velik broj autora navodi da su upravo direktno manipuliranje memorijom i nepravilnosti kod višestrukog naslijeđivanja primarni razlog pogrešaka u C++ jeziku. Gubitak fleksibilnosti koju pruža višestruko naslijeđivanje Java kompenzira upotrebom sučelja (engl. "interface"). Sučelja su posebna vrsta objekata koji opisuju koju funkcionalnost programer mora implementirati da bi dobio funkcionalnu cijelinu.

Prilikom stvaranja Jave autori su imali na umu pet glavnih ciljeva:

- Biti objektno orijentirani programski jezik
- Omogućiti izvođenje istog programa na različitim operacijskim sustavima
- Podržavati mrežnu komunikaciju
- Omogućiti sigurno izvršavanje koda sa udaljenih računala.
- Omogućiti lagano korištenje korištenjem dobrih strana OOP-a.

Jednostavnost jezika očituje se u tome da ga programeri mogu brzo naučiti. Autorima Jave bio je cilj učiniti je što sličnijom programskim jezicima koji već duže vrijeme postoje i većini programera su poznati (C, C++).

Objektna-orijentiranost znači da se kao programeri možemo usredotočiti na podatke (objekte) i metode (klase – skup metoda) pomoću kojih ćemo obaviti neki posao, a da ne moramo uvijek samo voditi brigu o tome kako ćemo napisati pojedine dijelove programa.

Distribuiranost – Java ima ugrađene sve osnovne funkcije za rukovanje mrežnim protokolima. Uz nju dobivamo biblioteke klasa koje komuniciraju s protokolima TCP/IP, FTP i HTTP. Zahvaljujući ovoj ugrađenoj mrežnoj podršci, programi pisani u Javi mogu bez

problema pristupati podacima koji su pohranjeni širom Interneta na različitim tipovima poslužitelja. Budući je zamišljena tako da podrži postojeće mrežne resurse, za Javu kažemo da je distribuirana.

Prenosivost i interpretiranje – Za razliku od kompiliranih jezika, kod kojih se iz izvornog koda odmah stvara izvršiva verzija programa namijenjena određenoj računalnoj platformi, prevođenje i izvršavanje Java aplikacija podijeljeno je u dvije faze. U prvoj, Java kompajler iz izvornog koda programa stvara bajt-kod. Bajt-kod bismo mogli nazvati "izvršnom" verzijom programa, iako se ne izvršava pod nadzorom operativnog sustava nekog računala niti je posebno kompajliran za njega. Umjesto toga, bajt-kod je međukod koji se pokreće pod nadzorom Javinog izvršnog sustava. Taj izvršni sustav interpretira bajt-kod i prevodi njegove naredbe u naredbe specificirane za računalo na kojem se izvršava aplikacija. Upravo zbog toga što se prevođenje programa iz bajt-koda u jezik platforme (strojni jezik) obavlja tijekom izvršavanja programa, Java se smatra interpretiranim jezikom.

Da bi se Java prenijela na neku novu platformu, potrebno je samo napisati izvršni sustav za nju. Izvršni sustav susrest ćemo pod imenom Java Virtual Machine (prividni stroj za Javu). Taj prividni stroj nije ništa drugo do okolina unutar koje se izvršavaju Java aplikacije. Možemo ga zamisliti kao "računalo u računalu" koje pokreće samo Java programe. Izvršni sustav može, ako je ispravno prenesen na novu platformu, izvršavati sve Java aplikacije, bez obzira na to tko ih je i na kojem računalu napisao. Izvršni sustav ugrađuje se u pretraživačke programe kako bi mogli pokretati programe pisane u Javi, a čak i neki operativni sustavi imaju ugrađen izvršni sustav i mogu pokrenuti Java aplikaciju poput bilo koje druge aplikacije pisane i kompajlirane za taj sustav. Java aplikacija ne mora se pritom niti najmanje mijenjati ili prilagođavati; ona se mrežom distribuira u svom bajt-kod obliku, a izvršni sustav je tijekom izvršavanja prevodi u specifične naredbe koje razumije računalo za koje je izvršni sustav napisan.

Robusnost – Budući je bila namijenjena pisanju aplikacija koje će se izvršavati u uređajima što se ubrajaju u potrošačku elektroniku, Java je zamišljena tako da bude vrlo sigurna i pouzdana. To, naravno, ne znači da se ne može napisati program bez pogrešaka. Budući precizno određuje veličine pojedinih tipova podataka i da zahtijeva precizno deklariranje metoda, Java omogućava kompilatoru da pronalazi programske pogreške. Javin memorijski sustav sam vodi brigu o oslobađanju i zauzimanju memorije, a zbog nepostojanja pokazivača nemoguće je da programer sam piše po nedozvoljenoj memorijskoj lokaciji i tako uništi neke podatke. Spomenuti sustav automatskog prikupljanja memorijskog smeća otklanja pojavljivanje memorijskih rupa.

Sigurnost – Kada se govori o Javi i sigurnosti izvršavanja Java programa nameću se sljedeća pitanja: Je li moguće u Javi napisati virus? Može li program pisan u Javi učiniti štetu na računalu na kojem se izvršava? Je li moguće u program unositi izmjene nakon što je kompiliran u bajt-kod? Odgovor je: Java je vrlo sigurna! Interpreter provjerava ispravnost

bajt-koda tijekom izvršavanja. Iako je kompilator napravio ispravan bajt-kod, interpreter ga još jednom provjerava kako bi zaista bio siguran da nitko s njime ništa nije radio ili ga mijenjao između kompajliranja i izvršavanja. Interpreter sprječava izvršavanje sumnjivog koda i koda koji bi sadržavao nedozvoljene radnje, poput pokušaja direktnog pristupa memoriji, nevažećih klasa, pogrešnih parametara koji se nalaze uz naredbe bajt-koda, pokušaja kršenja prava pristupa datotekama na disku ili nepravilnih pretvaranja među tipovima podataka.

Nakon što se uvjerio u ispravnost bajt-koda, interpreter određuje memorijski raspored klasa, osnovnih elemenata od kojih se gradi Java aplikacija. To je druga razina zaštite Java programa. Netko tko bi želio izmjeniti kompajlirani program u bajt-kodu, to ne može učiniti jer ne zna kako će klase biti raspoređene u memoriji.

Odličan učinak – Svaki programski jezik koji se zasniva na interpretiranju pati od sporosti, jer proces interpretiranja zahtijeva dosta procesorskog vremena za prevođenje samih naredbi čime se gubi na brzini izvršavanja. Unatoč sporosti, za većinu radnji koje obavljamo pomoću Jave – kao što su stvaranje korisničkih sučelja ili mrežne komunikacije – nije potrebna velika procesorska snaga. Aplikacija većinu svojeg vremena provodi besposlena, čekajući sljedeću naredbu korisnika ili pristizanje novih podataka s mreže.

Višenitnost – Java je programski jezik koji podržava višenitno izvršavanje programa. Višenitnost omogućava programeru da unutar svog programa pokrene više niti izvršavanja i time ubrza rad programa. Primjerice, jedna nit može u pozadini obavljati složena preračunavanja, dok druga već prikuplja nove podatke od korisnika. Višenitno izvršavanje programa donosi i veću učinkovitost procesora, jer jedna nit može koristiti njegove resurse u trenucima dok je druga nit besposlena.

Dinamičnost – Java je dinamički jezik, kojem se bez problema mogu dodavati novi objekti. Tako se i svi programi pisani u Javi ponašaju potpuno dinamički. To u praksi mnogo znači i donosi najviše prednosti kod pretraživačkih programa. Ako program pisan u Javi ili pretraživački program pokuša pristupiti nekom novom tipu podatka za koji ne zna kako ga obraditi, program može zamoliti poslužitelja da mu pošalje klasu koja rukuje tim tipom podataka. Program u Javi sam će se dinamički nadograditi dobivenom klasom i obaviti željeni posao.

5. Odabrano rješenje i korištene tehnologije

5.1. Cmake

Cmake je „open-source“ program koji omogućuje kreiranje projekata za različita razvojna okruženja i različite operacijske sustave (eng. cross-platform make system). Koristi se za kontrolu procesa kompajliranja softvera koristeći jednostavnu platformski-neovisne i kompajlerski-neovisne konfiguraciske datoteke. Cmake generira projektne i izvršene datoteke koje su prirodne za računalo na kojem se žele pokrenuti (eng. native makefiles) te se mogu koristiti za kompajlesku okolinu koja se odabere. Cmake je sofisticiran, omogućuje korištenje kompleksnih okolina koje zahtjevaju sistemske konfiguracije, generiranje predprocesorskih direktiva, generiranje koda i instanciranje predložaka. [10]

Glavne značajke koje omogućuje Cmake su:

- Podržava složena, velika razvojna okruženja. U praksi se dokazao na više velikih projekata.
- Generira „native“ izvršne datoteke (make datoteke za Unix, workspaces/projects za MS Visual C++). Iz tog razloga se alat može koristiti na bilo kojoj platformi i za bilo koji kompajler.
- Posjeduje jake komande za lociranje „include“ datoteke, biblioteke, izvršne datoteke. Uključuje vanjske Cmake datoteke koje omogućuju standardne funkcionalnosti, sučelja prema sustavima za testiranje, podržava rekurzivni obilazak direktorija sa nasljeđivanjem varijabli, može pokretati vanjske programe, podržava uvjetnu izgradnju, podržava regularne izraze, itd.
- Podržava izgradnju na licu mjesta (ako je potrebno izgraditi datoteke u istom direktoriju u kojem se nalazi Cmake datoteke) i izgradnju u drugim direktorijima. Više kompajlerskih verzija je moguće iz jednog izvora.
- Omogućuje lako dodavanje novih dodataka.
- Cmake je „open-source“.
- Cmake je napravljen tako da omogućuje povezivanje s grafičkim editorima.

```
PROJECT(FOO)
# make sure cmake addes the binary directory for the project to the include
INCLUDE_DIRECTORIES(${FOO_BINARY_DIR})
# add the executable that will do the generation
ADD_EXECUTABLE(my_generator my_generator.cxx)
GET_TARGET_PROPERTY(MY_GENERATOR_EXE my_generator LOCATION)
# add the custom command that will generate all three files
ADD_CUSTOM_COMMAND(
  OUTPUT ${FOO_BINARY_DIR}/output1.cpp ${FOO_BINARY_DIR}/output2.h ${FOO_BI
  COMMAND ${MY_GENERATOR_EXE} ${FOO_BINARY_DIR} ${FOO_SOURCE_DIR}/input.txt
  DEPENDS my_generator
  MAIN_DEPENDENCY ${FOO_SOURCE_DIR}/input.txt
  )
# now create an executable using the generated files
ADD_EXECUTABLE(generated
               ${FOO_BINARY_DIR}/output1.cpp
               ${FOO_BINARY_DIR}/output2.h
               ${FOO_BINARY_DIR}/output3.cpp)
```

Slika 5.1 Primjer Cmake konfiguracijske datoteke

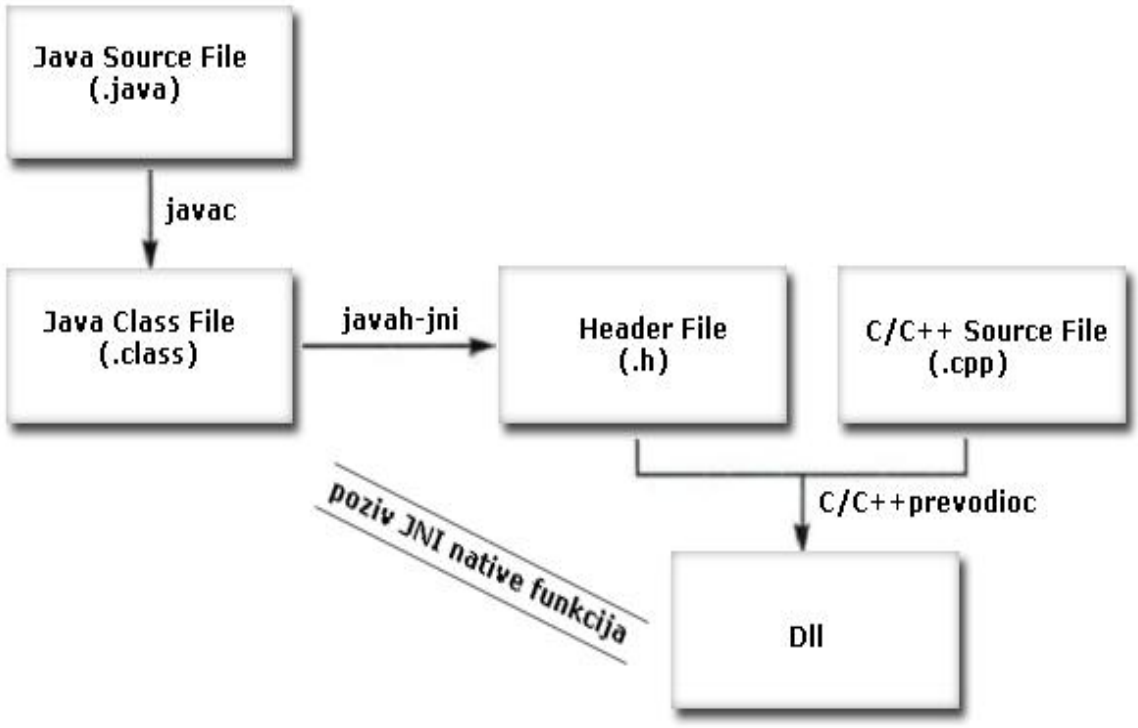
Na slici 5.1 je primjer Cmake konfiguracijske datoteke koja omogućuje generiranje više izlaznih datoteka gdje svaka od njih mora biti dio izvršnog programa. U ovom primjeru program mora čitati *input.txt* i generirati tri datoteke *output1.cpp*, *output2.cpp*, *output3.cpp*, i na kraju te tri datoteke moraju biti kompajlirane u izvršni program.

5.2. JNI

Java Native Interface (JNI) je jedini standardni mehanizam koji omogućuje interakciju programskog jezika Jave s jezicima čije se naredbe prevode izravno u strojni kod. [9]

Strojni kod (eng. native code) je kod koji se izvršava izravno u kontrolnoj procesnoj jedinici – CPU.

Postoji velik broj aplikacija napisanih u drugim programskim jezicima i namjenjenih izvršavanju na jednom računalu. Razvojem Interneta, javila se potreba korištenja tih aplikacija i na Internetu. Zbog kompleksnosti većine napisanih aplikacija i zbog velikih troškova njihove reimplementacije, koriste se mehanizmi koji mogu interaktirati između web tehnologija i programskih jezika koji se direktno prevode u strojni kod, od kojih je jedan JNI.



Slika 5.1 Prikaz korištenja JNI u procesu povezivanja Jave i C++ programskog jezika.

Poziv JNI native funkcija se odvija na Java sloju. JNI funkcije pozivaju DLL (eng. Dinamic Link Library) u kojem su zapisane prevedene metode korištene na C++ sloju. Na taj način je ostvarena veza između Java sloja i C++ sloja.

Proces prikazan na slici 5.2. možemo zapisati po koracima:

1. Kreiranje DLL na temelju metoda napisanih na C/C++ sloju
2. Generiranje JNI klasa
3. Implementacija JNI native metoda na Java sloju
4. Prevođenje programskog koda napisanog u Javi i njegovo izvršavanje.

Prednosti JNI arhitekture su:

- Binarna kompatibilnost – JNI programski kod je kompatibilan sa svakom JVM (eng. Java Virtual Mashine) istog operacijskog sustava. To znači da se Java aplikacija može prevesti na jednoj JVM, a izvoditi na bilo kojoj drugoj JVM koja podržava JNI.
- Ograničenje pristupa – JNI ograničava pristup JVM-u iz programskih jezika koji se direktno prevode u strojni kod.

- Napredni mehanizam regulacije zagušenja toka programa.

Nedostatci korištenja JNI:

- Portabilnost – korištenje JNI u Javi ograničava korištenje Java aplikacije na određeni operacijski sustav. Za korištenje Java aplikacije na Windows sustavu potreban je drugačiji format zapisa DLL-a.
- Kompleksnost – Aplikacija koja koristi JNI je kompleksna iz razloga što zahtjeva od programera dobro poznavanje Java programskog jezika i C/C++ jezika.
- Sigurnost – Java aplikacije koje koriste funkcije koje se direktno prevode u strojni kod nisu toliko sigurne kao čiste Java aplikacije.

5.3. SWIG

SWIG (eng. Simplified Wrapper and Interface Generator, Deavid Beazly) je razvojni alat (eng. program development tool) nastao istraživanjem u Los Alamos National Laboratory-u. Namjenjen je korisnicima koji žele koristiti skriptne jezike ili Javu zajedno sa svojim C/C++ programom, a ne žele se zamarati detaljima implementacije korištenih programskih jezika. [7]

SWIG automatski generira sav potreban programski kod za povezivanje C/C++ funkcija sa skriptnim jezicima ili Javom pri tome koristeći samo ulaznu datoteku koja sadrži deklaracije varijabli i funkcija (eng. header file). Na temelju toga, SWIG generira omotač oko (eng. wrapper code) oko C/C++ klasa koji se onda kasnije koristi iz skriptnog jezika ili Jave.

SWIG nudi mogućnost podešavanja generiranog omotača u svrhu bolje prilagodbe aplikaciji koja ga koristi. [8]

SWIG trenutno uspješno premošćuje C/C++ i sljedeće jezike:

- Allegro CL
- C#
- Chicken
- Guile
- Java
- Modula-3
- Mzscheme
- OCAML
- Perl
- PHP

- Python
- Ruby
- Tcl

Na ulaz SWIG prima ulaznu datoteku koja sadrži nekoliko specifičnih direktiva SWIG-a, a ostatak datoteke čine ANSI C funkcije i deklaracije varijabli. Ulazna datoteka ima ekstenziju -.i, npr. WebJava.i.

Ulazna datoteka koju zahtijeva SWIG je ista kao ulazna datoteka za C++ programski jezik, kao što je vidljivo na slici 5.3. Sa slike možemo isčitati deklaracije funkcija i varijabli napisane u C++, a dodatak koji je potreban SWIGU da napravi „*wrapper file*“ je - %module directiva. Ona postavlja ime inicijalne funkcije koju SWIG poziva. Programski kod unutar %{, %} bloka, kopira se direktno na izlaz omogućujući uključivanje izvornog C/C++ koda (header file).

```
%module swigjava
%include "std_string.i"
%include "std_vector.i"
%{
#include "wrappers/webjava/WebJava.h"
%}


class EurovocElemJava {
public:
          EurovocElemJava();
          ~EurovocElemJava();
          int getID();
          void setID(int ID);
          std::string getName();
          void setName(std::string NM);

private:
          int id;
          std::string name;
};

class ClassifierJava{
```

Slika 5.3 Prikaz WebJava .i datoteke korištene u projektu izgradnje Web eCadis sustava.

U datoteci WebJava.i se nalaze sve deklaracije varijabli i korištenih funkcija iz TMT biblioteke pri izradi Web eCadis-a. Korištene funkcije je bilo potrebno prilagoditi potrebama Java sloja.

Na slici 5.4. je prikazan izgled datoteke dobivene na izlazu SWIG-a. Datoteka ima ekstenziju -_wrap.cxx i mnogo je veća od datoteke na ulazu. Zbog toga ju je bilo nemoguće čitavu prikazati, ali iz ovog dijela koji je na slici vidi se rezultat dobiven pretvorbom.

```
#ifdef __cplusplus
extern "C" {
#endif

SWIGEXPORT jlong JNICALL
Java_eCadis_swigjavaJNI_new_1EurovocElemJava(JNIEnv *jenv, jclass jcls)
{
  jlong jresult = 0 ;
  EurovocElemJava *result = 0 ;

  (void)jenv;
  (void)jcls;
  result = (EurovocElemJava *)new EurovocElemJava();
  *(EurovocElemJava **)&jresult = result;
  return jresult;
}


SWIGEXPORT void JNICALL
Java_eCadis_swigjavaJNI_delete_1EurovocElemJava(JNIEnv *jenv, jclass jc
 {
  EurovocElemJava *arg1 = (EurovocElemJava *) 0 ;

  (void)jenv;
  (void)jcls;
  arg1 = *(EurovocElemJava **)&jarg1;
  delete arg1;

}
```

Slika 5.4 Prikaz datoteke WebJava_wrap.cxx koju je izgenerirao SWIG na temelju ulazne datoteke WebJava.i

SWIG pruža podršku gotovo svim funkcijama C++ jezika. Podržava sve tipove podataka, reference, klase, nasljeđivanje među klasama, template, statičke metode... U slučaju nestandardnog tipa podataka, SWIG pretpostavlja da je ta specifična struktura podatka definirana ranije u ulaznoj datoteci, u protivnom on ju neće moći prepoznati. Takav slučaj deklaracije specifičnog tipa podatka je vidljiv na slici 6.4 i ima oznaku %template.

```
class EurovocJava {
public:
        EurovocJava(std::string eurovocPath);
        ~EurovocJava(){};
        void setEurovocTree(std::string eurovocPath);
        std::vector<int> EurovocJava::getElementPath( int id);
        std::vector<EurovocElemJava *> getKids(int id, bool hr );
        std::vector<EurovocElemJava *> getRootKids( bool hr );
        std::string getDescription(int id, bool hr );
        bool hasKids(int id);
        TMT::Eurovoc *getEurovoc();
private:
        TMT::Eurovoc *eurovoc;
};


%template(vectorEuroElem) std::vector<EurovocElemJava *>;
%template(vectorInt) std::vector<int>;

%template(pairCW) std::vector < std::string >;
```

Slika 5.5 Prikaz korištenja specifičnog tipa podatka std::vector<EurovocElemJava *> u ulaznoj datoteci WebJava.i.

Samo neke od mogućnosti C++ programskog jezika nisu u potpunosti podržane u SWIG-u, virtualne metode i generiranje omotača za ugnježđene klase.

5.4. WEB tehnologije

Iako WEB tehnologije ne omogućuju toliko naprednu izradu sučelja kao standardne „desktop“ aplikacije, u zadnjih par godina su se pojavile metode koje tu razliku maksimalno umanjuju.

Nemoguće je opisati u par stranica JavaScript skriptni jezik, HTML, DOM i CSS. Može se naći tisuće stranica koje opisuju ove tehnologije, no, dobar početak za upoznavanje je knjiga *„HTML, XHTML, and CSS Bible – Bryan Pfaffenberger, Steven M. Schafer, Chuck White, and Bill Karow“* koja daje dobar presjek ovih tehnologija, dok je za bolje upoznavanje sa mogućnostima JavaScript jezika dobra knjiga *„JavaScript Bible – Danny Goodman with Michael Morrison“*. No, čitatelje je ipak potrebno bar malo upoznati s pojmovima.

5.4.1. HTML

„Hypertext Markup Language“ tekstualni format zapisa dokumenta, prepoznatljiv WEB preglednicima. WEB preglednici prepoznaju određene dijelove teksta i vizualno ih predočavaju korisnicima. Npr. element:

<input width="150px" type="submit" value="Index">

predstavlja input element tipa „submit“ što pregledniku govori da ga prikaže kao gumb, a vrijednost atributa *„value“* govori pregledniku koji tekst da napiše na gumb te vrijednost atributa *„width“* predstavlja širinu gumba. Tako se pomoću elemenata (*input*) i njihovih atributa (*type*, *value*, *width*) u HTML-u opiše izgled WEB stranice.

Osnovna struktura HTML stranice je sljedeća:

```
<html>
  <head></head>
  <body></body>
</html>
```

Unutar <*head*> elementa se upisuju metatagovi, naslov stranice, uključuju se JavaScripte, CSS – ovi i sl., dok u element <*body*> dolaze elementi koji predstavljaju sadržaj stranice.

5.4.2. CSS

„Cascading Style Sheets“ „stylesheet“ jezik koji se koristi prilikom opisivanja prezentacijskog dijela dokumenta pisanog u HTML ili sličnim formatima. Također prepoznatljiv WEB preglednicima i koristi se za opisivanje izgleda HTML elemenata, npr. definiranje boje, fonta, visine, širine i ostalih aspekata vizualnog izgleda. CSS je prvenstveno dizajniran da se može napraviti podjela između sadržaja i prezentacije. Do tada se izgled HTML elemenata opisivao pomoću atributa elemenata i nije se moglo ponovno iskorištavati jednom napisani izgled npr. gumba. To je naravno povećavalo veličinu dokumenata, tako da je uvođenjem CSS-a popravljena dostupnost sadržaja, omogućena veća fleksibilnost i kontrola prilikom specificiranja prezentacijskih karakteristika te je smanjena kompleksnost i ponavljanje u strukturi. Uključivanje CSS – a u HTML stranice se može vršiti na više načina, no, najbolje je izdvojiti CSS stilove u posebne dokumente te ih uključiti u HTML stranicu sljedećom linijom koda:

<link type="text/css" rel="stylesheet" href="css/design.css"/>

koja se upiše u <*head*> element HTML stranice. Unutar datoteke *„design.css“* se zatim pišu CSS stilovi, npr.

```
.example {
  font: bold 2.5em "Arial", Sans-Serif;
  margin: 0;
  letter-spacing: -1px;
}
```

koji se mogu dodavati HTML elementima preko atributa *„class“* :

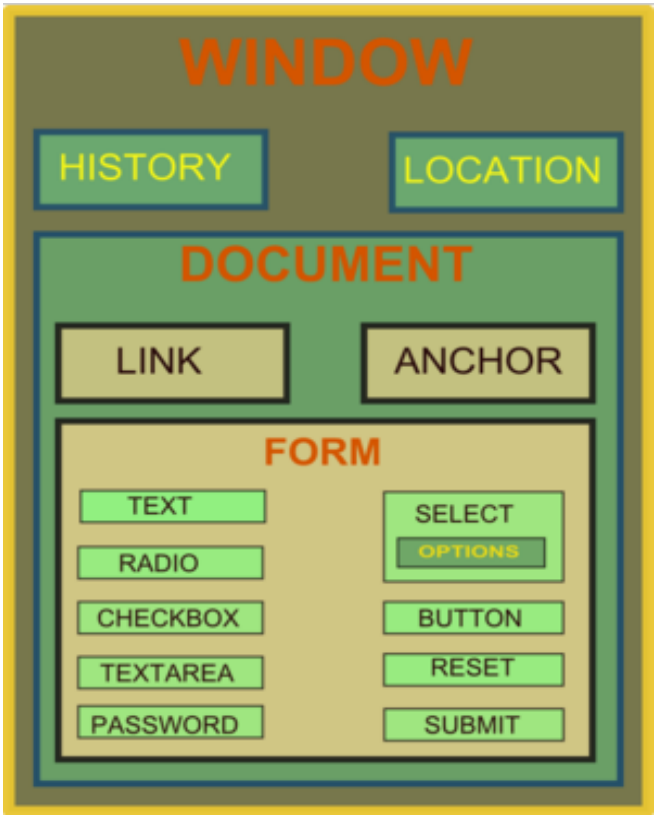
```
<div  class="example"></div>
```

Na ovaj način je omogućeno korištenje jednog stila na više mjesta, smanjena količina teksta koje korisnik mora učitati čime HTML stranice postaju preglednije.

5.4.3. DOM

„*Document Object Model*“ sadrži strukturu XML/HTML dokumenata. Naime, prilikom učitavanja WEB stranice na klijentskoj strani WEB preglednici pohranjuju XML/HTML dokument memoriju tako da izgrade hijerarhijsku strukturu, stablo od svih elemenata dokumenta. Na takav način se dobilo sučelje koje omogućuje pristup elementima XML/HTML dokumenta koje koriste korisnički skriptni jezici poput JavaScript i Vbscript jezika.

Na vrhu hijerarhije se nalazi objekt *window* koji predstavlja prozor preglednika u kojem je otvorena WEB stranica. On omogućuje manipulaciju sa navigacijom, povijesti, lokacijom trenutno otvorenog prozora. Najvažniji objekt je objekt *document* koji predstavlja sadržaj WEB stranice. Unutar njega se nalaze svi elementi WEB stranice, počevši od elementa <*head*> i <*body*> pa prema svim njihovim unutarnjim elementima, slika 5.6.



Slika 5.6 Hijerarhija objekata u DOM-u

5.4.4. JavaScript

Skriptni jezik koji se izvršava na klijentskoj strani. Postoji više načina za uključivanje JavaScript koda unutar XML/HTML stranica, no, u praksi se pokazalo da je najbolje izdvojiti JavaScript funkcije u zasebne datoteke i onda ih uključiti u stranicu, nešto poput „#include“ u C, C++ i sličnim programskim jezicima. Uključivanje se vrši dodavanjem sljedeće linije koda u <*head*> element HTML stranice, kao i prilikom dodavanja CSS datoteka:

```
<script language="JavaScript" type="text/JavaScript" src="js/test.js"></script>
```

JavaScript-a može pristupati DOM elementima i ima definirane funkcije za manipulaciju DOM elementima, npr. dohvaćanje elementa s id-jem 102:

```
var elem_102 = document.getElementById(„102“);
```

U JavaScripti postoje razne funkcije koje omogućuju manipulaciju dohvaćenim elementom. Može se kopirati, pristupati njegovoj djeci, roditeljima, mijenjati sadržaj, atributi, stilovi, itd. Npr. ako želimo promijeniti elementu CSS stil, odnosno njegov atribut koji mu određuje širinu to ćemo učiniti na sljedeći način:

```
elem_102.style.width = „200px“;
```

5.4.5. AJAX

„Asynchronous JavaScript and XML“ je tehnika izrade WEB aplikacija koja se koristi za kreiranje interaktivnih WEB aplikacija. WEB aplikacije se čine puno interaktivnije tako da se razmjenjuju male količine podataka sa serverom u pozadini, tako da se cijela WEB stranica ne mora ponovno učitavati svaki put kada korisnik napravi neku akciju/promjenu. Povećava interaktivnost WEB stranica/aplikacija, brzinu, funkcionalnost i iskoristivost.

AJAX je asinkron jer se AJAX učitavanje ne miješa sa normalnim učitavanjem stranica. Iz JavaScript programskog jezika se pozivaju AJAX funkcije, dok se podaci koji se se vraćaju uglavnom formatiraju u XML format i to pomoću XMLHttpRequest objekta, kojim je ostvaren AJAX poziv.

JavaScript funkcije za manipulaciju AJAX pozivima je dovoljno jednom napisati:

```
function loadAjax( url, callbackFunc, post_data, targetID){
    var xmlhttp = new XMLHttpRequest();
     if( post_data == null){
            xmlhttp.open( 'GET', url, true);
        }else{
            xmlhttp.open( 'POST', url, true);
            xmlhttp.setRequestHeader('Content-type', 'application/x-www-form-
     urlencoded');
            xmlhttp.setRequestHeader('Content-length', post_data.length);
            xmlhttp.setRequestHeader('Connection', 'close');
        }
        xmlhttp.setRequestHeader( 'If-Modified-Since', 'Sat, 1 Jan 2000 00:00:00
     GMT');
        xmlhttp.onreadystatechange = function(){
            if(xmlhttp.readyState == 4)
                callbackFunc( xmlhttp.responseText, targetID);
        }
        xmlhttp.send( post_data);
        return false;
}

function loadModuleContent( url, target){
  loadAjax( url, loadModuleContentCallback, null, target);
}

function loadModuleContentCallback( loadedAjax, target) {
    if(target != null)
       document.getElementById(target).innerHTML = loadedAjax;
}
```

Ove tri funkcije omogućavaju gotovo sve AJAX mogućnosti, bar one najvažnije. Prva funkcija ima 4 parametra:

1. URL za koji se želi napraviti AJAX poziv
2. callback funkcija koja se poziva nakon završenog učitavanja i kojoj se predaju dva parametra:
    a. xmlhttp.responseText - učitani text
    b. targetID – element u koji se želi spremiti učitani tekst
3. post_data – koji ako je *null* onda otvara GET request a inače POST
4. targetID – koji se proslijedi callback funkciji.

Može se raditi za svaki AJAX poziv posebna *callback* funkcija ili se može koristiti standardna, treća u gore navedenim funkcijama, tako da se učitavanje podataka sa *URL*–a u element sa id = *targetID* može napraviti jednostavnim pozivom druge funkcije:

loadModuleContent(URL, targetID);

AJAX je također i neovisan o platformi i iskoristiv na svim operacijskim sustavima, kompjuterskim arhitekturama i WEB preglednicima jer je temeljen na otvorenim standardima kao i JavaScript i XML.

5.4.6. JSP

„Java Server Pages“ je Java tehnologija koja omogućuje ubacivanje poziva Java funkcija, metoda, klasa, unutar HTML koda.

Unutar JSP stranice se mogu nalaziti HTML elementi i/ili JSP elementi. Standardni JSP elementi su *direktive, deklaracije, izrazi i skriptleti.*

Direktive se uključuju na vrh stranice i sadrže posebne instrukcije za WEB server. Najčešća direktiva je *page* direktiva, npr.

<%@ page import=“java.util.Date, java.io.“ %>*

Ova *page* direktiva kaže WEB serveru da uključi *java.util.Date* klasu i *java.io* paket i onda možemo koristiti te klase unutar drugih JSP elemenata.

Deklaracije služe za deklariranje metoda i varijabli. Npr.

```
<%! int godine = 23; %>
<%! public int getGodine(){
return godine;
}%>
```

Ono što se deklarira unutar ovih JSP elemenata je vidljivo ostatku JSP stranice.

Izraz je element koji kaže WEB serveru da prije nego što vrati rezultat korisniku *izraz* zamijeni rezultatom koji on vraća. Npr.

```
....
<p>Ja imam <%= getGodine() %> godina. </p>
....
```

Ova linija će se prikazati u pregledniku kao *„Ja imam 23 godine.“* jer će WEB server prije nego vrati rezultat korisniku izračunati sve *izraze* i uglaviti ih u rezultat.

Skriptlet element se može koristiti bilo gdje unutar JSP stranice. *Skriptlet* predstavlja Java kod okružen JSP elementom <% %>. Npr.

```
<%  for(int i=0; i<10;i++){
out.println(i);
} %>
```

6. Aplikacijsko rješenje

6.1. Opis sustava

Osnova aplikacije leži u već postojećoj TMT biblioteci o kojoj je bilo više riječi u prethodnim poglavljima. Biblioteka je napisana u C++ programskom jeziku. C++ programski jezik nije WEB tehnologija i stoga nije bilo moguće direktno korištenje TMT bibloteke. O tome je također bilo više riječi u prethodnim poglavljima. Rezultat toga je pretvaranje TMT biblioteke u .DLL (eng. Dinamic Link Library) za Windowse ili u .SO za Unix/Linux operacijske sustave. Kako je bilo potrebno omogućiti izvođenje projekta na Windows serverima i na Unix/Linux serverima web tehnologija u kojoj je realiziran projekt je Java/JSP.

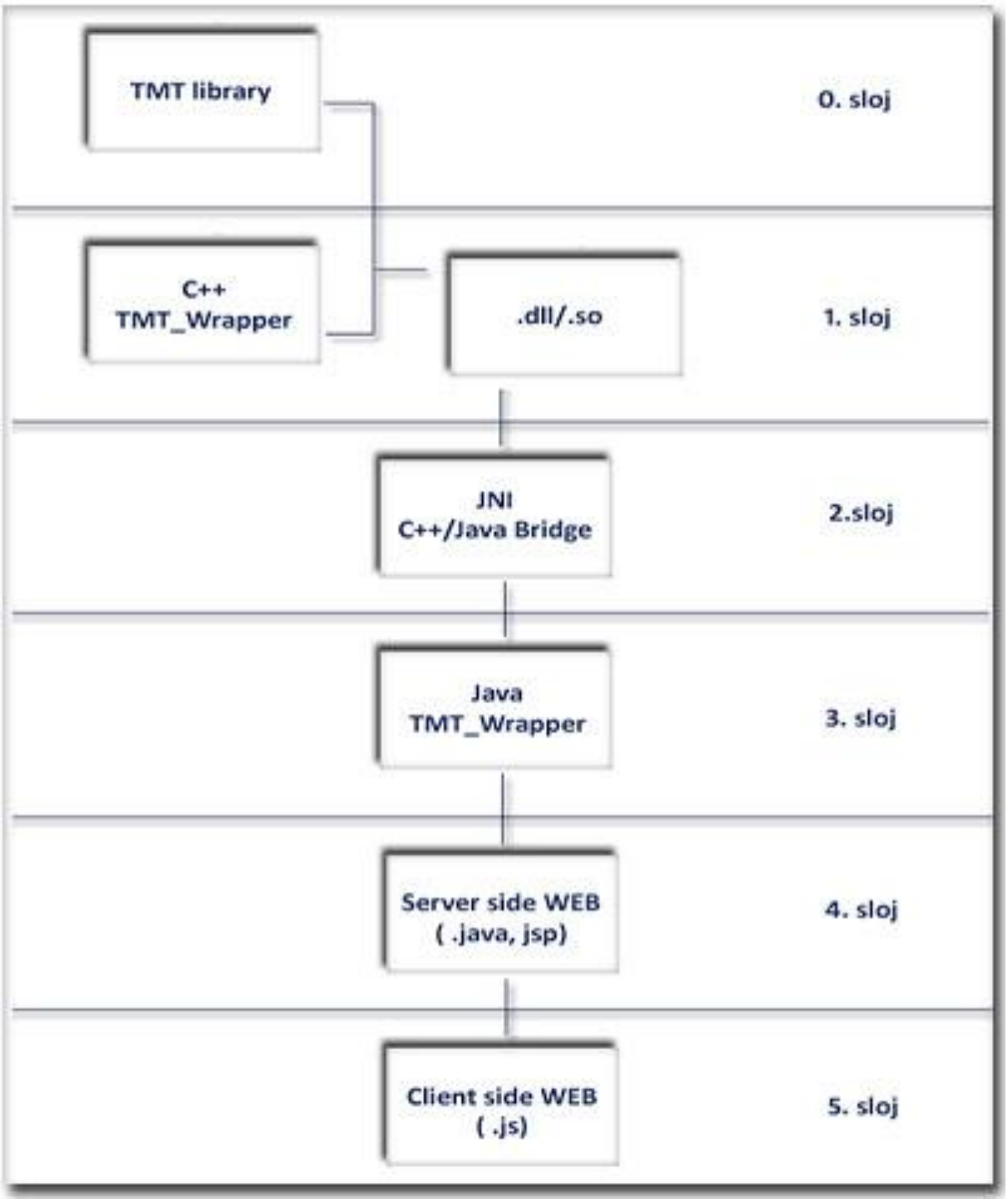
Iako postoji tehnologija za izvođenje ASP na Unix/Linux serverima (Mono), no, kako je tehnologija nova i u razvoju, sve funkcionalnosti .NET Frameworka nisu podržane, i također pomanjkanje iskustva je rezultiralo odabirom Java tehnologija za realizaciju projekta.

Kako su već prije opisane sve tehnologije koje su korištene u implementaciji može se naslutiti da realizacija nije išla glatko, no, na kraju bi se uvijek pronašlo rješenje i rezultat toga je sustav kojeg možemo opisati slikom 6.1.

Ispred TMT biblioteke je napravljeno nekoliko klasa - omotača koje služe kao sučelje prema Javi, naravno preko JNI sučelja. U tih nekoliko klasa su implementirane sve funkcionalnosti koje omogućuju rad WEB eCadis aplikacije. Naime, zbog premošćivanja dviju tehnologija (C++ Java) optimalno rješenje je bilo obaviti sve izračune i pozive TMT biblioteke u istoj tehnologiji (C++) i predavati Java sloju uređene i relativno jednostavne podatke i tipove podataka, nego omogućiti da Java poziva sve funkcije TMT biblioteke i onda izračune raditi na Java sloju. Na taj način smo dobili na jednostavnosti JNI međusloja, sveli smo ga na minimum i dobili na brzini izvođenja, jer se kompajlirani C++ kod (dll/so) brže izvodi od Java koda.

Ipak, na kraju nije bilo potrebno pisati JNI kod, korišten je SWIG, program za generiranje JNI koda. SWIG-u se preko konfiguracijskog/ih .i fileova daju naredbe koje mu govore za koje klase će izgenerirati Java klase i JNI kod za komunikaciju sa DLL-om.

Na taj način je stvorena veza C++ i Jave, no, također je povećana složenost razvoja aplikacije. Što to znači? Za bilo kakvu promjenu na C++ strani potrebno je raditi novi dll/so i naravno, svaki put promijeniti konfiguracijski .i file za SWIG i pozvati SWIG da ponovo izgenerira Java klase i JNI kod. Tek tada je moguće raditi promjene na Java strani i/ili sučelju.



Slika 6.1 Prikaz cjelokupnog sustava po komponentama, od WebPei biblioteke, do web aplikacije implementirane pomoću jsp i JavaScript/Ajax tehnologije

Dakle, sve funkcionalnosti koje se žele napraviti na WEB eCadis sustavu korištenjem TMT biblioteke, potrebno/poželjno je napraviti na C++ sloju. Prije povezivanja s Java slojem potrebno je napraviti mali *main.cpp* u kojem se provjere da li su rezultati koje vraća novonapravljena funkcionalnost (metoda postojeće klase ili metoda novonapravljene klase)

zadovoljavajući, tj. da li radi ono što bi trebala raditi. Tek nakon što se dobiju dobri rezultati na C++ sloju, radi se povezivanje s Javom.

Nakon povezivanja (korištenjem SWIG-a) potrebno je i na Java strani napraviti identičan *main.java* program onom na C++ sloju (*main.cpp*) te testirati je li povezivanje prošlo bez problema, odnosno jesu li rezultati koje vraćaju novoimplementirane funkcionalnosti na Java strani jednaki onima koje iste te funkcionalnosti vraćaju na C++ sloju.

Nakon svih testiranja može se krenuti u implementaciju novih funkcionalnosti na WEB sloju. U tom trenutku je sigurno da nove metode/klase vraćaju/sadrže prave vrijednosti koje trebamo prikazati korisniku ili koristiti prilikom razvoja aplikacije. Razvoj ove aplikacije i njena nadogradnja se tako može razložiti na tri neovisna razvojna okruženja, C++ – Java – WEB, dok se briga oko premošćivanja C++ i Java sloja može prepustiti SWIG-u, ali uz mali oprez prilikom pisanja C++ omotača oko TMT biblioteke zbog razlika u tipovima varijabli, podataka, i sl.

Najsigurniji, ali i nešto sporiji način, pokazao se prijenos svih potrebnih podataka na Java sloj kao jednostavne ugrađene tipove ili kao manje ugrađene objekte tipa String ili Vektor ili bilo kakve ručno napravljene objekte. I naravno naprednije C++ mogućnosti „ostaviti“ na C++ sloju pokazivače, pokazivače na funkcije i sl., osim ako nije apsolutno nužno jer ih SWIG može prebaciti, ali ih je ipak sigurnije ne koristiti.

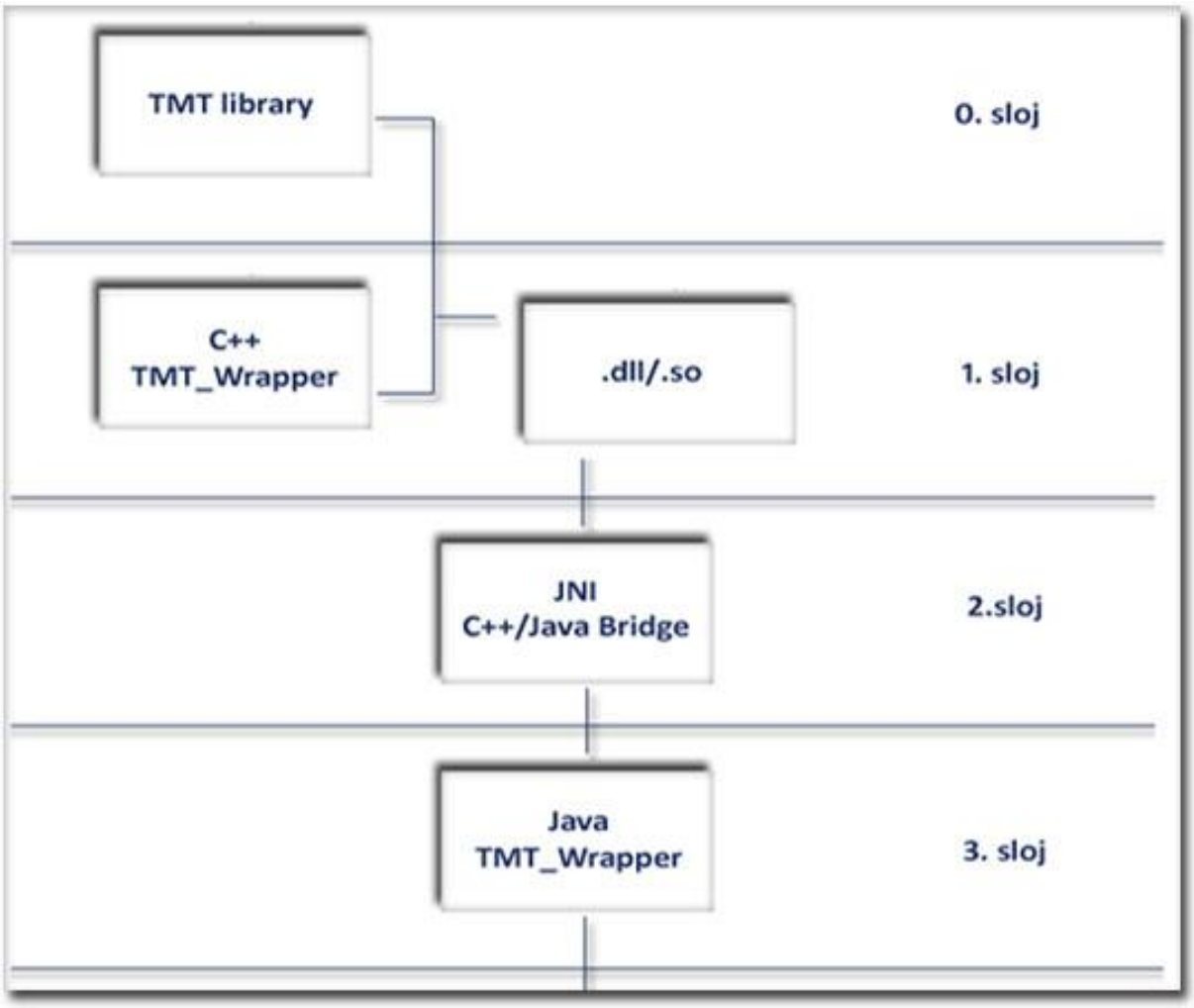
Ovo je bio samo mali pregled stvaranja aplikacije i pregled postupka dodavanja novih funkcionalnosti, a detaljniji opis izrade i postupak dodavanja novih funkcionalnosti sa primjerima slijedi dalje u ovom poglavlju.

6.2. Sloj Model – Java – JNI – DLL/SO

Većina današnjih WEB aplikacija se razvija provjerenom WEB arhitekturom MVC – Model – View – Controler. To ne znači da aplikacija nužno ima samo tri sloja. Svaki od tih slojeva može također biti razložen na više slojeva.

MVC arhitektura predstavlja dobar model prilikom razvoja WEB aplikacija jer razdvaja podatkovni sloj – Model, od prezentacijskog sloja – View, što je standardno i prilikom razvoja desktop aplikacija, ali uvodi i sloj koji kontrolira tok – Controler koji je potreban prilikom razvoja malo složenijih WEB aplikacija.

Model uglavnom predstavlja sloj koji se bavi podacima, pristupom bazi, dohvatom podataka i sl. Iako u ovoj aplikaciji ne postoji baza podataka u klasičnom smislu, podaci i dohvat podataka, odnosno Model čini struktura predstavljena slikom 6.2.



Slika 6.2 Struktura Modela u WEB eCadis sustavu

Kako se to može prozvati modelom? Može se napraviti odlična usporedba prilikom razvoja standardnih aplikacija/WEB aplikacija koje koriste baze podataka i razvoja WEB eCadis sustava/aplikacije.

Bazu u ovom slučaju možemo zamisliti kao C++ sloj, odnosno DLL/SO, JNI sloj se može zamisliti kao JDBC (eng. Java DataBase Connector), i Java_wrapper je Java omotač prema pozivima prema bazi.

Na taj način se sveo razvoj WEB eCadis sustava/aplikacije na razvoj standardne WEB aplikacije, gdje Model čini struktura sa slike 6.2.

Model WEB eCadis sustava ima tri sloja:

- Sloj C++, (TMT, wrapper, dll/so),
- Sloj JNI,

- Sloj Java (java_wrapper, SWIG-om izgenerirane Java klase).

6.2.1. Sloj c++, (TMT, wrapper, dll/so)

Početak rada na WEB eCadis sustavu/aplikaciji se svodio na razmatranje o primjenjivim WEB tehnologijama. Bilo je više opcija, no, na kraju je odluka pala na Javu i Java WEB tehnologije. Kako premošćavanje C++ native koda i Javinog međukoda nije jednostavno, čak ni uz pomoć SWIG-a, morao se pojednostaviti pristup TMT biblioteci i njenim funkcionalnostima. Pojednostavljenje je napravljeno na način da se pravio omotač oko TMT biblioteke. Naime, znalo se koje funkcionalnosti se očekuju od WEB eCadis-a. Nije bilo moguće koristiti TMT biblioteku na Java strani iz više razloga, no, glavna dva su:

1. Nejednakost jezika (pokazivači, pokazivači na funkcije ...) - razlog je u tome što smo se odlučili za korištenje SWIG programa za generiranje veze C++ i Jave i stoga smo se morali držati u okvirima njegove funkcionalnosti koja je ograničena u smislu da ne podržava apsolutno sve funkcionalnosti koje omogućuje C++, a dio njih je primjenjivan i u TMT biblioteci. Druga opcija je bila da sami pišemo JNI kod, a to nije bilo prihvatljivo zbog složenosti postupka.

2. Sporije izvršavanje - razlog nastaje zbog drastično različitog broja poziva C++ sloja preko JNI sloja od strane Java sloja. Da se kojim slučajem i uspjelo preći preko prvog razloga, dobili bi cijelu TMT biblioteku u obliku Java klasa. Svaku funkcionalnost bi trebalo pisati u Javi što i nije neki problem. No, onda bi sve funkcionalnosti koristile pozive C++ sloja preko JNI sloja.

Biranje načina se svodilo na odabir između:

1. Učenje JNI „jezika“, pisanja JNI koda za cijeli TMT i prihvaćanja sporijeg rješenja da bi mogli cijelu aplikaciju pisati u Javi.

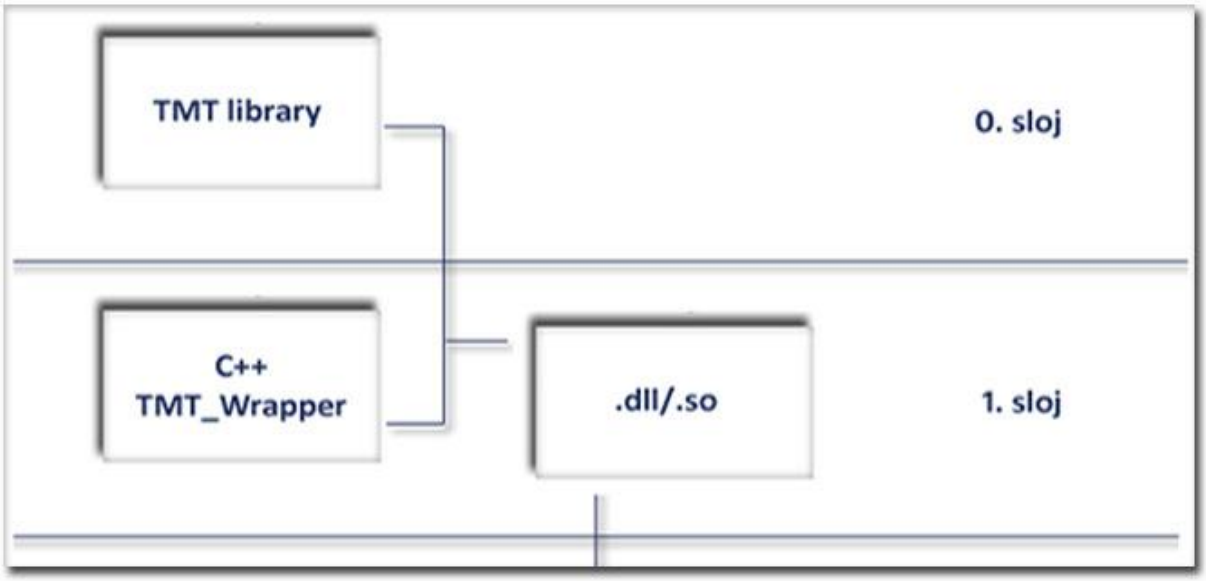
2. Pokušati do krajnjih granica koliko je dobar SWIG i nadati se da će sve proći bez potrebe za naknadnim ispravljanjem JNI koda i prihvaćanja sporijeg rješenja da bi mogli cijelu aplikaciju pisati u Javi.

3. Pokušati do krajnjih granica koliko je dobar SWIG i ispravljati JNI kod i prihvaćanja sporijeg rješenja da bi mogli cijelu aplikaciju pisati u Javi.

4. Za svaku funkcionalnost koja je potrebna u WEB eCadisu napisati metodu u C++ i povratnu vrijednost svesti na što je moguće jednostavniji oblik koji SWIG sigurno može prevesti.

Naravno, odabrali smo 4 opciju koja je bila po svim pitanjima najprihvatljivija, osim po jednom. Nije se moglo odraditi JNI most i otada nadalje raditi u Javi. Ovako se konstantno mora, za bilo kakvu novu funkcionalnost koju se želi dodati u WEB eCadis, prvo napraviti metodu na C++ sloju, dodati je u .i konfiguracijsku datoteku za SWIG, pokrenuti SWIG i onda je dobiti na Java sloju.

Iako povećava složenost razvoja s jedne strane, ipak je bilo prihvatljivije od ostalih opcija.



Slika 6.3 Sloj C++, (TMT, wrapper, dll/so)

TMT_Wrapper je C++ file u kojem se trenutno nalaze klase:

- ClassifierJava
- EurovocJava
- EurovocElemJava
- PairOfStrings

Klasa ClassifierJava implementira funkcionalnosti koje su vezane za indeksiranje dokumenata, dohvat deskriptora, lema ... Sastoji se od sljedećih metoda i varijabli:

```
public:
  ClassifierJava();
 // konstruktor
```

```
  ~ClassifierJava(){};
// destruktor
  void setLematizator(std::string lemmatizationPath, std::string stopWordsPath);
//postavlja lematizator
  void setClassifierData( std::string classifierPath, std::string featureBuilderPath);
//postavlja podatke bitne za indeksiranje
  void classify(bool language, std::string text, EurovocJava *eur);
//indeksira dani tekst
```

Kao što je vidljivo, ove funkcionalne metode ne vraćaju nikakve vrijednosti. Ako se malo bolje pogleda u kod ovih funkcija, u TMT biblioteci postoje iste ili slične klase/funkcije, ali one zapravo vraćaju vrijednosti uglavnom preko reference i na takav način radi većina metoda iz TMT biblioteke što uvodi komplikacije prilikom izlaganja C++ koda Javi. TMT_wrapper upravo služi zaobilaženju takvog načina vraćanja podataka iz funkcija. Ako se malo obrati pažnja na ulazne parametre, svi su jednostavni tipovi podataka, osim *EurovocJava* klase. Unutar ovih metoda je implementiran poziv svih potrebnih klasa i njihovih metoda iz TMT biblioteke koje su potrebne za postavljanje svih parametara, indeksiranje teksta, dohvat deskriptora, lema i sl., i spremljeni su rezultati u privatne varijable:

```
private:
  TMT::Lemmatization *lemmatization;
  ClassifierData *classifierData;
  XMLDocumentSpanifier *spanifier;
  std::vector<PairOfStrings *> suggestedDescriptors;
  std::vector<PairOfStrings *> descriptors;
```

Sve dohvaćene i izračunate vrijednosti se spremaju u gore navedene privatne varijable. Način na koji ova klasa služi kao most između Jave i C++ je takav da prve tri metode *setLemmatizator, setClassifierData, classify* služe kao ulazne metode u kojima se pozivaju sve potrebne klase i metode iz TMT biblioteke, i spremaju se rezultati. Dolje navedene metode služe za dohvat tih podataka u relativno jednostavnom obliku, kao parovi (SWIG nije prepoznavao pair ugrađeni tip, pa smo morali napraviti pomoćnu klasu, *PairOfStrings*, za prijenos para na Java sloj) ili kao običan String.

```
public:
  std::vector <PairOfStrings *> getSuggestedDescriptors();
  std::vector <PairOfStrings *> getIndexDescriptors();
  std::string getTextXML();
```

Npr. pomoću iduće tri linije koda se pomoću TMT biblioteke indeksira tekst:

```
  vector<int> categoryIDs;
  vector<float> weights;
  this->classifierData->classify(text, &categoryIDs, &weights);
```

Takav način prijenosa vrijednosti ne postoji u Javi i uvelo bi dodatnu komplikaciju u JNI kod, a upitno je da li bi SWIG dobro izgenerirao JNI kod za ovaj slučaj, a i na kraju svega potrebno je urediti podatke pa je bolje i taj posao odraditi na C++ sloju.

Zato se te tri linije koda nalaze u metodi *classify* klase *ClassifierJava* i Java sloj ih niti vidi, niti ima doticaja s njima.

Preko JNI mosta je Java sloju izloženo, na ovom primjeru, tri ulazne metode *setLemmatizator, setClassifierData, classify* i tri metode kojima se mogu dohvatiti vrijednosti *getSuggestedDescriptors, getIndexDescriptors, getTextXML*.

Ako se žele dodavati nove funkcionalnosti, dodaju se nove metode, npr. *getLemas*, dok se dio koda koji postavlja privatnu varijablu i dohvaća leme pomoću TMT biblioteke sakriva od Jave unutar neke metode, ovisno o mogućnostima u metodu *classify* ili u samu metodu *getLemas*.

U TMT_wrapper-u se također nalaze i još dvije klase za rad sa Eurovoc pojmovnikom, *EurovocElemJava* i *EurovocJava*. Kao što im i sam naziv govori, one predstavljaju jedan Eurovoc element *EurovocElemJava* i kolekciju Eurovoc elemenata *EurovocJava*. Kao i kod indeksiranja tako je i unutar ove dvije klase sakriven cijeli postupak dohvaćanja Eurovoc elemenata, opisa, širih pojmova i ostalih stvari potrebnih za WEB eCadis, i izloženi su Java sloju preko jednostavnih metoda kojima se dobiju rezultati u obliku String-a ili u obliku Vektora ako su podaci višedimenzionalni.

6.2.2. Sloj JNI

O ovom sloju se ne može puno pričati zato što ga nije trebalo razvijati samostalno. Kao što je prije navedeno, za razvoj JNI sloja je korišten program SWIG čija se dokumentacija može pronaći na stranicama www.swig.org .

Buduci je prije opširnije opisano što je SWIG, ovdje se nalazi samo mali osvrt na to poglavlje i upute za korištenje.

Naime, SWIG je program koji u ovom projektu generira omotač (JNI kod) za C++ kod, točnije za DLL/SO. Pokreće se iz komandne linije ili se može uključiti u razvojno okruženje, ako razvojno okruženje to dozvoljava.

Za korištenje SWIG-a je potrebno napraviti DLL/SO od C++ koda i potrebno je napisati konfiguracijsku datoteku sa ekstenzijom .i koja se predaje SWIG-u i on na osnovu te konfiguracijske datoteke i DLL/SO-a izgenerira omotač.

Izgled .i datoteke je jako sličan .h datoteci (u dokumentaciji od SWIG-a se može pronaći da se umjesto .i datoteke može koristiti .h datoteka) i njoj je opisano koje klase i metode se žele premostiti.

Na slici 6.4 se nalazi primjer .i datoteke u kojoj se vidi da je klasa EurovocElemJava opisana kao u .h datoteci.

```
%module swigjava
%include "std_string.i"
%include "std_vector.i"
%{
#include "wrappers/webjava/WebJava.h"
%}


%exception {
    try {
        $action
    }
    catch (const std::exception &e) {
            jclass clazz = jenv->FindClass("java/lang/Exception")
                jenv->ThrowNew(clazz, e.what());
                return $null;
    }
}

class EurovocElemJava {
public:
         EurovocElemJava();
         ~EurovocElemJava();
         int getID();
         void setID(int ID);
         std::string getName();
         void setName(std::string NM);

private:
         int id;
         std::string name;
```

Slika 6.4 Izgled konfiguracijske .i datoteke koja se predaje SWIG programu

Kako SWIG nije program koji generira samo JNI wrapper za premošćivanje Jave i C++, već služi za premošćivanje raznih viših programskih jezika (skriptnih, interpretiranih i sl.) i jezika koji se prevode izravno u strojni kod (eng. native, jezici poput C++ i C), zato je potrebno naglasiti prilikom SWIG poziva koji od viših programskih jezika se koristi. U ovom

slučaju je to Java. Na taj način SWIG je dobio sve potrebne informacije koje su mu potrebne za generiranje C++ omotača.

Naveden je mali primjer komandno linijskog poziva SWIG programa za generiranje C++ omotača za SO dinamički modul za Linux:

```
%swig -c++ -java example.i
%g++ -c -fpic example.cxx
%g++ -c -fpic example_wrap.cxx -I/usr/java/j2sdk1.4.1/include
-I/usr/java/j2sdk1.4.1/include/linux
%g++ -shared example.o example_wrap.o -o libexample.so
```

Kako bi ovakav način razvoja bio prespor, pogotovo iz razloga što se dio razvoja odvija na Linux, a dio na Windows operacijskim sustavima, tu je u pomoć stigao Cmake Open Source program. Još jedan program koji olakšava rad na ovom projektu. I o tom programu je više bilo riječi u prethodnim poglavljima, a ovdje se osvrće na upotrebu tog programa.

Bit Cmake programa je u pravljenju projekta za različite operacijske sustave i za različita razvojna okruženja na njima. Npr. razvojno okruženje korišteno na Windows operacijskom sustavu je .NET. U konfiguracijskim datotekama za Cmake program se opiše projekt. Opiše se na način na koji Cmake razumije, a upute o korištenju Cmake programa se mogu naći na njihovim WEB stranicama www.cmake.org . Postoje različite naredbe, od toga da se stvori projekt za XY operacijski sustav, za koje razvojno okruženje, koje datoteke taj program uključuje i sl.

Zašto se ovaj tekst nalazi u ovom poglavlju, možda se pitate? Pa razlog je jednostavan, prilikom nadogradnje WEB eCadis aplikacije bilo bi potrebno imati u glavi previše stvari koje je moguće izbjeći korištenjem Cmake programa.

6.2.3. Sloj Java

Do ovog trenutka u ovom poglavlju je opisano kako se teoretski pravi veza C++ i Jave. Praktično se svodi na:

1. doda se nova funkcionalnost - metoda u postojeću klasu u WebJava.cpp datoteci ili se napravi nova klasa u WebJava.cpp datoteci,
2. nova funkcionalnost se doda u .i konfiguracijsku datoteku za SWIG,
3. ako nisu dodani novi .cpp fileovi nije potrebno mijenjati Cmake konfiguracijsku datoteku, no, ako jesu, tada je potrebno uključiti novi file u projekt,

4. pokrene se Cmake program koji stvori projekt za operacijski sustav na kojem se radi i razvojno okruženje u kojem se radi,

5. pokrene se projekt iz razvojnog okruženja.

Što su rezultati pokretanja? Rezultati su: DLL/SO datoteka, JNI most i Java klase koje su analogne C++ klasama koje su bile opisane u .i datoteci.

Naime, u konfiguracijskoj datoteci Cmake programa za razvoj WEB eCadis aplikacije je uključeno više stvari. Cmake napravi projekt koji kada se pokrene kreira DLL/SO, izgenerira JNI omotač i izgenerira Java klase. Kako se to dogodi? Pa dogodi se iz razloga što se u Cmake konfiguracijskoj datoteci nalaze naredbe kojima Cmake uključi SWIG u projekt i konfiguracijske datoteke za SWIG što je vidljivo na slici 6.5. Na ovaj način se razvoj aplikacije maksimalno pojednostavio.

```
FIND_PACKAGE(SWIG REQUIRED)
INCLUDE(${SWIG_USE_FILE})

FIND_PACKAGE(Java REQUIRED)
FIND_PACKAGE(JNI REQUIRED)

INCLUDE_DIRECTORIES(${JAVA_INCLUDE_PATH})

IF(WIN32 OR MINGW OR UNIX)
  INCLUDE_DIRECTORIES(${JAVA_INCLUDE_PATH2})
ENDIF(WIN32 OR MINGW OR UNIX)

INCLUDE_DIRECTORIES(${JAVA_AWT_INCLUDE_PATH})
LINK_DIRECTORIES(${JAVA_AWT_LIB_PATH})

SET(CMAKE_SWIG_OUTDIR ${CMAKE_CURRENT_SOURCE_DIR}/WebPei/src/org/fer/tmt)

SET_SOURCE_FILES_PROPERTIES(WebJava.i PROPERTIES CPLUSPLUS ON)
SET_SOURCE_FILES_PROPERTIES(WebJava.i PROPERTIES SWIG_FLAGS "-includeall;-package;org.fer.tmt")
SWIG_ADD_MODULE(WebJava java WebJava.i WebJava.cpp Classifier.cpp XMLDocumentSpanifier.cpp)
SWIG_LINK_LIBRARIES(WebJava tmt)
```

Slika 6.5 Dio konfiguracijske datoteke kojim se daju upute Cmake programu da uključi SWIG u projekt i upute SWIG programu kako da napravi JNI omotač.

Također se napravi i test_webjava projekt kojim se mogu testirati novododane funkcionalnosti, jednostavnim promjenama u main.cpp programu koji također postoji. Testiranje se svodi na:

1. primjene se nove funkcionalnosti u *main.cpp* datoteci i ispišu rezultati na standardni izlaz

2. pokrene se *test_webjava.exe*

Ako su rezultati zadovoljavajući, može se pokrenuti projekt WebJava (u run modu) čije pokretanje rezultira sljedećim:

1. kreiranje WebJava.dll/so

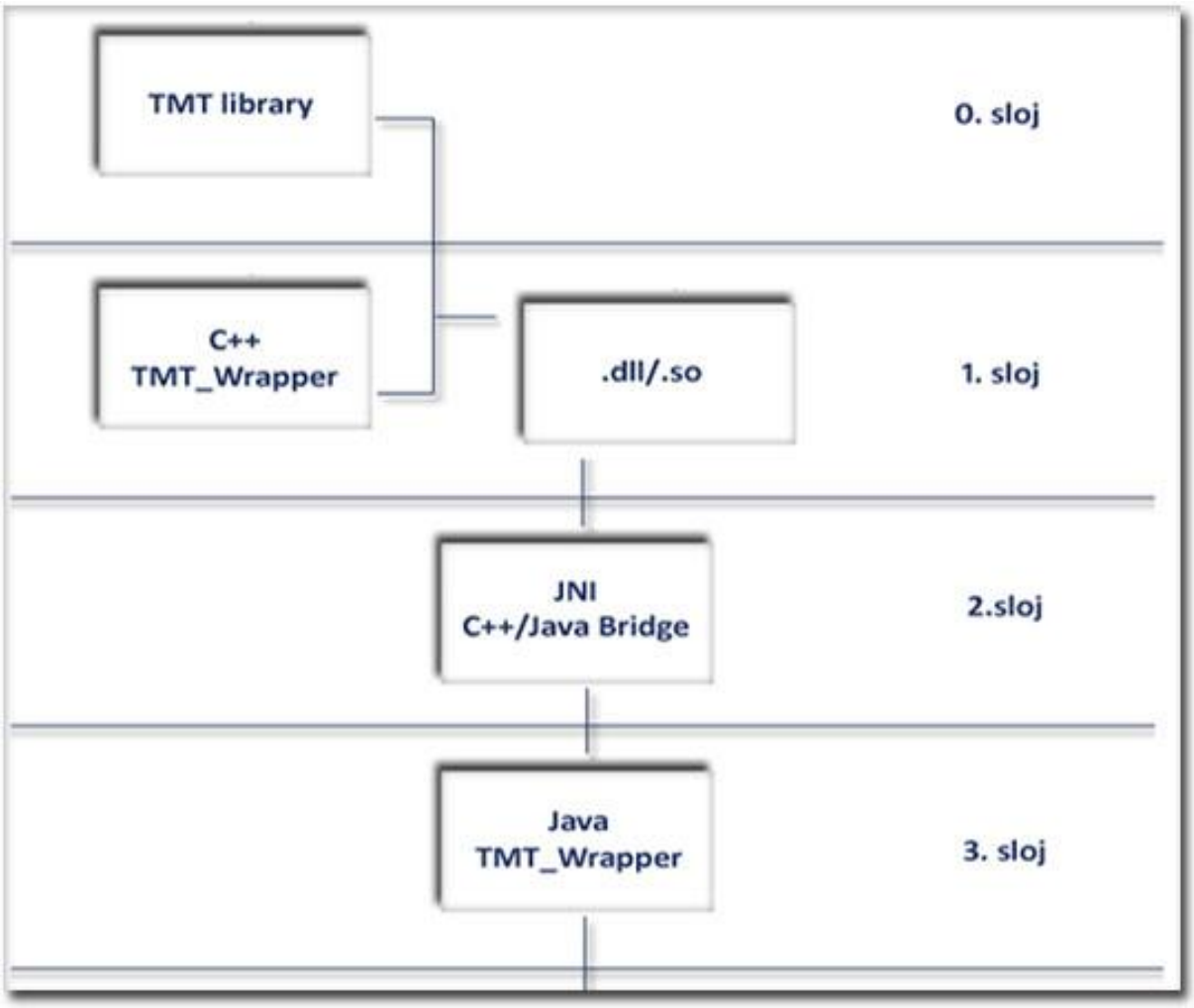
2. generiranje JNI omotača

3. generiranje Java klasa identičnih onima koje su izložene SWIG-u preko .i konfiguracijske datoteke.

4. generiranje Java klasa potrebnih za pozive C++ funkcija preko JNI koda.

Bitno je napomenuti da nije potrebno mijenjati niti jednu od generiranih datoteka, čak štoviše ne bi bilo pametno mijenjati.

Osim SWIG-om izgeneriranih Java klasa, postoje još dvije Java klase, *Eurovoc.java* i *Classifier.java*. U njima su ostvarene jednostavne funkcije koje služe JSP-ovima za dohvat podataka iz TMT biblioteke preko izgeneriranjih Java klasa, JNI sloja do DLL/SO. Može se reći da ove klase pripremaju podatke iz TMT biblioteke tako da ih JSP-ovi mogu lakše formatirati u HTML.



Slika 6.6 C++ Java premošćivanje

Nakon ovog postupka stanje aplikacije je prikazano slikom 6.6. Stvoren je JNI most između C++ i Jave i moguće je nastaviti razvijati aplikaciju u Javi i Java WEB tehnologijama.

6.3. Sloj Controler – Servlet, JSP

Tehnički gledano, MVC arhitektura nije uvela nove tehnologije u razvoj WEB aplikacija, samo je na mnogo praktičniji i ergonomičniji način počela koristiti postojeće tehnologije.

MVC arhitektura je zapravo nastala radom i iskustvom prilikom razvijanja WEB aplikacija. Kako bi misao vodilja svakog programera trebala biti da nikada ne pišu isti dio koda 2 ili više puta, odnosno, ponovna iskoristivost jednom napisanog koda, može se reći da je MVC arhitektura plod jednog takvog razmišljanja.

Takvim modelom se ista baza podataka te jednom napisan program za pristup toj bazi podataka može koristiti prilikom razvoja više WEB aplikacija. Ovo vrijedi za Model, kao i za View. Jednom napravljena prezentacija, HTML stranica za prikaz određenih podataka, nema razloga da ne bude iskorištena ponovno, ako se pojavi potreba.

Potreba za ponovnom iskoristivosti naročito dolazi do izražaja prilikom razvoja većih aplikacija. Iako WEB eCadis aplikacija ne spada u skupinu većih aplikacija, korištenje MVC arhitekture također postaje praktično i prilikom razvoja aplikacija u čijem razvoju sudjeluje više ljudi.

Kako bi u ovom poglavlju trebalo pisati Controler dijelu MVC arhitekture, malo ćemo se osvrnuti i na njega. U klasičnom smislu Controlera, on trenutno ne postoji u WEB eCadis sustavu.

Što je Controler? Npr., u WEB eCadis sustavu, da je postojala potreba omogućiti korisniku da unutar jednog prozora/stranice učita tekst i promijeni jezik indeksacije sa hrvatskog na engleski, zadaća kontrolera bi bila zatražiti od C++ sloja indeksiranje podataka na engleskom jeziku i dohvat podataka na engleskom jeziku. Kada bi se još uvele različite prezentacijske stranice, tj. da se rezultati indeksiranja na engleskom jeziku prikazuju na različitoj stranici od rezultata indeksiranja na hrvatskom jeziku, to bi onda bila zadaća Controler-a.

Općenito, Controler služi tome da na osnovu različitih ulaznih parametara jedan jedini Controler dohvaća različite podatke iz Modela i prosljeđuje ih određenom View-u.

Kako je radi jednostavnijeg korištenja aplikacije smanjena interakcija korisnika s aplikacijom, napravljena je posebna verzija aplikacije za Engleski jezik i posebna za Hrvatski jezik. To je napravljeno zbog smanjenja broja potrebnih akcija da bi se došlo do rezultata. Izbačeno je biranje indeksatora, jezika indeksiranja, jezika prikaza, i sl., čime je korištenje aplikacije maksimalno pojednostavljeno.

Zbog takvih praktičnih razloga je nestao i Controler dio iz aplikacijskog rješenja. No, ipak je s razlogom na slici 6.1 nacrtan 4. sloj kao Java, JSP sloj.

Da postoji Controler u sustavu slika 6.1 bi na drugačiji način prikazivala 3., 4. i 5. sloj. U 4 sloju bi se našli, kako i u naslovu ovog potpoglavlja piše, Java klase i JSP/Servleti. No, kako nema Controlorera u 4. sloj se nalaze serverske strane View – a, JSP stranice. JSP-ovi formatiraju podatke dobivene pozivom Java funkcija iz *Eurovoc.java* i *Classifier.java* klasa u običan HTML kod spreman za direktno ubacivanje unutar stranice.

6.4. 5. Sloj View – JavaScript, JSP

Prezentacijski sloj je dio koji jedino zanima korisnike. Preko prezentacijskog sloja je korisniku ponuđena interakcija s aplikacijom i iz tog razloga je cilj i zadaća, prilikom razvoja bilo kakve aplikacije, napraviti što jednostavniji i intuitivniji prezentacijski sloj.

U prvoj verziji WEB eCadis aplikacije bila je omogućena sva funkcionalnost desktop aplikacije, bilo je moguće mijenjati jezik indeksiranja, jezik prikaza, bilo je moguće mijenjati indeksatore, stop riječi i sve ostale parametre koje je moguće mijenjati u standardnoj verziji aplikacije. Na slici 6.7 prikazan je prozor koji omogućuje mijenjanje postavki eCadis sustava. Nadalje, kako je ovo ipak WEB verzija aplikacije čija je namjena da je koriste šire mase koje nisu toliko upućene u sve mogućnosti eCadis sustava, bilo je potrebno reducirati mogućnosti koje se nude korisniku WEB eCadis sustava te je onemogućeno mijenjanje postavki, iako se po potrebi mogu vratiti.

Slika 6.7 Postavke/Settings

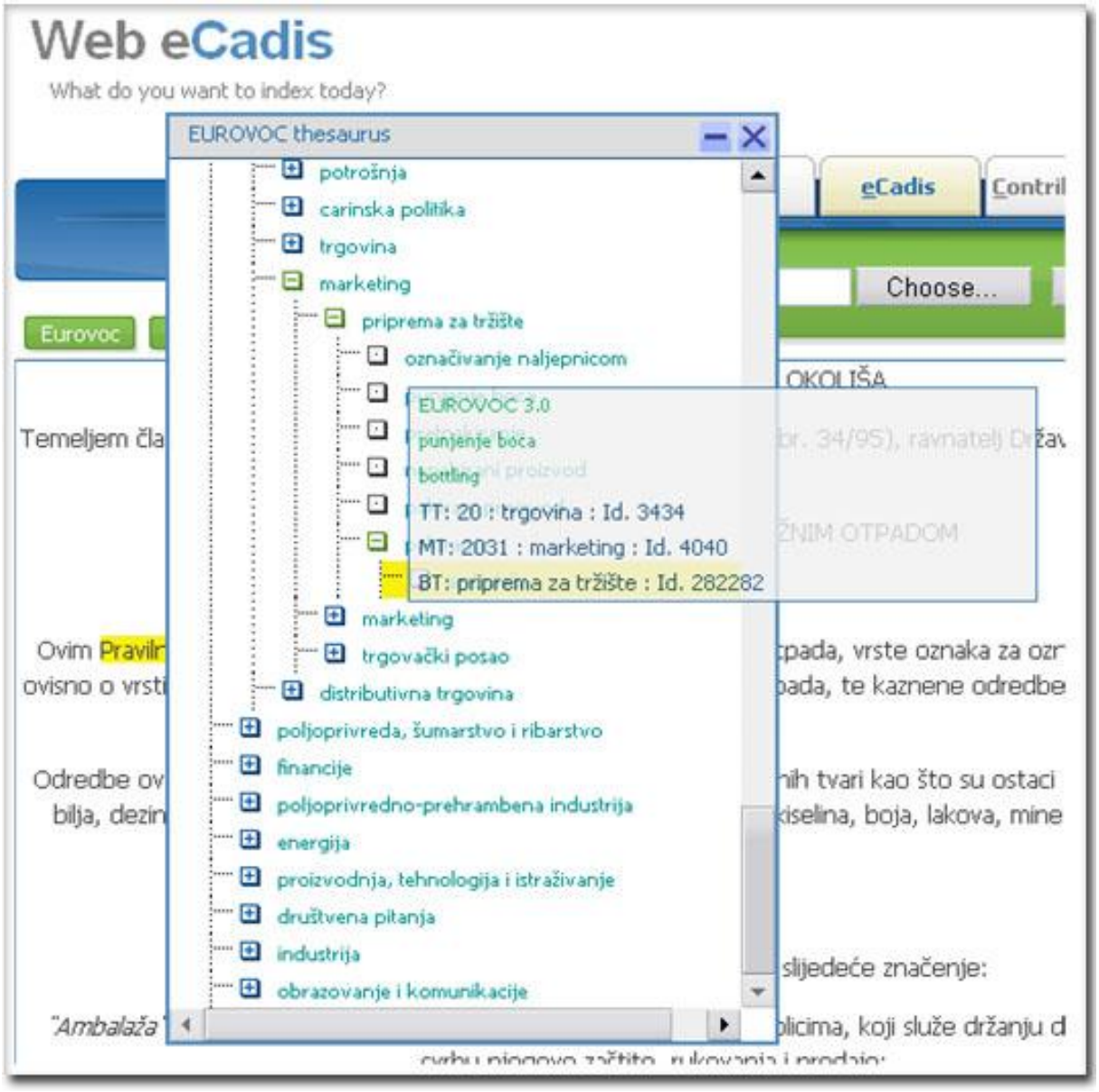
Prezentacijski sloj je predstavljen u WEB eCadis aplikaciji kao kombinacija JavaScript koda i JSP koda. Korištena je AJAX tehnologija opisana u prethodnim poglavljima, koja omogućuje WEB aplikacijama ponašanje i dojam standardnih desktop aplikacija.

Bilo koja akcija koja je omogućena korisniku i koja zahtijeva komunikaciju sa serverom je napravljena pomoću asinkronog HTTP zahtjeva (XMLHttpRequest) i na taj način se izbjeglo konstantno osvježavanje stranice/a i omogućeno je da se cijela aplikacija napravi kao jedna WEB stranica. Svaka korisnikova akcija poziva određenu JavaScript funkciju koja je zadužena za tu akciju. Ako je potrebna komunikacija sa serverom JavaScript funkcija je odradi.

Npr., prikaz Eurovoc stabla. Dohvat svih elemenata Eurvoc stabla, prilikom otvaranja Eurovoc prozora, bio bi jako spor i predugo bi se učitavao na korisničkoj strani. Umjesto toga je napravljena JavaScript funkcija koja je zadužena za pozivanje samo određenog nivoa stabla. Ta funkcija AJAX pozivom prima podatke sa servera preko posebno napravljene JSP datoteke. Naime, ta JSP datoteka je zadužena za poziv Java funkcija koje vraćaju Eurovoc elemente, djecu zadanog elementa. Java funkcije, pak, komuniciraju s TMT datotekom na već prije opisan način.

Da se vratimo na JSP datoteku. Konkretno u ovom slučaju *euThesaurusElements.jsp* koji komunicira sa Java klasama i formatira rezultate dobivene u običan HTML te ga JavaScript funkcija koja je pozvala JSP dobije kao rezultat poziva i može ga direktno uključiti u HTML koji se prikazuje korisniku u njegovom Browseru.

Ili primjer šireg opisa Eurovoc elementa, koji se počne učitavati tek prelaskom pokazivačem miša preko Eurovoc elementa. Prelaskom pokazivača miša preko Eurovoc elementa se pozove JavaScript funkcija *„hoverIn(this)“* kojoj se proslijedi sam element te se AJAX pozivom pozove *euThesaurusElementsDescription.jsp* sa parametrima id elementa i jezika na kojem se želi opis prikazati (odabrani jezik sučelja). U toj JSP datoteci se rade Java pozivi te se pripremi rezultat koji nakon što se učita na korisničnoj strani prikaže korisniku u za to posebno napravljenom prozoru, slika 6.8. prikazuje širi opis Eurovoc elementa *„punjenje boca“* .



Slika 6.8 Prikaz Eurovoc stabla

Prva funkcionalnost koja je bila implementirana u WEB eCadis sustavu je prikaz Eurovoc stabla. Na aplikaciji postoji zeleni gumb „Eurovoc“ kojim se otvara prozor u kojem je prikazano Eurovoc stablo. Stablo se može pretraživati ručno otvaranjem djece klikom na sliku „+“. Također, prelaskom pokazivača miša preko elemenata stabla se otvara prozor sa detaljnijim opisom elementa, TT, MT, BT i sl. što je vidljivo i na slici 6.7.

Kako je prije napisano, ne učitava se cijelo Eurovoc stablo odjednom na korisničkoj strani. Prilikom otvaranja prozora Eurovoc stabla otvori se samo prvi nivo stabla, a za daljnje otvaranje Eurovoc stabla i njegovo pretraživanje, kao i za otvaranje prvog nivoa koristi se funkcija *testiraj(el, path)*. Funkcija prima parametre:

- el – element na koji je kliknuo korisnik
- path - popis svih Eurovoc elemenata koji se žele otvoriti

Kako je postojala i potreba za direktnim otvaranjem stabla, za zadani element je trebalo otvoriti sve njegove roditelje i doći do zadanog elementa iz tog razloga postoji i funkcija *testirajD(el, path)*. Prije nego se pozove ova funkcija potrebno je saznati sve roditelje od traženog elementa, jer se oni ne pamte na Java/JavaScript/HTML sloju. Oni se mogu saznati u TMT biblioteci. Stoga, ako se želi direktno pristupiti nekom elementu Eurovoc stabla, poziva se JavaScript funkcija *direktni(param)* u kojoj se AJAX pozivom *jfGetEuThesaurusElementPath.jsp* dobiju svi roditelji elementa:

```
var xmlhttp = new XMLHttpRequest();
 xmlhttp.open( 'POST', 'jfGetEuThesaurusElementPath.jsp?id=' + param, false);
 xmlhttp.send( null);
```

zatim se obradi dobiveni rezultat:

```
pomocni = xmlhttp.responseText.split(',');

  if(pomocni.length==1){
     var strTMP = "" + parseInt(pomocni[0]);
     time = setTimeout("scrollToTreeNode('nodex" + strTMP  +"_a')", 700);
     return false;
  }if(pomocni.length>1)
    var path = parseInt(pomocni[1]);
  if(pomocni.length>2){
    for(var i=2;i<pomocni.length;i++){
      path +="," + parseInt(pomocni[i]);
    }
  }
```

i potom se pozove funkcija *testirajD(el, path)*

```
testirajD(document.getElementById("nodex" + parseInt(pomocni[0]) +"_pic"),
path);
```

U *jfGetEuThesaurusElementPath.jsp* se radi poziv Java funkcije *Eurovoc.getElemPath(element)*

```
<%
request.setCharacterEncoding("UTF-8");
 int pom = Integer.parseInt(request.getParameter("id").trim());
 response.setContentType("text/xml;charset=UTF-8");
 ArrayList djeca;
 djeca = Eurovoc.getElemPath(pom);
```

te se kao rezultat ispišu elementi razdvojeni zarezom:

```
String value = djeca.get(djeca.size()-1).toString();
 for(int i=djeca.size()-2; i>=0; i--){
    value += "," + djeca.get(i);
 }

 %>
 <%=value%>
```

U Java funkciji *Eurovoc.getElemPath(element)* se preko JNI sloja pozove TMT biblioteka i dobije se lista svih predaka zadanog elementa. Sve JavaScript funkcije vezane za rad sa Eurovoc stablom se nalaze u *TreeWalker.js* datoteci.

Ovo je samo jedan primjer implementacije sučelja, no, u principu, implementacija bilo kakvih funkcionalnosti u sučelju se svodi na sljedeće (odozgo prema dolje):

1. Poziv JavaScript funkcije na neku od akcija (klik, dvostruki klik, i sl.)
2. JavaScript funkcija AJAX-om poziva JSP
3. JSP dobivene podatke od Java funkcija formatira u HTML/običan tekst (po potrebi)
4. Java funkcije dohvaćaju podatke komunicirajući s TMT bibliotekom preko JNI sloja.

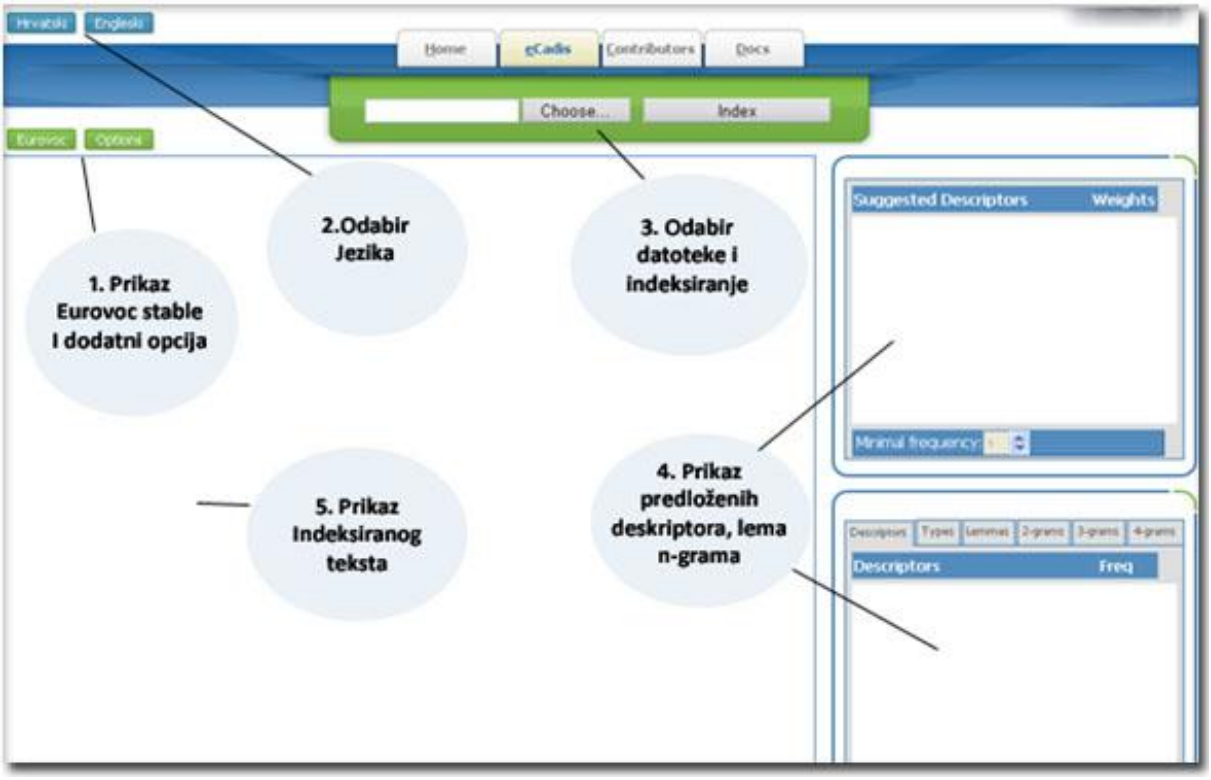
7. Implementacija izvorne funkcionalnosti Pei sustava za prikaz i analizu dokumenata u web tehnologijama

U prethodnom poglavlju je opisana struktura aplikacije od TMT biblioteke prema sučelju, gdje je fokus bio na korištenju TMT biblioteke te premošćivanju C++ i Jave i više baziran na tehničkoj implementaciji aplikacije.

Ovo poglavlje se više fokusira na aplikaciju s korisničke strane te stvaranju vizualnog identiteta aplikacije, odnosno implementaciji izvorne funkcionalnosti Pei sustava i njenog korištenja putem WEB preglednika.

7.1. WEB eCadis

U petom poglavlju su objašnjene korištene WEB tehnologije u WEB eCadis aplikaciji. Pomoću njih se uspjela implementirati većina izvorne funkcionalnosti Pei „desktop“ aplikacije. U ovom poglavlju slijedi opis implementiranih funkcionalnosti pomoću ranije opisanih tehnologija.



Slika 7.1 Sučelje WEB eCadis aplikacije prije indeksiranja dokumenta

Na slici 7.1 je prikaz WEB eCadis aplikacije za indeksaciju dokumenata prije indeksacije. Radi lakšeg objašnjavanja prikazano je pet područja za koja slijedi objašnjenje.

1. Područje sadrži dva gumba, Eurovoc i Options:
    a. Eurovoc gumb otvara prozor u kojem je prikazano Eurovoc stablo koje se može pretraživati
    b. Options gumb otvara prozor u kojem se nalaze postavke eCadis aplikacije, ali koje su radi jednostavnosti korištenja onemogućene te ih obični korisnici trenutno ne mogu mijenjati.
2. Područje sadrži jezike koje podržava WEB eCadis aplikacija te se mogu mijenjati.
3. Područje koje sadrži dva gumba
    a. odabirom prvog gumba (*Choose... ili Browse...*) otvara se prozor s pogledom na korisnikovo računalo gdje se može odabrati datoteka koju se želi indeksirati.
    b. nakon toga se može odabrati drugi gumb koji pokreće indeksaciju odabrane datoteke.
4. Područje je na početku prazno, no nakon indeksacije se popuni:
    a. prvi prozor sa predloženim deskriptorima
    b. drugi prozor, podijeljeno po tabovima, sa deskriptorima, lemama, pojavnicama, n-gramima...
5. Područje, nakon indeksacije se učita dokument koji je indeksiran.

Nakon indeksacije datoteke izgled sučelja se promijeni tako da se područja 4 i 5 popune s odgovarajućim vrijednostima što se može vidjeti na slici 7.2:

Slika 7.2 Izgled WEB eCadis splikacije nakon indeksiranja s otvorenim Eurovoc stablom

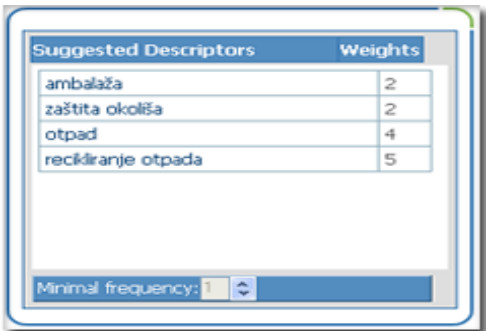
Nakon indeksiranja, datoteka se učita u područje 5 i područje 4 se popuni s odgovarajućim vrijednostima. Na slici 7.2 je prikazan izgled aplikacije nakon indeksiranja i sa otvorenim prozorom Eurovoc stabla. Također su učinjene još 4 akcije:

1. mišem je dvostruko kliknuto na deskriptor *„ambalaža“* pa su u indeksiranom tekstu obojene pojavnice riječi „ambalaža“. Višestrukim dvostrukim klikanjem na deskriptore (pa tako i leme, n-grame, ...) se ide od prve prema zadnjoj pojavnici te riječi. Također se u Eurovoc stablu otvara dio stabla do te riječi.

2. jednom je kliknuto mišem na deskriptor *„potrošač“* pa je u Eurovoc prozoru otvoren dio stabla do riječi „potrošač“ i označen žutom bojom

3. ručno su otvoreni elementi u Eurovoc stablu redom : *„marketing – priprema za tržište“* klikom na sliku „+“ pored svakog elementa stabla.

4. nakon toga se pokazivačem miša došlo do Eurovoc elementa *„nepakirani proizvod“* kada se pojavi pomoćni prozor sa širim opisom elementa.



Slika 7.3 Prikaz prozora s predloženim deskriptorima

Na slici 7.3 se nalazi prozor s predloženim deskriptorima koji ima osim gore navedenih funkcionalnosti (otvaranja elemenata Eurovoc stabla klikom na deskriptor i označavanje deskriptora u indeksiranom tekstu na dvostruki klik) postoji i mogućnost sortiranja tablice abecedno klikom miša na naslov kolone koja ima popis naziva deskriptora ili po vrijednosti klikom miša na naslov kolone sa vrijednostima (težine, frekvencija pojavljivanja).

Osim toga, moguće je odrediti i minimalne vrijednosti koje moraju imati deskriptori da bi se prikazali u popisu.

7.2. Podržani formati datoteka za indeksiranje

Jedini format koji podržava TMT biblioteka prilikom indeksiranja dokumenata je XML. No, kako je XML format rijetko korišten za pisanje običnih tekstualnih dokumenata, postojala je potreba da WEB eCadis aplikacija podržava još dva, češće korištena, formata za zapis tekstualnih dokumenata, a to su .html i .doc.

Tako da su podržani formati dokumenata koje je moguće indeksirati pomoću WEB eCadis aplikacije:

1. XML
2. HTML
3. DOC

U slučaju da predate datoteku sa drugačijom ekstenzijom WEB eCadis aplikaciji dobit ćete poruku da aplikacija trenutno ne podržava tip dokumenta koji želite indeksirati.

8. Srodni radovi

Razmatranja i počeci pristupa problemu automatskog indeksiranja dokumenata javili su se 1957. godine kada Luhon [12] u svom radu predlaže statistički pristup u rješavanju problema, počevši od statističke analize skupa dokumenata, formiranja riječnika i izvlačenja relavantnih informacija iz teksta pa do implementacije odgovarajućeg programa.

1961. Maron objavljuje rad u kojem razmatra rješenje problema automatske klasifikacije dokumenata prema njihovoj tematici. Slično kao i Luhon, predlažu se statističke metode koje pomažu računalu u analizi dokumenata i odabiru njemu odgovarajuće kategorije. Metode se temelje na broju pojavljivanja ključnih riječi u dokumentu.

Treba spomenuti i rad Plaunta i Norgarda [18] na Sveučilištu u Kaliforniji, Berkeley, koji su razvili algoritam za automatsko indeksiranje koristeći statističu vjerojatnost za povezivanje leksičkih jedinki u tekstu (naslov, autori i sažetak) i pojmova u rječniku. Pojmovnik koji je korišten formiran je prema naslovima dokumenata iz seta za testiranje.

Lathinen [20] predlaže u svom radu metodu kreiranja automatskog indeksatora koji proučava sadržaj dokumenta i gradi indeks kombinirajući podatke o frekvenciji riječi i podatke dobivene korištenjem analizatora rečenične strukture. Mnogi komercijalni sustavi poput Indexicona, CIDEX-a i MACREX-a također obavljaju sličnu funkciju.

Od 1982. do 1993. u NASA-i razvijan je sustav za strojno potpomognuto indeksiranje MAI (Machine Aided Indexing System) [22]. Bitna osobina ovog sustava jest što se za indeksaciju dokumenta ne koriste metode strojnog učenja i statističke obrade teksta za predlaganje deskriptora[2]. Za indeksiranje, sustav koristi pojmovnik te bazu koja sadrži parove ključeva i pojmova iz NASA-inog pojmovnika i odvija se proces uspoređivanja niza znakova iz dokumenta sa nizom znakova iz pojmovnika.

NASA-in pojmovnik pokriva široko područje znanosti, inženjerstva i ostalih specijaliziranih tehnologija te sadrži oko 17 800 pojmova. Najbolje se klasificiraju dokumenti iz ovih područja, ali i za dokumente iz drugih područja, MAI daje jako dobre rezultate.

Web aplikacija je napravljena kasnije da stručnjacima i običnim korisnicima omogući korištenje sustava koji određuje sadržaj dokumenta i indetificira ključne riječi i pojmove unutar njega. Aplikaciji se može predati bilo kakav tekst kao ulaz u sustav, npr. kratke sadržaje, čitave dokumente pa čak i web stranice. Dokument koji je predan na indeksaciju sustavu obrađuje se po skupovima riječi. Uzorci nizova riječi unutar dokumentu se uspoređuju sa zapisima u bazi. U slučaju preklapanja sa nekim zapisom u bazi, korisnik dobiva popis deskriptora koji odgovaraju tom zapisu. Proces indeksacije teksta traje samo nekoliko sekundi, a lista dobivenih deskriptora je sortitrana po frekvenciji i dana su na procjenu korisniku.

Prvi rad koji za klasifikaciju dokumenata koristi metode strojnog učenja je rad Ferbera iz 1997. godine u kojem opisuje izgradnju aplikacije koja koristi OECD višejezični pojmovnik. OECD pojmovnik je sličan EUROVOC pojmovniku zbog višejezičnosti. OECD podržava samo četiri jezika i nije toliko opsežan kao EUROVOC pojmovnik. [14]

Opsežan rad u Zajedničkom istraživačkom centru (JRC) Europske komisije imao je za cilj razvoj automatskog sustava za indeksiranje dokumenata. Od 1997. do 2001. godine, Pouliquen, Steinberger su razvili sustav koji dokumente pisane na različitim jezicima indeksira deskriptorima iz istog višejezičnog pojmovnika EUROVOC o kojem će kasnije detaljno biti riječ. Za pridjeljivanje ključnih riječi, sustav se koristi skupom ručno indeksiranih dokumenata za učenje, kako bi konstruirao, za svaki deskriptor, skup s njim konceptualno povezanih riječi iz prirodnog jezika. Svaki deskriptor dobiven ovakvom analizom ima pripadnu frekvenciju pojavljivanja unutar teksta. [16][17] Nastavljajući se na rezultate ovog rada, do 2003. godine, isti je tim znanstvenika razvio potpuno višejezični sustav za automatsko indeksiranje teksta koristeći deskriptore iz pojmovnika EUROVOC.

CISMeF (fra. Catalogue et Index des Sites Médicaux Francophones) je sustav za indeksiranje zdravstvenih dokumenata na francuskom jeziku. Nastao je u razdoblju od 1995. godine do 1999. godine. Područje koje CISMeF pokriva je znanstvena medicina te briga i skrb o pacijentima. Za organizaciju informacija, CISMeF sustav koristi kontrolirani pojmovnik MeSH (eng. Medical Subject Heading) preuzet od US Nacionalne Knjižnice za medicinu i set meta-podataka baziranog na nekoliko setova podataka uključujući Dublin Core format meta-podatka za opis i indeksaciju svih medicinskih dokumenata uključenih u CISMeF, neke elemente IEEE 1484 te HIDDEL set meta-podataka koji poboljšavaju transparentnost i kvalitetu zdravstvenih informacija na Internetu. [21] Sustav koristi prihvaćene Internet standarde za razmjenu podataka u svrhu lakše i pouzdanije razmjene podataka sa ostalim servisima na Internetu. Sustav radi tako da prvo prikuplja informacije na Internetu, filtrira ih i zatim generira kratki opis za taj dokument i obavlja proces indeksacije.

Burnside i drugi su predstavili automatski sustav razvijen u svrhu indeksiranja doktorskih citata rezultata mamografija. Sustav za indeksiranje koristi metodu minimiziranja razlike kvadrata, a koristi klasični BI-RADS rječnik termina.

Ripplinger i Schmidt su predstavili sustav za automatsko indeksiranje dokumenata na engleskom i njemačkoj jeziku, AUTINDEX koji koristi niz sofisticiranih metoda obrade prirodnog jezika da bi dokumentu pridružio skup ključnih riječi bez korištenja pojmovnika ili klasifikacijske sheme. [19]

Projekt CONDORCET na Sveučilištu u Twenteu je imao za cilj razvoj i implementaciju sustava za povrat informacija, dio kojeg je i poluautomatski sustav za indeksiranje dokumenata. Sustav analizira naslove i sažetke dokumenata pomoću koncepata i relacija definiranim u ontološkom pojmovniku.

Montejo Ráez u svom radu iz 2002. godine predstavlja program čiji je cilj olakšavanje rada ljudskih indeksatora, jer su kako on ističe, rezultati postojećih sustava za automatsko indeksiranje dokumenata kvalitetom daleko od indeksiranja obavljenog ljudskom rukom. Sustav je razvijen kako bi pomogao u radu prilikom indeksiranja tematski uskog skupa dokumenata vezanih uz energetsku fiziku. Sustav koristi statistički pristup problemu i korisniku predlaže skupove iz poznatog DESY pojmovnika.

Sustav WEB eCADIS je sustav za automatsko indeksiranje dokumenata na Internetu. Podržava višejezičnost, točnije, indeksaciju dokumenata na engleskom i hrvatskom jezku, no, metode korištene za indeksaciju su napravljene općenito tako da u budućnosti neće biti problem proširiti skup podržanih jezika. Sustav nudi korisniku pregled nađenih deskriptora, lema, n-grama te popis sugeriranih deskriptora. Omogućeno je pretraživanje nađenih lema, deskriptora i n-grama unutar dokumenta kao i prikaz EUROVOC pojmovnika.

Najsličniji je MAI sustavu, od svih gore navedenih sustava. Oba sustava rade preko web sučelja i na taj način su dostupni većem broju korisnika. Podržavaju automatsko pronalaženje deskriptora unutar teksta i pretraživanje ponuđenih deskriptora unutar pojmovnika. [28]

9. Zaključak

Primarni cilj Pei sustava bio je olakšati zadaću indeksiranja službenih dokumenata Republike Hrvatske deskriptorima iz Eurovoc pojmovnika. Izrada sustava je završena u travnju 2006. godine. Pei (Poluautomatski Eurovoc Indeksator) je preimenovan u eCADIS sustav koji obavlja automatsko indeksiranje teksta i omogućuje brže, efikasnije i uniformnije indeksiranje dokumenata u odnosu na ručno indeksiranje.

U današnje vrijeme Interneta i njegove dostupnosti te sve većim brzinama i mogućnostima, kao logičan sljedeći korak bio je napraviti WEB eCadis sustav.

Prvi dio zadatka bio je istražiti moguće WEB tehnologije u kojima bi se mogla ostvariti WEB eCadis aplikacija, a drugi dio implementirati WEB eCadis sustav u odabranoj tehnologiji.

Zahtjevi su na početku bili skromni i nije se očekivalo da bi se mogla ostvariti puna funkcionalnost „desktop“ eCadis sustava. No, marljivim radom su očekivanja nadmašena i uspješno je implementirana gotovo sva funkcionalnost. Dijelovi koji nisu implementirani su odbačeni radi ciljanih korisnika koji se razlikuju od korisnika „desktop“ eCadis aplikacije.

Uspješno je iskorištena TMT biblioteka i njena funkcionalnost je sada dostupna u Java programskom jeziku. Trenutno su dostupne samo funkcionalnosti koje su bile potrebne za implementaciju WEB eCadis sustava, no, od sada postoji znanje i iskustvo te bilo kakva funkcionalnost iz TMT biblioteke lako može biti dostupna u Javi.

Sustav WEB eCADIS je sustav za automatsko indeksiranje dokumenata na Internetu. Podržava višejezičnost, točnije, indeksaciju dokumenata na engleskom i hrvatskom jezku, no, metode korištene za indeksaciju su napravljene općenito tako da u budućnosti neće biti problem proširiti skup podržanih jezika. Sustav nudi korisniku pregled pronađenih deskriptora, lema, pojavnica, n-grama, popis predloženih deskriptora te njihovu pretragu unutar dokumenta kao i prikaz EUROVOC pojmovnika.

10. Literatura

[1] Kolar M, Vukmirović I, Dalbelo Bašić B, Šnajder J. Computer Aided Document Indexing System, Journal of Computing and Information Technology CIT, Vol. 13, No. 4, December 2005.

[2] Kolar M, Šarić F, Vukmirović I. Strojno potpomognuto indeksiranje dokumenata , Zagreb 2006, dostupno na Internet adresi: http://flambard.zemris.fer.hr/rektor/.

[3] Extensible Markup Language (XML) Version 1.0 Specification. World Wide Web Consortium, 2004.

[4] Thesaurus Eurovoc - Volume 2: Subject-Oriented Version. Ed. 3. Annex to the index of the Official Journal of the EC. Luxembourg, Office for Official Publications of the European Communities, dostupno na Internet adresi: http://europa.eu/eurovoc.

[5] Bratanić M, ed. Pojmovnik EUROVOC, 2nd ed., HIDRA, Zagreb, 2000.

[6] Službene stranice text mining grupe, dostupno na Internet adresi: http://textmining.zemris.fer.hr/tmtdoc

[7] David M. Beazley, An Easy to Use Tool for Integrating Scripting Languages with C and C++,Presented at the 4th Annual Tcl/Tk Workshop, Monterey, CA. July 6-10, 1996

[8] Upute za korištenje SWIG alata, dostupno na Internet adresi: www.swig.org

[9] Enterprise Java Computing, Govind K. Seshadri, Cambrige University Press, 1999

[10] Upute za korištenje CMAKE alata, dostupno na Internet adresi: http://www.cmake.org

[11] Šilić A, Šarić F, Dalbelo Bašić B, Šnajder J, TMT: Object-oriented Text Classification Library (FALI REf)

[12] Luhn HP. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. In: IBM Journal of Research and Development, 1(4), 1957.

[13] Maron ME. Automatic Indexing : An Experimental Inquiry. In: Jorunal of the ACM, Vol. 8, No. 3, 1961.

[14] Ferber R. Automated Indexing with Thesaurus Descriptors: A Co-occurrence based Approach to Multilingual Retrieval. In: Research and Advanced Technology for Digital Libraires. Proceedings of European Conference of Digital Libraries 1997.

[15] Pouliquen B, Steinberger R, Ignat C. Automatic Annotation of Multilingual Text Collections with a Conceptual Thesaurus. In: Proceedings of the Workshop Ontologies and Information Extraction Summer School The Semantic Web and Language Technology – Its Potential and Practicalities. Bucharest, Romania, 28 July – 8 August 2003.

[16] Steinberger R. Cross-lingual Keyword Assignment. In: Proceedings of the XVII Conference of the Spanish Society for Natural Language Processing, Jaen, Spain, 12-14 September 2001.

[17] Steinberger R, Pouliquen B. Cross-lingual Indexing. Final Report for the IPSC Exploratory Research Project. JRC Internal Note, October 2003.

[18] Plaunt C, Norgard B. An Association-Based Method for Automatic Indexing with a Controlled Vocabulary. In: Journal of the American Society for Information Science 49(10), 1998

[19] Ripplinger B, Schmidt P. Automatic Multilingual Indexing and Natural Language Processing. In: Proceedings of SIGIR, Athens, 2000.

[20] Lahtinen T. Automatic indexing: an approach using an index term corpus and combining linguistic and statistical methods. Academic Dissertation, University of Helsinki, Faculty of Arts, December 2000.

[21] Službene stranice Catalog and Index of French-language Health Internet resources, dostupno na Internet adresi: http://www.cismef.org/

[22] Silvester JP, Genuardi MT, Klingbiel PH, Machine-aided indexing at NASA, Information Processing & Management, 1994, vol. 30, no 5, p. 631-645.

[23] CARNet - Časopis Edupoint godište II | broj 3 | Zagreb | 20.2.2002.

[24] Dario Sušanj: Java, Znak, Zagreb 1997.

[25] Upute za korištenje PHP jezika, dostupno na Internet adresi: http://www.php.net

[26] .NET tehnologija, dostupno na Internet adresi:http://hr.wikipedia.org/wiki/.NET

[27] Malenica M. Primjena jezgrenih metoda u kategorizaciji teksta. Diplomski rad, Zagreb, rujan 2004.

[28] Zorović I. Web servis za za prikaz statističkih značajki teksta i EUROVOC pojmovnika. Diplomski rad, Zagreb, rujan 2007.

[29] L.L. Earl, (1970), Experiments and Automatic Extracting and Indexing, Information Storage and Retrieval, 6, pp.313-334, Pergamon Press, via (Viestam,2001)