Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Peter Grlica

Tehnike spletnega luščenja podatkov

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00473/2013
Datum: 08.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **PETER GRLICA**

Naslov: **TEHNIKE SPLETNEGA LUŠČENJA PODATKOV**
WEB SCRAPPING TECHNIQUES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Samodejna obdelava podatkov na svetovnem spletu je zelo otežena, ker je večji del podatkov objavljen v delno strukturirani HTML obliki, prilagojeni uporabnikom in ne računalnikom. Če izvajamo ponavljajoče akcije, bi potrebovali orodja in pristope, ki nam takšna opravila olajšajo. Predvsem je težava v prisotnosti dinamične vsebine in uporabi različnih tehnik s strani urednikov spletnih strani, ki računalnikom onemogočajo samodejno obdelavo podatkov na spletni strani. V okviru diplomske naloge preglejte področje spletnega luščenja podatkov in predlagajte svoj pristop in pripravite prototip, ki uporablja napredne pristope anonimizacije (uporaba posredniških strežnikov, Tor omrežja, onemogočanja detektiranja avtomatiziranega dostopa idr.). Prototip preverite na kompleksnem primeru, kjer prikažite vse vidike delovanja. Preglejte tudi zakonodajo, ki velja na tem področju in predstavite ugotovitve.

Mentor:

doc. dr. Dejan Lavbič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Peter Grlica, z vpisno številko **63010205**, sem avtor diplomskega dela z naslovom:

Tehnike spletnega luščenja podatkov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Dejana Lavbiča
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15. marca 2013 Podpis avtorja:

Pri pisanju diplome se zahvaljujem mentorju za strokovno pomoč, vodenje in nenehno dosegljivost pri problemih.

Kazalo

Seznam uporabljenih kratic

HTML (ang. HyperText Markup Language) Označevalni jezik, ki se uporablja pri opisu strukture spletnih strani.

XHTML (ang. Extensible HyperText Markup Language) Družina označevalnih jezikov na osnovi XML, ki nadgrajuje HTML jezik.

XML (ang. Extensible Markup Language) Označevalni jezik, ki definira pravila za strukturiranje dokumentov v človeku in računalniku razumljivi obliki.

PHP (ang. PHP: Hypertext Preprocessor) Programski jezik, najbolj pogost pri implementaciji spletnih strani.

AJAX (ang. Asynchronous JavaScript and XML) Tehnika spletnega razvoja, uporabljena v prid asinhronim povezavam iz odjemalca na strežnik.

API (ang. Application Programming Interface) Programski vmesnik do specifične spletne storitve.

RSS (ang. Rich Site Summary) Družina formatov spletnih virov, ki se uporabljajo na spletnih straneh z novicami ali blogih.

W3C (ang. World Wide Web Consortium) Mednarodna organizacija, odgov-

orna za upravljanje s standardi na spletu.

ID3 Vsebnik meta podatkov, najbolj pogosto prisoten v MP3 datotekah.

EXIF (ang. Exchangeable image file format) Standard, ki se uporablja za specifikacijo formatov slik in zvoka pri vhodno/izhodnih napravah.

OCR (ang. Optical character recognition) Mehanizirana pretvorba izrisanega teksta v elektronsko obliko.

URL (ang. Uniform Resource Locator) URL se uporablja za določanje lokacije dokumentov na spletu.

BDD (ang. Behavior Driven Development) Proces razvoja programske opreme, ki temelji na testno usmerjenem razvoju, pomaga pa tehnično in vodstveno usmerjenim kadrom pri testiranju funkcionalnosti.

XMP (ang. Extensible Metadata Platform) ISO standard, ustvarjen s strani podjetja Adobe, namenjen procesiranju in izmenjavi standardiziranih in prilagojenih meta podatkov.

DOM (ang. Document Object Model) Programski vmesnik za prikaz in delo z gradniki v dokumentih HTML, XHTML in XML.

MVC (ang. Model View Controller) Zasnovni vzorec pri načrtovanju in implementaciji programske opreme, ki ločuje poslovno logiko od uporabniške interakcije.

CRUD (ang. Create, Read, Update, Delete) Osnovne funkcije trajnega pomnenja; opisuje metode, ki prikazujejo, iščejo in spreminjajo podatke v uporabniškem vmesniku.

SGML (ang. Standard Generalized Markup Language) Splošni označevalni jezik, ki opisuje druge standardizirane označevalne jezike, iz njega izhajata HTML in XML.

MIT (ang. Massachusetts Institute of Technology) Inštitut za tehnologije v Massachusettsu.

AWS (ang. Amazon Web Services) Amazonove storitve v oblaku, ki nudijo poceni alternativo gručam strežnikov.

EC2 (ang. Amazon Elastic Compute Cloud) Amazonova spletna storitev, ki ponuja virtualne instance strežnikov prilagodljive velikosti.

SSD (ang. Solid State Drive) Diski, ki namesto magnetnih plošč uporabljajo flash pomnilnik.

NoSQL (ang. Not Only SQL) Horizontalne podatkovne baze, ki za razliko od klasičnih relacijskih baz ne potrebujejo tabelarične sheme.

P2P (ang. Peer 2 Peer) Omrežna arhitektura, kjer so naprave, računalniki med seboj enakovredni, nastopajo kot strežniki in kot odjemalci hkrati.

Seznam slik

Povzetek

V diplomski nalogi smo poskušali analizirati različne metodologije dostopa do nestrukturiranih podatkov na spletnih straneh. Posvetili smo se predvsem načinom izločanja informacij iz predstavitvene plasti (HTML parsing) z uporabo specifičnih orodij, ki jih lahko najdemo v open source skupnosti, kot tudi pomanjkljivostim trenutnih komercialnih luščilnih orodij in spletnih storitev.

Zaradi izkušenj s programskim jezikom PHP smo uporabili orodja, napisana v tem jeziku. Pod drobnogled smo vzeli tehnike spletnega luščenja s knjižnico Curl v povezavi z Xpath, uporabo "headless" brskalnikov za napredno izločanje podatkov na straneh, kjer se uporablja tehnologija AJAX za prikaz podatkov in orodja za avtomatizacijo testiranj spletnih aplikacij Mink.

Ker je uporaba spletnih agentov v porastu, je tudi preprečevanje dostopa veliko bolj prisotno, zato smo prikazali tudi tehnike anonimizacije (uporaba Tor omrežja in posredniških strežnikov), ter same omejitve dostopa agentov. Ob tem smo pa omenili še zakonodajo in sivo območje legalnosti tega početja, ter zgodovino bolj znanih primerov, kjer je prišlo do tožb.

Na koncu smo predstavili še pozitivne in negativne strani implementiranega luščilca, kot tudi možne nadgradnje in razširitve s strani paralelizacije zahtev in porazdeljenega izvajanja na različnih strežnikih.

Ključne besede: PHP, Mink, luščenje podatkov, AJAX, anonimizacija, zakonodaja, avtomatizacija, odprta koda

Abstract

In this thesis we tried to analyse different methodologies of access to unstructured data on websites. Our main focus was on different techniques of gathering information from presentation layer (HTML parsing) with the use of specific tools that we can find in the open source community as well as downsides of commercial data scrapers and scraping services.

Because of experience in PHP programming language and a plethora of tools, libraries and products implemented in it, we focused on techniques of web scraping with Curl library in combination with Xpath. Other techniques were also the use of "headless" browsers for advanced scraping of data on websites where AJAX requests are used extensively and a tool for automatization of website functionality testing Mink.

With the rise and demand of webcrawlers many content providers try to disable access for them by tracking access of the bots. There are different uses of anonymization tools and user identification techniques being used on websites that we analyzed, as well as tackled the legislation concerning webscraping and most widely known legal cases in this industry.

Lastly, we mentioned positive and negative aspects of the implemented scraper, as well as upgrading and extending the implementation in terms of request parallelization and distributed control on different servers.

Keywords: PHP, Mink, webscraping, AJAX, anonymization, legality, automatization, open source

Poglavje 1

Uvod

Spletno luščenje je proces, ki obsega iskanje podatkov v delno strukturiranih dokumentih[1]. Vsebuje še analizo strukture in prepoznavo vzorcev ter tehnike pridobivanja teh podatkov za uporabo v drugem kontekstu. Je kot nekakšen programski vmesnik API med programsko opremo in ciljno spletno stranjo. API vmesniki do spletnih storitev so vedno bolj pogost pojav, vendar so v mnogih primerih omejeni na število zahtev ali pa vsebujejo le osnovne informacije. Ker so podatki, ki jih programski vmesniki posredujejo odjemalcem, strukturirani za mehansko obdelavo, je interpretacija teh struktur hitrejša. Poleg tega pa celovitost podatkov in prilagodljivost standardom pri izpisu predstavitvene plasti (HTML) ni zagotovljena. Koncept spletnega luščenja dodatno otežijo večpredstavne vsebine, kot sta Flash in AJAX tehnologiji, podatki v slikah in omejevanje dostopa s CAPTCHA sistemom.

Temo za diplomsko nalogo sem si izbral predvsem zaradi potrebe po avtomatiziranem iskanju letalskih kart, saj mi je v želji za čim cenejšo karto kmalu upadla motivacija zaradi nenehnega klikanja po internetnih straneh. Na pomoč sem se zatekel h komercialnim luščilnim rešitvam, ena izmed teh je spletna storitev v obliki zbiralnika uporabniških luščilnih skript. Z znanjem programiranja v jeziku Python, Ruby ali PHP lahko enostavno razvijemo svoj spletni luščilec in ga kot vtičnik uporabljamo na namenskem strežniku. Z brezplačnim dostopom pa ne moremo izkoriščati večje porabe pasovne širine,

implementirano luščilno skripto, pa moramo deliti z ostalimi uporabniki.

Na voljo imamo tudi namizne aplikacije, ki nam omogočajo podatkovno rudarjenje po enostavnih množicah podatkov. V večini primerov so to že vnaprej sestavljeni makroji, ki nam omogočajo enostavno spletno luščenje prek vgrajenega spletnega brskalnika. Nekatere različice omogočajo tudi specificiranje lastnega uporabniškega agenta in povezavo čez posredniški strežnik. Nimamo pa možnosti paralelizacije in določanje prioritete zahtev. Pri izvajanju dolgotrajnih in kompleksnih nalog, kar se tiče vhodno-izhodnih operacij in prenosa podatkov na mrežnem nivoju, pa je pomembno, da imamo čimmanj procesov, ki te operacije zavirajo, kot so morebitni antivirusni programi in P2P klienti, ki se izvajajo v ozadju. Pri luščenju z namiznimi aplikacijami pa moramo postaviti bodisi strežnik z grafičnim vmesnikom, ali luščiti na osebnem računalniku.

Na podlagi izkušenj v razvoju spletnih strani, sem se odločil za uporabo orodij za avtomatizacijo testiranja spletnih storitev, primarno zaradi preprostega dela s HTML gradniki spletnih strani. Iskalniki letalskih kart (pa tudi veliko drugih spletnih strani v turizmu) namreč uporabljajo uporabniški vmesnik, razširjen s tehnologijo AJAX, ki mogoče celotno uporabniško izkušnjo nadgradi, pri spletnem luščenju pa upočasni dostop do podatkov. Dostop do HTML strukture brez preverjanja statusa vsebine je še zmerom mogoč, ampak v tem primeru prenesemo samo uporabniški vmesnik brez same vsebine. Le-ta se prikaže naknadno z asinhronimi AJAX zahtevami. Komercialne luščilne storitve v obliki uporabniških skript tega problema ne rešujejo, saj se vse zahteve izvajajo v oblaku. Implementacija luščilca v oblaku, ki ves nabor javnih skript periodično izvaja, pa deluje v brezokenskem načinu in ne podpira akcij nad vizualnimi gradniki v brskalniku, kot so povleci in spusti ali klik na gumb. To sicer omogočajo namizne aplikacije, a nam tu zmanjka paralelizacija in možnost hranjenja podatkov v specifični bazi. Na podlagi teh zahtev sem se odločil za implementacijo lastnega luščilca in določene pomanjkljivosti poskušal odpraviti. Osredotočil sem se na možnost paralelizacije zahtev, shranjevanje podatkov v kakršnokoli podatkovno bazo in anonimizacijo za-

htev prek posredniških strežnikov, v želji za čimvečjo uporabniško simulacijo, pa sem uporabil orodja za avtomatizacijo testiranj. S tem sem poskušal implementirati nekakšen programski vmesnik med luščilcem vsebine strani in temi testnimi orodji, ki so podatke pridobili.

Struktura in vsebina poglavij v diplomski nalogi je podana v nadaljevanju. V 2. poglavju sem opisal tehnologijo, ki se v prostokodni domeni uporablja pri podatkovnem luščenju, 3. poglavje pa je namenjeno pregledu trenutnega stanja na trgu brezplačnih in plačljivih orodij, s katerimi je mogoče v določeni meri luščiti podatke s spletnih virov. 4. poglavje opiše teoretično zamisel realizacije spletnega luščilca v odprtokodni domeni, v 5. poglavju pa sem v dveh sklopih opisal implementacijo, namestitev ter vmesnik osnovne in napredne verzije lastnega luščilca. Temu v zadnjem, 6. poglavju, sledi analiza možnih posodobitev trenutne implementacije.

Poglavje 2

Osnovni pojmi

Spletni luščilec lahko implementiramo v skoraj vsakem programskem jeziku, naj bo to skriptni jezik Perl, lupina Bash ali Objective C. PHP smo izbrali zaradi enostavnosti namestitve zunanjih orodij prek sistema Composer, privzeti podpori Curl knjižnice in regularnih izrazov ter ogrodja Symfony. Prav pa je, da se spoznamo tudi z načini optimizacije dostopa do spletnih virov, bodisi prek posredniških strežnikov ali šifrirane povezave.

2.1 Semantični splet

Izraz semantični splet se je pojavil že v koncu 90-ih let prejšnjega stoletja, ko se je .com mehurček šele razpočil. Že v tistem času je takratni internetni iskalnik Google poskušal približati podatke - te je s pomočjo svojih spletnih robotov indeksiral in luščil informacije - uporabnikom svetovnega spleta na enostaven in prijazen način. V začetku prejšnjega desetletja se je že začelo gibanje "semantic web" s strani organizacije W3C, razvoj pa že od takrat poteka počasneje, kot so si obetali[2].

Osnovna ideja semantičnega spleta je namreč povezljivost vseh podatkov na internetu na strojem razumljiv način. V pričetku internetne dobe je bil poudarek predvsem na datotekah, njihovem deljenju in spletni infrastrukturi (s stališča ponudnikov vsebin). Proti koncu desetletja pa se je pojavil web 2.0,

kjer ni bil več poudarek na datotekah, ampak na povezljivosti podatkov (tu se je predvsem razširila uporaba tehnologij API in RSS). Tudi pri spletu (še zdaj) nimamo semantične povezljivosti med različnimi storitvami in podatki. Tu pa tiči trenutno največji problem semantičnega spleta - strojno učenje in razvoj umetne inteligence.

2.2 Semantika in sintaksa

Princip dostopa podatkov ne deluje več v striktnem iskanju, ampak na korelaciji med podatki in vsebino. Poudarek je predvsem na tem, da stroji ne pridobivajo podatkov na podlagi strukture, ampak na podlagi pomena. Kar je uporabniku preprosto, a zamudno opravilo (najdi knjižnico v bližini in se včlani vanjo), je trenutno velik izziv za računalnike. V ta namen so bila razvita različna orodja:

- ontologije
- meta podatki

2.2.1 Ontologije

Ontologije so gradniki za organizacijo informacije in se uporabljajo na področju umetne inteligence, semantičnega spleta, bibliografskih sistemov in informacijskih struktur kot predstavitev znanja okoliškega sveta. Predstavljajo formalno znanje in nabor konceptov (tudi razmerje med različnimi koncepti) v domeni, jo opisujejo s pomočjo razumevanja entitet. Te entitete so lahko razredi (nizi, množice, zbirke), atributi (lastnosti, parametri) in razmerja (povezave) med njimi, vsebujejo informacijo o njihovem pomenu in omejitve pri uporabi. Specificirane so v jezikih, kjer jih lahko ločimo od podatkov, njihovih struktur in implementacijske logike, ki jih uporablja.[3]

2.2.2 Meta podatki

Ker je strojem trenutno težko razbrati pomen iz podatkov na spletu, se poleg obstoječe vsebine podaja tudi informacije o podatkih na strojem razumljiv način. Ti meta podatki so ponavadi podani na strukturiran način in vsebujejo informacije o avtorju, času in datumu nastanka, lokaciji te vsebine, uporabljenih standardih ter o namenu teh podatkov.

Primeri meta podatkov:

- ID3 tag v mp3 datotekah
- EXIF, XMP, geografski podatki pri slikovnih datotekah
- zapis pogovora in informacije o vsebini v video zapisih
- meta podatki v HTML datotekah (meta tag)

2.3 Luščenje podatkov

Luščenje podatkov se nanaša na tehniko mehanskega procesiranja vsebin, ki so prikazane na ljudem intuitiven in berljiv način.

Običajno je povezava med dvema sistemoma zgrajena na osnovi izmenjave podatkov z uporabo vnaprej določenih in standardiziranih podatkovnih struktur, prilagojenih za avtomatsko obdelavo. Te strukture sledijo določenim standardom, formatom, so definirane in dokumentirane, predvsem pa omogočajo strojem enostavno luščenje podatkov na konsistenten način. Tu predstavitveni nivo ni pomemben, saj so podatki namenjeni strojem in ne ljudem. Pri luščenju podatkov gre za razliko od navadne izmenjave med dvema programoma pri pridobivanju informacij iz podatkov, ki so namenjeni samo prikazovanju končnim uporabnikom, zato so velikokrat slabo ali nič dokumentirani, imajo strukturo, ki ni namenjena luščenju ali kot vhodni podatek drugemu programu.

Pri podatkovnem luščenju nas tako ne zanimajo elementi predstavitvene plasti, kot so slike (če niso del zahtevanih podatkov), multimedijske vsebine ali konstrukti uporabniškega vmesnika. Večinoma se uporablja kot vmesnik

pri zastarelih sistemih, ki ne omogočajo vmesnika uporabniškega programa API ali so nezdružljivi z novejšo strojno opremo.

Podatkovno luščenje je v računalniški znanosti predstavljeno kot začasna rešitev, saj implementacijo spremlja veliko balastne komunikacije, predstavitveni nivo, namenjen interpretaciji človeških bitij, pa se pogosto spreminja, zato prihaja pri luščenju do napak.

2.3.1 Luščenje prikazovalnikov

Podatke, ki niso namenjeni komunikaciji oz. prenosu na drugi sistem in kjer je problem predvsem v nezdružljivi strojni opremi luščimo kar iz zaslona.

Kot primer lahko vzamemo ponudnike finančnih podatkovnih zbirk Reuters, Telerate in Quotron, njihovi sistemi so prikazovali podatke v terminalih velikosti 24x80 znakov končnim uporabnikom. Investicijske banke pa so na drugi strani (kot registriran uporabnik sistema) luščile podatke iz terminala in jih uporabljala v svojih podatkovnih bazah in finančnih izračunih, s temi programi (Reuters je imel temu namenjen celoten sistem Logicizer, ki je bil osnovan na operacijskem sistemu VMS), pa so nadomestili vmesni člen - človeka, ki je te podatke prepisoval iz terminala v interni poslovni sistem.

V današnjem času se luščenje prikazovalnikov uporablja pri optičnem prepoznavanju znakov (OCR - Optical Character Recognition), kjer so vhodni podatki v obliki slik, papirnatih dokumentov, na roko spisanih listin, itd. OCR spada v raziskave na področju prepoznave oblik, umetne inteligence in računalniškega vida, trenutno z 71 do 98 odstotnim uspehom prepoznave naključnih časopisnih strani, tiskanih v latinici.

Kljub temu, da je uspeh komercialnih OCR programov sorazmerno uspešen, je pri validaciji nujno potreben človek. Tega principa se je dodatno poslužil Google leta 2009, ko je investiral v algoritem reCAPTCHA ter si pomagal pri validaciji neuspelih OCR rezultatov (tudi tistih s prenizko verjetnostjo rezultata) s pomočjo množic. Ta sistem so v naslednjih letih uporabile strani za validacijo človeka in preprečile dostop spletnim agentom, ter s tem preverile 100 milijonov zapisov na dan.

2.3.2 Spletno luščenje

Na internetu je trenutno neznatna količina podatkov, shranjenih v podatkovnih bazah na strojem razumljiv način, obdelanih v programskih jezikih, kjer se sintaksa človeka sreča s sintakso stroja in kjer se semantični prepad venomer manjša. A kljub temu so ti podatki, ki so namenjeni nadaljnjemu deljenju med uporabniki in spletnimi stranmi prikazani na načine, ki strojem omejujejo procesiranje le-teh.

Spletno luščenje je - podobno kot luščenje prikazovalnikov - povratni inženiring predstavitvene plasti, v primeru spleta internetnih strani v obliki HTML/XHTML datotek, ki jih program nato dodatno analizira in obdela. Pri "scraperjih" poteka analiza po celotni strukturi dokumenta, algoritem namreč išče točno določene vzorce v strukturi, ki je ponavadi predstavljena samo ljudem. Lahko bi rekli, da je "scraper" odjemalec za spletno storitev, kjer v tem primeru na drugi strani strežnik ne podaja programskega vmesnika API, ampak človeku prilagojeno vsebino. Tu se pogosto pojavlja problem v strukturni celovitosti podatkov, saj so spletni brskalniki dosti bolj tolerantni do teh napak (pri prikazu vsebine), kot spletne storitve, kateri so namenjeni samo strojem. Tako se mora spletni "scraper" prilagajati strukturnim napakam, tu pa preži največja past spletnega luščenja.

2.4 Tehnologija, uporabljena pri luščenju

2.4.1 DOM

Document Object Model (DOM) je programski vmesnik API za strukturo datotek HTML in XML. Definira logično strukturo dokumentov in način, kako se do njih dostopa ter kako se z njimi upravlja. Nastal je kot specifikacija, ki dovoli Javascript in Java programom prenosljivost med spletnimi brskalniki.

Njegov prednik – "Dynamic HTML" – je bil prvotno mišljen v domeni spletnih brskalnikov, a se je pozneje ideja razširila na druga področja, kot so bili HTML in XML urejevalniki. Te implementacije so zaradi prejšnjih

izkušenj z orodjem SGML vplivale na samo zasnovo DOM dokumenta.

V DOM specifikaciji se izraz dokument uporablja v širšem smislu, XML se namreč vse bolj uporablja kot način predstavitve različne vrste informacij, ki so lahko shranjene v različnih sistemih. Te datoteke XML bi se lahko po definiciji interpretirale kot surovi podatki in ne dokumenti, DOM pa te podatke upravlja. Organizacija W3C je izbrala takšno ime zato, ker je DOM načrtovan v tradicionalnem objektno usmerjenem smislu. Dokumenti so namreč obravnavani kot objekti, model ne definira samo strukture dokumenta, ampak tudi logiko dokumenta in njegovih gradnikov.

DOM dokument specificira:

- vmesnike in objekte, ki predstavljajo in manipulirajo z dokumentom
- obnašanje in lastnosti objektov in vmesnikov (semantiko)
- razmerja ter sodelovanje med njima

Po specifikaciji W3C je med najvišjimi prioritetami zagotoviti enoten programski vmesnik, ki se lahko uporablja v različnih okoljih in aplikacijah, načrtovan pa je tako, da do njega dostopamo iz katerega koli programskega jezika. Tako lahko s specifikacijo DOM spreminjamo strukturo dokumenta, mu dodajamo, spreminjamo ali brišemo vsebino.

Dokumenti programskega vmesnika DOM imajo logično strukturo, zelo podobno skupini dreves – "forest". Vsak dokument vsebuje največ eno doctype vozlišče, korensko vozlišče – to služi kot koren drevesa elementov v dokumentu - in več komentarjev ali procesirnih navodil. DOM dokument torej združuje množico gradnikov (objektov), vsak gradnik pa definira svoj nabor lastnosti in obnašanja. DOM specifično ne navaja obvezne drevesne strukture ali obnašanja vsebovanih gradnikov, z drevesi opisuje način sprehoda po dokumentu na že uveljavljene pristope pri pregledovanju takšnih struktur.

2.4.2 HTML

HTML ali "Hypertext Markup Language" je preprost format podatkov, ki se uporablja za ustvarjanje HTML dokumentov, ki so preprosto prenosljivi med

različnimi sistemi. V množično uporabo je prišel že leta 1991 kot opis prvih 18 elementov enostavnega HTML dokumenta, pred tem, pa je obstajal le kot skupek neformalne dokumentacije.[4]

Celoten nabor značk v HTML besedilu nam omogoča:

- objavljanje elektronskih dokumentov z naslovi, besedili, tabelami, seznami, fotografijami
- pridobivanje spletnih informacij prek spletnih povezav
- postavljanje vnosnih polj za upravljanje z oddaljenimi storitvami in uporabi pri iskanju informacij, naročilih
- vključevanje ostalih binarnih dokumentov – video in avdio vsebin v dokument

Vsaka naslednja različica je poskušala izražati večje soglasje v industriji, saj je doseganje interoperabilnosti ključnega pomena pri zniževanju stroškov za ponudnike vsebin in preprečevanju nastajanja novih nezdružljivih in zaprtih formatov, ki bi le zmanjševali uporabnost spleta.

HTML je bil razvit z vizijo, da bi lahko vse naprave učinkovito uporabljale standard za pridobivanje informacij na spletu, od računalnikov z različnimi operacijskimi sistemi in različnih arhitektur, do mobilnih telefonov, dlančnikov ali celo vhodno/izhodnih naprav za pomoč invalidnim uporabnikom.

Že v osnovi je na sintakso močno vplival standardizirani splošni označevalni jezik SGML, prvotno je bil HTML osnovan na tem jeziku, pozneje pa se ločeno razvil in sedaj vsebuje le še manjši nabor značk. Sintakso je določal SGML Document Type Definition – DTD. Po prvi verziji HTML sta sledila še HTML+ in HTML 3.0, ki sta standardizirala tabele in vnosna polja, ter dodatno nadgradila z izboljšavami. Tej verziji sta v kratkem sledili še 4.0 in 4.1 z dodatnim naborom elementov, predvsem pri delu s tabelami, slikami in vnosnimi formami.[5]

HTML je bil do trenutne verzije osnovan na SGML tehnologiji – "Standard Generalized Markup Language" – standard organizacije ISO.

Glavne prednosti SGML so bile:

- razvit je bil lahko na kateremkoli sistemu
- jezik je neodvisen od oblikovalnika
- uporaba parov značk (npr. <h1></h1>)

Hiperpovezave – funkcionalnost, nepodprta v SGML jeziku – je Tim Berners Lee dodal v HTML naknadno (kot "anchor" element) in s tem še povečal uporabnost in popularnost HTML jezika.

Medtem, ko je bil HTML opredeljen kot uporaba splošnega jezika SGML, se je kmalu pojavil nov standard XHTML, ki temelji na jezikovnem ogrodju XML. XHTML dokumenti so, za razliko od HTML razčlenjevalnika, ki je dopuščal bolj svobodno uporabo sintakse, morali imeti bolj striktno strukturo zaradi posledične uporabe XML razčlenjevalnikov. XHTML je družina trenutnih in prihodnjih dokumentov, ki razširjajo HTML 4, obenem pa se podrejajo XML standardu.

Največje pridobitve pri prehodu iz HTML na XHTML:

- XHTML dokumenti so skladni z XML strukturo, zato jih lahko pregledujemo, spreminjamo in validiramo z obstoječimi XML orodji
- XHTML dokumente lahko uporabljajo aplikacije, katere se ravnajo po HTML DOM, ali po programskem vmesniku XML DOM
- dokumenti v XHTML sintaksi lahko delujejo enako ali celo bolje pri uporabniških agentih, ki se podrejajo standardu HTML 4 ali XHTML 1

Prihodnost jezika HTML

Od leta 2004 pa je v fazi osnutka že HTML 5, nadgradil bo dokumente HTML 4, XHTML 1 in DOM2HTML – programski vmesnik Javascript za HTML/XHTML, obenem pa je že združljiv s prejšnjimi inačicami. V poskusu, da bi bil jezik za razvojnike čim lažji za uporabo, so ovrgli uporabo nekaterih

elementov in značk, ter jih premestili v druge jezike (npr. predstavitveni elementi v CSS).[6]

Osnutek vsebuje še:

- HTML 5 določa jezik HTML, ki dopušča pisno sintakso HTML ali XML
- določa podrobne modele za vpeljavo interoperabilnih implementacij
- izboljše oznake za dokumente
- uvaja oznake in programski vmesnik za prihajajoče tehnologije
- podpora MathML in SVG vsebini
- razširjena podpora audio in video vsebinam

Koda 2.1 : HTML 5 sintaksa v HTML obliki

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Example document</title>
  </head>
  <body>
    <p>Example paragraph</p>
  </body>
</html>
```

Koda 2.2 : HTML 5 sintaksa v XML obliki

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Example document</title>
  </head>
  <body>
    <p>Example paragraph</p>
  </body>
</html>
```

2.4.3 XML

XML je razširljivi označevalni jezik za dokumente, ki vsebujejo strukturirane informacije. Te datoteke lahko vključujejo besedilo, slike, zvok, obenem pa vključujejo tudi meta podatke o tej vsebini in imajo določeno strukturo. Označevalni jezik se tako uporablja za določanje strukture v dokumentih, XML specifikacija pa definira standardni način za dodajanje oznak k dokumentom.

Pri XML specifikaciji, za razliko od HTML, nabor značk in semantika značk ni fiksna. Pri HTML specifikaciji je nabor teh značk odvisna od proizvajalcev spletnih brskalnikov, ter njihovih zahtev. Pomembna je tudi združljivost s starejšimi sistemi, zato je vpeljava novih značk počasen in nadzorovan proces. Pri XML specifikaciji pa tega problema ni, saj ne določa ne nabora značk, ne njihove semantike. Lahko bi ga označili kot meta jezik za opisovanje označevalnih jezikov. Zagotavlja možnost definiranja značk in strukturne odvisnosti med njimi, nabora značk pri XML specifikaciji sploh ni. Kakršnakoli semantika je posledično odvisna od aplikacije, ki procesira ta XML dokument, ali od slogovnih datotek.

Validacijo XML izvedemo prek opisnih XML datotek, ki določajo semantiko in opisujejo zgradbo XML datoteke. Po teh navodilih potem XML orodja za validacijo preverijo XML.[7]

Nekaj bolj znanih XML shem:

- KML "Keyhole Markup Language" se uporablja pri geografskem prikazu podatkov
- XUL "XML User Interface Language" uporablja predvsem fundacija Mozilla za izgradnjo uporabniških vmesnikov
- RDF "Resource Description Framework" za opis meta podatkov
- Atom in RSS pri deljenju novic
- SVG opisuje strukturo vektorskih slik

2.4.4 XPath

Xpath jezik nam omogoča dostop do elementov in atributov v drevesu nekega dokumenta XML. Ime povzema po načinu iskanja po drevesih, saj se sprehaja po gradnikih dokumenta s pomočjo sintakse, ki ni v XML obliki.

Rezultat izraza Xpath je lahko nabor vozlišč, izolirana entiteta ali pa kakršnokoli zaporedje, opisano s sintakso, ki ga podpira podatkovni model. Izrazi Xpath ne operirajo na izpisani sintaksi dokumenta, ampak na samem podatkovnem modelu (logični strukturi), ki je opisan v Xquery in Xpath 2.0 podatkovnem modelu.

Glavni konstrukt sintakse je Xpath izraz, s katerim poiščemo objekt v dokumentu, nabor možnih rezultatov:

- neurejen seznam vozlišč brez dvojnikov
- logična vrednost
- zapis s plavajočo vejico
- niz znakov

Xpath 1.0 v programskem jeziku PHP uporablja knjižnica DOMXPath, ki nam omogoča sprehod po XML ali HTML (DOM) dokumentu z Xpath sintakso.

Primer uporabe xpath na HTML datoteki:

Koda 2.3 : HTML datoteka s seznamom povezav

```
<html>
  <body>
    <div id="center">
      <div class="menu">
  <ul class="links">
    <li class="pagerlink">
      <a class="actionlink" href="http://www.link.com/1">first</a>
    </li>
    <li class="pagerlink">
      <a class="actionlink" href="http://www.link.com/2">2</a>
```

```
11      </li>
12      <li class="pagerlink">
13        <a class="actionlink" href="http://www.link.com/3">3</a>
14      </li>
15      <li class="pagerlink">
16        <a class="actionlink" href="http://www.link.com/4">4</a>
17      </li>
18      <li class="pagerlink">
19        <a class="actionlink" href="http://www.google.com">last</a>
20      </li>
21    </ul>
22        </div>
23      <div id="footer"></div>
24    </body>
25  </html>
```

Primer dostopa do zadnje povezave v strukturi

Koda 2.4 : Najbolj osnoven xpath izraz ki ga lahko pridobimo že prek spletnega brskalnika Chrome

```
1  //*[@id="center"]/div[1]/ul/li[5]/a
```

Koda 2.5 : S pomočjo text() funkcije

```
1  //a[text()="last"]
```

Koda 2.6 : Grupiramo po vseh povezavah in najdemo zadnjo

```
1  (//a)[last()]
```

Koda 2.7 : S sprehodom po vseh povezavah

```
1  foreach ($xpath->query('//a') as $link) {
2    if ($link->getAttribute('class') == 'actionlink') {
3      echo $link->nodeValue . PHP_EOL;
4    }
5  }
```

2.4.5 Regularni izrazi

Podatke lahko iz besedila izločamo tudi z regularnimi izrazi, omogočajo nam primerjavo določenega dela besedila ali specifičnih ponavljajočih se vzorcev v podatkih. Podprti so v mnogih urejevalnikih besedila, operacijskih sistemih in višjih programskih jezikih. Čeprav nam lastnosti regularnih izrazov v večini primerov močno olajša delo pri tekstovnih podatkih, pa niso primerni za spletno luščenje. Pri omejevanju izrazov na vzorce v tekstu moramo biti ravno prav specifični, da ne izločimo podatkov, ki niso del vzorca, obenem pa paziti, da ne vključujemo odvečne vsebine. HTML strani se lahko spreminjajo (nekaterim spletnim stranem je to tudi taktika omejevanja pajkov), od nas pa je odvisno, ali bodo regularni izrazi znali izločiti pravilne podatke.

2.4.6 Curl

Curl ali "Client for URLs" je eno glavnih orodij pri spletnem luščenju, ker nam omogoča dostop ali prenos datotek prek URL naslovov.

Dandanes to orodje najdemo v mnogih programih, operacijskih sistemih (nekaterih distribucijah Linux in vseh Mac OS X operacijskih sistemih), televizijah, Blueray predvajalnikih, igrah, zunanjih programskih knjižnicah (libssh2), pametnih telefonih (iOS, WebOS), spletnih strežnikih, programskih jezikih prek knjižnic in sistemov za upravljanje z verzijami(git, Mercurial, ...)[8].

Curl orodje je sestavljeno iz dveh produktov:

- libcurl knjižnica
- curl

Libcurl knjižnica

Libcurl je enostaven in prenosljiv vmesnik do velikega nabora internetnih protokolov, podpira namreč:

- protokole FTP, FTPS, HTTP, HTTPS, IMAP, POP3, POP3S, SCP, SFTP, SMTP, TELNET, ...
- certifikate HTTPS
- HTTP POST in PUT
- nalaganje prek FTP
- nalaganje prek HTTP FORM
- uporabniške piškotke
- namestniške posredniške strežnike
- operacijske sisteme Solaris, NetBSD, FreeBSD, Linux, HPUX, Irix, Windows, Amiga, OS2, BeOS, Mac OS X, RISC OS, DOS, Symbian, Android, ...
- IPv6

Libcurl je v programskem jeziku PHP podprt že od verzije 4.0.2.

Koda 2.8 : Primer curl uporabe

```
1  <?php
2
3  $ch = curl_init('slashdot.org');
4
5  // lahko uporabimo POST, GET, PUT
6  curl_setopt($ch, CURLOPT_CUSTOMREQUEST,  'GET');
7
8  // podpira tudi uporabniske piskotke
9  curl_setopt($ch, CURLOPT_COOKIEJAR, 'cookie.txt');
10
11 // shranimo povratne informacije v spremenljivko
12 curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
13
14 // sledimo povezavam (vse preusmeritve)
15 curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
16
17 // ne potrebujemo glave HTTP povezave
18 curl_setopt($ch, CURLOPT_HEADER, 0);
```

```
19
20 // nastavimo svojega agenta
21 curl_setopt($ch, CURLOPT_USERAGENT, 'Googlebot/2.1 (+http://www.google.
       com/bot.html)');
22
23 // vsebina zahtevane strani
24 $content = curl_exec($ch);
25
26 // podrobne informacije o povezavi
27 $headers = curl_getinfo($ch);
28
29 curl_close($ch);
```

Curl

Orodje curl uporablja libcurl knjižnico, zato vsebuje identične specifikacije, uporablja se predvsem v pomoč pri programiranju ali sistemski administraciji, saj nima uporabniškega vmesnika in se enostavno uporablja v testnih ukaznih datotekah, kjer dostopa do knjižnice libcurl ni.

Koda 2.9 : Testiranje odzivnosti strani

```
1 curl -w '\nPreXfer time:t%{time_pretransfer}\nStartXfertime:t%{
      time_starttransfer}\nTotal time:t%{time_total}\nHTTP Code:%{
      http_code}\nNumber of redirections:%{num_redirects}\n' -o /dev/null
      -s slashdot.org
```

Koda 2.10 : Več informacij o strežniku

```
1 curl -I slashdot.org
```

2.4.7 Posredniški strežniki

V praksi uporabljamo posredniške strežnike kot posrednike med odjemalci in strežniki. Zahtevo po datoteki, ki poteka od odjemalca do končnega strežnika, prestreže namestniški strežnik. Ta poskuša tej zahtevi zadostiti sam, če pa ne

najde specifične informacije na svojem sistemu, posreduje zahtevo na ciljni strežnik.

Pogosto se ga uporablja kot sistem za predpomnjenje spletnih strani, kar drastično pohitri uporabniško izkušnjo odjemalcem. Deluje namreč na principu hranjenja podatkov, do katerih se pogosto dostopa, s tem se v omrežju zagotavlja višjo kakovost storitve, saj nam minimizira porabo pasovne širine, s predpomnenjem pa krajša odzivni čas dostopa do podatkov. Lokalna kopija pa nam lahko služi kot dodatna razpoložljivost vsebine, saj jo lahko postrežemo ob morebitnem izpadu strežnika z originalno vsebino.

V izobraževalnih ustanovah ali določenih podjetjih s specifično politiko dostopa lahko delujejo tudi kot posredniški strežniki za filtriranje vsebine, podpirajo avtentikacijo uporabnikov, hranijo dnevnike z zapisi povezav in prometa ter preverjajo vsebino z dodatnimi antivirusnimi programi.

Vsebino lahko preverjamo na več načinov:

- prek črnega seznama URL naslovov in DNS zapisov
- preverjanja URL naslovov prek regularnega izraza
- filtriranja MIME podatka
- pregledovanja podatkov po ključnih besedah

V primerih pregledovanja podatkov po ključnih besedah in črnega seznama URL naslovov načeloma uporabljamo seznam nekega zunanjega ponudnika, ki ga lahko tudi sami dopolnemo glede na notranjo politiko dostopa naše organizacije.

Skrbniki posredniških strežnikov lahko pregledujejo tudi šifriran promet prek SSL protokola s tako imenovano "man in the middle" tehniko napada, a le z namestitvijo certifikatov (zgeneriranih na posredniškem strežniku) v odjemalčev spletni brskalnik in preverjanjem veljavnosti certifikata oddaljenega strežnika.

Uporabniki, ki želijo obiti spletno filtriranje na lokalnih posredniških strežnikih bodo posegli po anonimnih različicah s HTTPS podporo. Ta

posredniški strežnik nato šifrira celoten promet prek SSL protokola, filtrirni strežnik pa tega kot zgoraj omenjeno ne more filtrirati, razen če ima dostop do certifikatov anonimnega posredniškega strežnika.

Obratni posredniški strežniki

Obratni posredniški strežniki pa so za odjemalca nevidni, saj prejmejo zahtevo in jo naprej posredujejo spletnim strežnikom v notranji mreži, promet pa posredujejo le točno določenim strežnikom in ne vsem odjemalcem. Tako se obratne posredniške strežnike v praksi uporablja kot strežnike za izenačevanje obremenitve (load balancing) – tu se morajo vsi zahtevki prepisati oz. nasloviti na točno določeni notranji strežnik, ves izhodni promet, pa je iz stališča notranje arhitekture ciljnemu uporabniku skrit. Lahko jih uporabimo tudi kot dodatni varnostni sloj pred notranjo infrastrukturo. Tu se lahko izvaja tudi šifriranje prometa, saj s tem omejimo funkcionalnost na specifični streznik, ki je lahko tudi temu primerno strojno podprt. Nenazadnje pa pridejo prav pri optimizaciji prometa kot strežniki statičnih vsebin ali stiskanju podatkov in s tem zmanjšanju porabe pasovne širine.

Anonimni posredniški strežniki

Pri spletnem luščenju pogosto pridejo prav anonimni posredniški strežniki ravno zaradi velikega hkratnega števila zahtevkov na ciljno spletno stran. Pomagajo nam pri možni razdelitvi tega prometa na različne strežnike, spletne strani v določenih industrijah (predvsem v turistični) namreč preverjajo sumljive, intervalno točne povezave na njihovo vsebino in jo dodatno zaščitijo.

Na internetu obstaja ogromno strani, ki ponujajo brezplačen dostop do vseh vrst anonimnih posredniških strežnikov. Z dostopom do strani, kjer se prenašajo naši osebni podatki, pa moramo biti previdni in zaupati skrbnikom sistema ali uporabljati šifrirano povezavo. Pri luščenju podatkov se lahko poslužimo tudi sistema Tor (glej 2.6.1).

2.4.8 Symfony

Celotno diplomsko nalogo smo si zamislili na ogrodju Symfony2[13], predvsem zaradi enostavnega dela z zagonskimi skriptami, bazo, predlogami in ostalimi orodji, s katerimi smo luščili specifične spletne strani.

Symfony2 je PHP ogrodje za implementacijo spletnih aplikacij, izdano pod licenco MIT v odprtokodni skupnosti že leta 2005. Celotno načrtovanje in razvoj je pod taktirko francoske programerske agencije Sensio Labs, open-source miselnost pa je le še bolj povezala celotno skupnost prek poštnih seznamov, internetnega klepeta IRC in ostalih drugih internetnih podjetij, ki so vložila znanje v ta projekt.

Manjka tudi ne odmevnih projektov, kot so TED, Delicious, Yahoo! Bookmarks, Yahoo! Answers, OpenSky, DailyMotion, wetter.com, lockerz.com, YouPorn, prihajajoča Drupal verzija 8, EZPublish, PHPUnit, phpBB itn.[9]

Največje prednosti ogrodja:

- uporaba programskih paketov Bundle – preprosta priključitev paketov v sam sistem, na internetu najdemo ogromno zbirko teh paketov prav zaradi velike skupnosti in hitrega porasta projektov
- enostavna namestitev prek različnih sistemov, bodisi je to git, svn ali composer
- sledi standardom razvijanja kode PSR-0
- omgoča preprosto uporabo vizualnih gradnikov prav prek namestitvenega orodja composer in programskih paketov
- objektno usmerjen pristop k programiranju
- uporablja že uveljavljene in dobro testirane knjižnice, kot so Monolog, Assetic, Doctrine, Propel, ...
- odlična dokumentacija, prosto dostopna knjiga, napisana s strani samih razvijalcev

- močna podpora skriptam v vmesniku z ukazno vrstico – tu lahko že ob samem pričetku projekta generiramo lupino lastnega programskega paketa v MVC arhitekturi, postavimo lahko predloge, skripte, strukturo baze, CRUD operacije nad bazo v povezavi s predlogami, upravljamo s predpomnilnikom in nameščamo sredstva – "assets"

2.4.9 Orodja za avtomatizacijo testiranja

Pri spletnem luščenju si lahko poleg knjižnice Xpath in regularnih izrazov pomagamo z "web acceptance" orodji, kot so Selenium[14], Sahi[16], Watir[17] in mnogimi drugimi.

Osnovna ideja prihaja iz specifičnega načina testiranja že implementiranih produktov. Funkcionalno testiranje je testiranje produkta v fazi zagotavljanja kvalitete, pri čemer se uporablja princip črne skatle. Pri tem se ne obremenjujemo s samim načinom implementacije in zgradbe programa; ne sprašujemo se, kako deluje sama zasnova, ampak kako jo uporabljamo in kakšen je rezultat. Poznamo vhod, pričakujemo pa določen rezultat, vmesne faze nas ne zanimajo. Vhodni podatki temeljijo na podlagi specifikacije o zahtevah sistema. Takšno testiranje pa na spletu vključuje klikanje po straneh, vnašanje teksta v vnosna polja, identifikacijo spletnega uporabnika, interakcijo z gradniki v brskalniku, ...

Kar bi razvijalcu vzelo veliko časa, rešijo orodja za avtomatizacijo testiranja spletnih aplikacij. Ena glavnih so Sahi, Selenium in Watir.

Sahi

Avtomatizacijsko orodje Sahi deluje na principu posredniškega strežnika. Ko zaženemo aplikacijo, se integrirani strežnik zažene na določenih vratih. Naslov tega strežnika in številko vrat pa uporabimo v svojem brskalniku, v katerem želimo testirati našo aplikacijo. To omogoča Sahi orodju, da preveri vsako zahtevo, ki se aktivira v brskalniku in ukrepa na podlagi našega testnega načrta. Spremenjena zahteva se v naš brskalnik vrne z vrinjeno javascript kodo, katera potem upravlja z delovanjem brskalnika.

Ker moramo za testiranje strani z AJAX zahtevki in vsebino, obogateno z javascript funkcionalnostjo simulirati uporabnika v spletnem brskalniku (klik na gumb, vnos teksta v polje, drag and drop, ...), je javascript dobra izbira, saj vsebuje programski vmesnik za vso potrebno interakcijo. Tako ne uporablja Xpath orodja, kot veliko drugih rešitev, ampak ovojnico okoli javascript DOM knjižnice. S temi vrinjenimi javascript ukazi Sahi nadzira vse prikazane elemente v brskalniku. Ker je v ozadju posredniški strežnik, lahko tudi javascript posreduje podatke nazaj, pristop, ki se uporablja pri testiranju nalaganja datotek.

Slika 2.1: Shema delovanja Sahi orodja

Ker uporablja svojo ovojnico okoli javascript programskega vmesnika, lahko Sahi uporabljamo z večino novejšimi brskalniki, ne glede na operacijski sistem. Podpira pa še:

- HTTP avtentikacijo
- HTTPS/SSL strani
- pojavna okna in okvirje "iframe"

Phantom.js

Phantom.js[18] je spletni brskalnik brez grafičnega vmesnika, ki temelji na tehnologiji Webkit. Webkit je ogrodje za prikaz gradnikov v spletnih brskalnikih, razvila ga je odprto-kodna skupnost KDE in izhaja iz ogrodja KHTML, uporabljata pa ga še brskalnika Google Chrome in Safari.

Poleg tega vsebuje tudi programski vmesnik za jezik Javascript, zato se uporablja predvsem pri testiranju funkcionalnosti spletnih strani, vedenjsko usmerjenem razvoju BDD, shranjevanju zaslonskih slik nekega procesa (vključno s canvas elementom in SVG datotekami), dostopu do programskega vmesnika DOM in Jquery ter performančnih analizah. Njegova največja prednost je predvsem uporaba na strežnikih, kjer načeloma ne uporabljamo grafičnega vmesnika.

Mink

Še večjo transparentnost in nivo abstrakcije smo zagotovili s knjižnico Mink, katero lahko preprosto namestimo kot programski paket dela Symfony2 prek sistema za upravljanje s paketi Composer.

S to knjižnico lahko preprosto pišemo lastne scenarije za testiranje aplikacij, neodvisno od uporabljenega orodja za avtomatizacijo. Podpira namreč Selenium, Selenium2[15], Goutte[19], Sahi in Zombie.js[20] emulatorje brskalnikov. Z isto kodo lahko z Mink orodjem nadziramo enega izmed teh orodij.

Glede na naše zahteve in značilnosti spletne aplikacije, ki jo testiramo, imamo pri Mink na izbiro:

- krmilniki spletnih brskalnikov
- emulatorji brskalnikov brez vmesnika "headless browser emulators"

Emulatorji, kot je Goutte, so bolj preprosta implementacija specifikacije HTTP, ki pošiljajo zahtevo na spletno aplikacijo, potem pa vsebino odziva razčlenijo in posredujejo aplikaciji na višji nivo. So načeloma hitrejši od krmilnikov, saj jih lahko poganjamo tudi na strežnikih, kjer nimamo urejenega grafičnega vmesnika, zato ne potrebujemo spletnega brskalnika. Ne podpirajo pa AJAX in javascript klicev, zato z njimi ne moremo testirati spletnih strani z veliko mero interaktivnosti.

Krmilniki pa sodelujejo s spletnim brskalnikom, da bi v čim večji meri simulirali uporabnika, negativne posledice tega pa so dodatne nastavitve

sistema in nujen grafični vmesnik. Pri tem testiranju moramo upoštevati tudi čas zagona brskalnika in čiščenje nastavitev ob vsakem novem testiranju.

Pri uporabi Mink orodja lahko enostavno upravljamo s katerim koli brskalnikom (ali pa lastnim orodjem, ki simulira brskalnik v obliki HTTP zahtev) ter med njim izbiramo brez kakršnekoli spremembe v programski kodi testnega scenarija.

Koda 2.11 : Primer emulacije uporabnika pri iskanju na strani Google

```
1  // izbira gonilnika za krmilnik
2  $client = new \Selenium\Client($host, $port);
3
4  // izbira strani in brskalnika
5  $driver = new \Behat\Mink\Driver\SeleniumDriver(
6      'firefox', 'http://www.google.com', $client
7  );
8
9  // kreiramo sejo
10 $session = new \Behat\Mink\Session($driver);
11
12 // zacetek seje
13 $session->start();
14
15 // vnosno polje za iskanje
16 $session
17   ->getPage()
18   ->find('xpath', '//input[@name="q"]')
19   ->setValue("web scraping");
20
21 // najdemo gumb "Im feeling lucky" in kliknemo nanj
22 $session
23   ->getPage()
24   ->find('xpath', '//input[@name="btnI"]')
25   ->click();
```

2.5 Načini blokiranja webscrapinga

Kot uredniki neke spletne strani si vedno želimo čim večje prepoznavnosti in s tem tudi večjega obiska strani. V tej nameri se poslužimo optimizacije spletnih strani ali tehnike povezovanja z drugimi stranmi. Če pa ponujamo

specifično funkcionalnost in s tem unikatne podatke, lahko do te vsebine dostopajo tudi avtomatizirana orodja. Blokiranje teh pajkov pa pogosto celo škoduje, kot koristi. Pri razčljenjevanju s pajki, kjer je govora večinoma o spletnih iskalnikih, so tehnike razmeroma preproste, pri specifičnih napadih, pa se moramo poslužiti naprednejših tehnik.

Pajke lahko omejujemo:

- z meta značkami v glavi HTML datotek
- z datoteko robots.txt
- z blokiranjem po IP naslovih
- na podlagi uporabniškega agenta
- z omejevanjem zahtevkov
- spreminjanjem razporeditve elementov na strani in različne uporabe tehnik prikaza vsebine
- uporabo AJAX tehnologije
- blokiranjem s CAPTCHA
- s komercialnimi rešitvami

Robots

Spletni pajki se ločujejo od razčlenjevalnikov po tem, da sledijo povezavam po straneh in hranijo meta podatke o strani in ne dejanskih informacij ter vsebine. Večina spletnih iskalnikov uporablja takšen način iskanja po internetu, zato so uvedli "Robots Exclusion" ali "robots.txt" protokol. Ta nam kot skrbniku sistema omogoča omejevanje dostopa teh pajkov na točno določeno stran s posebno datoteko robots.txt, ki jo postavimo v korenski imenik spletne strani. Vsebuje lahko seznam vseh podstrani, ki jih lahko ali ne sme obiskati in

zakasnitev pri dostopu do te strani. Vse te nastavitve pa lahko specificiramo ločeno za vsakega pajka, saj se identifikacija vrši na nivoju uporabniškega agenta.

Strani lahko omejujemo tudi prek značke <meta> v glavi HTML strani, standard, ki so ga sprejeli že v HTML 4.01 verziji. Z vrednostmi atributa CONTENT lahko specificiramo, ali pajek lahko sledi povezavam na strani (pomembno je vedeti, da do iste povezave lahko dostopa prek druge podstrani, ki nima te značke v glavi) in ali mu pustimo stran indeksirati.

Koda 2.12 : Primer značke <meta>v glavi strani

```
<html>
<head>
<title>...</title>
<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
</head>
```

Koda 2.13 : Primer robots.txt za neobstoječi pajek

```
User-agent: HackerBot
Allow: /
Disallow: /admin/
Disallow: /tmp/
Allow: /tmp/README.rd
```

Zavedati pa se moramo, da je ta konvencija s strani avtomatiziranih orodij opcijska, zato se večina zlonamernih pajkov ne poslužuje datoteke robots.txt ali meta značke v glavi.

Blokiranje IP naslovov

Vsak skrbnik spletne strani bi se moral posvečati tudi strežniškim dnevnikom, kjer je možno razbrati tudi vzorce zahtevkov iz določenih številk IP. Velikokrat se površno implementirani razčlenjevalniki ne posvečajo zakasnitvam in opravljajo svoje delo z največjo možno hitrostjo, to pa je enostavno opazno na strežnikih. Ob naprednejši uporabi posredniških strežnikov, pa ta tehnika

kmalu postane neuporabna. Isti pajek lahko namreč dostopa iz različnih naslovov, vzdrževanje seznama blokiranih IP-jev, pa postane zamudno.

Spreminjanje razporeditve elementov in izgleda

Nekatere strani uberejo drugačno pot in se posvetijo predvsem spreminjanju videza celnega dela sistema (Easyjet redno spreminja strukturo, po novem, pa so dodali še sistem CAPTCHA). S tem ciljajo predvsem na razčlenjevalnike, ki se osredotočijo prav na njihove podatke, a s tem minimalno pridobijo na času, saj je moč po implementaciji celotnega razčlenjevalnika hitro adaptirati algoritem na male spremembe.

Obstajajo tudi tehnike prikaza tekstovne vsebine v slikovnih formatih, ampak s tem povečamo obremenitev na strežniku pri generiranju te vsebine; onemogočimo tudi dostop omejenim uporabnikom (bralnik za slepe), ter s tem posledično tudi zmanjšamo lastno vidnost na spletnih iskalnikih.

Druge varijante so tudi uporaba AJAX tehnologije, saj s tem onemogočijo dostop emulatorjem brskalnikov, ki dostopajo do podatkov v sistemu brez vmesnika. S pomočjo CSS stilov lahko tudi skrivamo določene dele strani, ki niso pomembne navadnemu uporabniku. Tako lahko dodamo povezavo, ki je skrita očem uporabnikom, a ji pajki ali razčlenjevalniki vseeno sledijo, s tem pa posledično dobimo IP avtomatiziranega orodja. Ker pa pri pisanju programa za razčlenjevanje z lahkoto preverjamo, ali je določena povezava skrita, obstaja tudi tehnika prikazovanja povezav v isti barvi, kot je ozadje strani.

Honeypot

Najbolj uporabna rešitev pa pride v upoštev s kombinacijo omenjenih tehnik. To s pridom uporablja sistem limanica ali "honeypot". Na spletni strani urednik postavi vsebino in jo skrije (npr. povezava, skrita s stilom CSS ali barvo), to pa z robots.txt datoteko zaklene. Kakršnakoli zahteva, ki bo sledila tej povezavi, bo v večini primerov avtomatizirana. Skrbnik sistema lahko tako v veliki meri pridobi IP naslove pajkov in trenutno zaklene dostop do strani

za trenutnega uporabnika. IP naslovov ni smiselno zaklepati za nedoločen čas prav zaradi uporabe posredniških strežnikov.

Ko je napadalec identificiran lahko idejo še razširimo z uporabo "teergrubing" ali "tarpitting" že uveljavljene tehnike pri boju proti vsiljivi pošti. Ko se pajek poveže na stran, poskušamo njegovo zvezo čim dlje ohraniti na strežniku in ga s tem umetno upočasnimo.

Celoten splet je zasnovan na tak način, da se prikazani podatki zelo težko zaklenejo za specifične uporabnike in obenem ostajajo odprti za iskalnike. V tem primeru je strokovno mnenje v skupnosti v prid razvijanju lastnega programskega vmesnika API do podatkov ali funkcionalnosti.

2.6 Anonimizacija

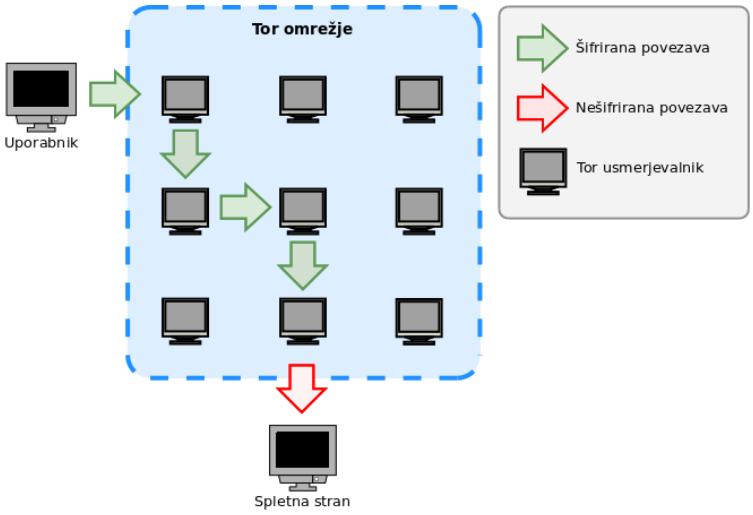
2.6.1 Tor

Tor je bil v začetku zasnovan kot projekt za usmerjanje prometa ameriške mornarice in šifriranja komunikacije ostalih državnih organov. Dandanes pa ga uporabljajo novinarji, aktivisti, vojska, vlade, hekerji, itd.

Deluje na osnovi strežnika in odjemalca. Ko pošljemo zahtevo do ciljne spletne strani, se na lokalnem računalniku prek aplikacije ustvari šifrirana povezava do naključno izbranega strežnika – Onion usmerjevalnika. Ta strežnik, kot prva naprava v verigi Tor posredniških strežnikov, pozna naš naslov in šifrirano povezavo. Ko sprejme podatke, pošlje zahtevo naslednjemu usmerjevalniku – ta pa pozna le usmerjevalnik, ki mu je to zahtevo poslal. Tako se zahteva posreduje prek večih strežnikov, nobenemu v vrsti, pa ni znana celotna pot zahteve, le odjemalcu, ki jo je sprožil.

Odjemalec ima dostop do baze, ki vsebuje vse usmerjevalnike. Tako ima možnost izbire med točno načrtovano potjo po naključnih ali po specifičnih usmerjevalnikih. Ko sproži zahtevo, jo mora podpisati z vsemi javnimi ključi sodelujočih strežnikov v obratnem vrstnem redu, začenši z javnim ključem zadnjega (izhodnega) usmerjevalnika. Na izhodnem strežniku se celotna zahteva odšifrira z zadnjim javnim ključem in z njegovim IP naslovom pošlje

na ciljni strežnik z zahtevano spletno stranjo.



Slika 2.2: Delovanje sistema Tor

Slabosti Tor omrežja so počasno delovanje, vsaka zahteva mora namreč slediti večim usmerjevalnikom. Podatki se tudi opcijsko šifrirajo, kar dodatno poveča čas poizvedbe. Sama odločitev šifriranja na strani odjemalca, pa predstavlja eno večjih slabosti omrežja Tor. Izhodno vozlišče lahko predstavlja kakršnakoli naprava, ki je priključena na internet. To lahko predstavlja naš domači računalnik, strežnik, nadzorovan s strani državnega organa ali pa v najslabšem primeru, osebe, ki spremlja promet skozi to vozlišče. Na tem vozlišču se lahko uporabi tehniko vrinjenega napadalca, ampak le v primeru, ko je povezava med tem usmerjevalnikom in ciljno internetno stranjo nešifrirana.

2.6.2 Navidezna zasebna omrežja

Navidezno zasebno omrezje VPN je skupina računalnikov, organizirana v lastno omrežje z omejenim dostopom v okviru javne komunikacijske infrastrukture. Poslovne organizacije uporabljajo VPN omrežja pri združevanju informacijskih sistemov v celoto ter omogočajo dostop do tega omrežja posameznikom, ki se ne morejo priključiti na fizično omrežje.

VPN omrežja nam pri razčlenjevanju pomagajo z zmanjševanjem možnosti morebitne identifikacije naših zahtevkov, saj se s povezavo prek navideznega zasebnega omrežja spremeni naslov IP. V primerjavi s posredniškimi strežniki so VPN omrežja dosti hitrejša, odzivna, ponujajo večjo varnost, s tem pa povzročajo tudi večje stroške, saj je dostop plačljiv. Tu ne obstaja možnost vrinjenega napadalca (iz stališča omrežja VPN), je pa avtomatizirana uporaba teh omrežij dosti bolj omejena, kot pri posredniških strežnikih.

Pri razčlenjevanju nas ne zanima toliko anonimnost in varnost podatkov, ampak možnost skrivanja zahtevkov za drugim IP naslovom. Ob tej uporabi se pogosto zanašamo na vmesne strežnike, ki ta promet lahko pregledujejo. Dokler prenašamo prosto dostopne vsebine, to ne povzroča velikih preglavic, za dodatno varnost, pa moramo poseči po plačljivih storitvah (s hitrostjo omrežja narašča tudi cena).

2.7 Spletno luščenje in zakonodaja

Uporaba spletnega luščenja se iz dneva v dan povečuje, saj veliko podjetij ponuja podatke iz raznih prosto dostopnih forumov, socialnih omrežij, oglasnih strani ali blogerskih platform. Podjetja najemajo storitve izvajalcev socialnih iskanj – zgodovino novega zaposlenega, njegovo zdravstveno stanje in možno kartoteko kaznivih dejanj. S pridom izkoriščajo pomanjkanje informiranosti uporabnikov, ki prosto objavljajo svoje podatke na teh platformah.

Pri tem se lahko vprašamo, ali je to početje legalno? Ali lahko strani, do katere dostopamo uporabijo legalne sankcije proti nam?

Kot primer lahko vzamemo Googlov algoritem, njegovi pajki dnevno obiščejo milijone strani in nam postrežejo z informacijami na njihovi strani.

Tako je spletno luščenje v sivem območju zakonodaje ne samo pri nas, ampak tudi v tujini. Kakšna je razlika med dostopom do spletne aplikacije s strani človeka, ki stran prebira ali kopira vsebino v tekstovni urejevalnik in med avtomatiziranim orodjem, ki počne isto stvar, le v večjem obsegu?

Najbolj znan primer je tožba med podjetjema eBay in Biddder's Edge leta 2000. Bidder's Edge podjetje je nudilo uporabnikom možnost iskanja po agregiranih rezultatih avkcij, ki so trenutno potekale na spletni strani eBay. Te podatke je luščilo brez kakršnih koli omejitev pri številu in časovnem zamiku dostopov. Pri Ebayu so to početje obravnavali kot prekoračenje dostopa do imovine ali "trespass to chattel". S tem se je dodatno povečala obremenitev strežnikov, iskalnik, pa naj ne bi v nekaterih primerih prikazoval vseh zadetkov. Ebay naj bi s tem početjem izgubil ugled, BE, pa se finančno okoristil.

Dokazano je BE povzročil 1.53% obremenitve vseh Ebay strežnikov, več takšnih robotov, ki ne spoštujejo datoteke robots.txt, pa bi posledično še povišalo obremenitev ali celo povzročilo izpad sistema.[11]

V Sloveniji takšnih tožb še ni bilo, saj tudi pri nas zakonodaja še ni zadosti specifična. 221. člen uradnega lista republike Slovenije, pa navaja[29]:

> **(1)** Kdor vdre v informacijski sistem ali kdor neupravičeno prestreže podatek ob nejavnem prenosu v informacijski sistem ali iz njega, se kaznuje z zaporom do enega leta.
>
> **(2)** Kdor podatke v informacijskem sistemu neupravičeno uporabi, spremeni, preslika, prenaša, uniči ali v informacijski sistem neupravičeno vnese kakšen podatek, ovira prenos podatkov ali delovanje informacijskega sistema, se kaznuje za zaporom do dveh let.
>
> **(3)** Poskus dejanja iz prejšnjega odstavka je kazniv.
>
> **(4)** Če je z dejanjem iz drugega odstavka tega člena povzročena velika škoda, se storilec kaznuje z zaporom od treh mesecev do petih let.

Tu meja med nezakonitim prenosom podatkov ali vdorom v informacijski sistem in spletnim luščenjem ni jasno definirana. Pri našem početju je smiselno:[10]

- preveriti pravila uporabe spletne strani
- upoštevati robots.txt datoteko
- kontaktirati skrbnike spletne strani in jih seznaniti z našimi nameni
- če prikazujemo podatke, podati povezavo na izvorno stran
- ne dostopati do podatkov na agresiven način; če spletna stran gostuje na počasnem strežniku, lahko pomotoma povzročimo DDoS napad, tukaj spet pride v upoštev datoteka robots.txt
- se omejiti samo na prosto dostopne strani in ne luščiti vsebin, ki so dostopne registriranim in prijavljenim uporabnikom

Poglavje 3

Analiza obstoječih rešitev in predlog lastne implementacije

3.1 Scraperwiki.com

Spletna rešitev za gostovanje uporabniških luščilnih skript in algoritmov podatkovnega rudarjenja v PHP, Python in Ruby programskih jezikih. Tu obstaja tudi seznam že naloženih algoritmov s strani registriranih uporabnikov, prek plačljivega dostopa, pa dobimo tudi možnost zasebnih projektov, dodajanja večih uporabnikov na račun in neomejenega izvajanja opravil v ozadju. Programski vmesnik API ponuja možnost uporabe vgrajenih PHP funkcij regularnih izrazov in zunanjih knjižnic (PHP Simple HTML DOM Parser). Podatke lahko izvozimo v več različnih formatov – XML, csv, MySQL in Sqlite.

Ponudbo razširi še možnost najema ekipe za individualno luščenje ali prenos podatkov iz drugih medijev, kot so PDF dokumenti, skenirani arhivi v obliki slikovnih datotek ali binarni podatki, kjer je potrebna predvsem specifična rešitev.

3.2 Vstavek za luščenje

Scraper dodatek[28] lahko enostavno uporabljamo kar v Chrome brskalniku. Zahtevan podatek v deloma strukturirani obliki označimo in prek menija izvedemo luščenje strani. Na uporabniku prijazen način lahko izvajamo osnovno luščenje posamezne strani, primeren je za dostop do podatkov, kjer potrebujemo veliko uporabniške interakcije, da dostopamo do iskanih rezultatov. Ker pri uporabi ni potrebno znanje programiranja (čeprav podpira Xpath in jQuery notacijo), je idealen pripomoček za osnovnega uporabnika.

3.3 Webscraping.com

Podjetje, ki nudi specifične rešitve na zahtevo v rangu med 150$ in 2000$, podpira tudi AJAX luščenje ter razbijanje sistema captcha. Ponuja tudi široko paleto brezplačnih podatkovnih zbirk, kot so seznami mest določenih drzav, kode regij, zbirko citatov in različnih pijač, podatke o valutah itd.

3.4 Visual Web Ripper

Namizno orodje za podatkovno rudarjenje Visual Web Ripper[21] na operacijskem sistemu Windows podpira Xpath sintakso in regularne izraze. S pomočjo naprednega vmesnika lahko izpolnjujemo spletne obrazce in dostopamo do podatkov, ki so prikazani s pomočjo AJAX tehnologije. Omogoča tudi po meri nastavitev uporabniškega agenta na vsako sejo, ter hranjenje pogostih scenarijev v predloge za večkratno uporabo. Uporabnik lahko celo uvozi bazo podatkov za uporabo v spletnih vnosnih formah, izhodne podatke pa shrani v obliki CSV, Excel, XML, SQL Server, MySQL, Oracle in OleDB datotek. Dodaten nivo anonimizacije je zagotovljen tudi prek lastnega nabora posredniških strežnikov ali storitve Private Proxy Switch. Ostale aplikacije v tej kategoriji:

- WebSundev Extractor[22]
- Easy Web Extractor[23]
- Web Content Extractor[24]
- Mozenda[25]
- OutWit Hub[26]
- Helium Scraper[27]

3.5 Predlog rešitve

Brezplačne spletne storitve in namizni programi so izvrstne rešitve za novinarje, študente in spletne trgovce. Z njimi lahko bodisi dostopamo do znanstvenih podatkov na straneh univerz, baze produktov v spletni trgovini ali prosto dostopnih podatkov na podstraneh vladnega portala. Vendar pa se je s številom informacij in frekvenco osveževanja kmalu potrebno usmeriti k lastni rešitvi. Večina obstoječih orodij, ki ponujajo ekvivalentno funkcionalnost, kot smo si jo zamislili v diplomski nalogi, pa je na voljo v obliki komercialnih programov. Največji prednosti lastne rešitve sta paralelizem izvajanja zahtevkov ter možnost luščenja na oddaljenih strežnikih. Alternativo nam lahko ponudijo le ozko usmerjene rešitve s strani ponudnikov podatkovnega rudarjenja. Glede na identificirane pomanjkljivosti v nadaljevanju predlagamo lastno rešitev na podlagi tehnologije, opisane v prejšnjem poglavju. Prioriteta nam je predvsem možnost izvajanja kode na oddaljenih strežnikih, enostavno dodajanje luščilcev novih virov podatkov ter anonimizacija zahtev.

Poglavje 4

Zasnova projekta

Celoten projekt smo načrtovali na podlagi ogrodja Symfony2, saj nudi izvrstno podporo lupinskim skriptam v programskem jeziku PHP. Zaradi izjemno velike skupnosti in možnosti razširitve tega ogrodja, pa smo uporabili že razvite funkcionalnosti v obliki paketov, imenovanih "bundles". V namen diplomske naloge pa smo razvili še pakete, namenjene spletnem luščenju specifičnih strani.

4.1 Ideja

Idejo spletnega luščenja smo snovali na dveh različnih pristopih. Osnovna tehnika večinoma uporablja standardne knjižnice v programskem jeziku PHP. Prikazati smo hoteli, da je možno že z osnovno distribucijo PHP v operacijskem sistemu Linux luščiti podatke na spletnih straneh. Izbrali smo si portal nepremicnine.net, predvsem zaradi tega, ker se HTML koda ne podreja standardom. Iskanje po strani je enostavno, saj grafični vmesnik ne uporablja tehnologije AJAX, zato smo z orodji Xpath in Simple HTML Dom dostopali do gradnikov strani in s pomočjo njihovega nabora te vrednosti uporabljali v naslovu spletne strani. Tako za primer regij na prvi strani pregledamo vse gradnike tipa input po določenem atributu in jih uporabimo v naslovu. Te gradnike pridobivamo z obema orodjema, pri tem se pokaže popularnost

orodja Xpath, saj njegova sintaksa deluje kot osnova mnogim drugim orodjem, prehod na Simple HTML Dom tako ni zahteval večjega truda.

Pri vsaki regiji smo morali na naslednji strani zbrati tudi občino, te smo na podoben način zbrali iz HTML kode. Premikanje po strani med iskalnimi formami ni povzročalo težav, saj se podatki prenašajo prek metode GET, za razliko od drugih strani, ki jih posredujejo prek POST. V tem primeru bi lahko še naprej uporabljali orodje curl, saj omogoča tudi POST podatkov na ustrezno lokacijo.

Ko smo zahtevane podatke prenesli prek HTTP protokola in ustrezno zluščili, pa smo jih shranili v podatkovno bazo Mysql. Symfony uporablja knjižnico Doctrine, ki deluje kot Object Relational Mapper. S pomočjo ukaznih orodij smo kreirali entitete, ki hranijo oglase in jih preko generiranega razreda vnašali v podatkovno bazo. To nam je omogočilo preprost, učinkovit in hiter razvoj administracijskega vmesnika. Delovanje osnovnega luščenja prikazuje Slika 4.1.

Pri naprednih tehnikah luščenja se nismo toliko posvetili samemu luščenju in dostopu do gradnikov dokumenta HTML, saj smo to preučili v osnovnem pregledu. Tu nam je bila pomembna predvsem emulacija končnega uporabnika kot tehnika anonimizacije avtomatiziranega orodja. Izbrali smo si bolj kompleksne spletne strani z AJAX tehnologijo in veliko uporabniške interakcije. Na obeh straneh, tako kayak.com, kot booking.com, bi lahko generirali del iskalnega zahtevka prek naslova URL, ampak smo zaradi uporabniške emulacije uporabili orodje za avtomatizacijo in testiranje spletnih strani Mink. Paziti smo morali predvsem na morebitno preverjanje direktnih dostopov na ciljnih strežnikih, saj se v primeru spletnega luščenja lažje zasledi avtomatizirane dostope.

Sam dostop do podatkov se tu loči na dva dela, najprej moramo simulirati dejansko iskanje na strani. To delovanje ni v celoti avtomatizirano, saj razvijamo luščilec in ne spletnega pajka. Razlika je očitna, saj smo se v primeru strani kayak in booking opredelili na točno določene gradnike na strani in temu primerno implementirali luščilec. Ko zaključimo z iskanjem,

moramo na rezultate zaradi postopnega prikaza uporabniku počakati, dokler se ne zaključi določena akcija. Ponavadi se na zaslonu pojavi opozorilo, da je iskanje zaključeno. Na strani kayak.com pa se med iskanjem izpisuje drsnik z napredkom, ko se gradnik skrije, vemo, da je iskanje zaključeno. Kar smo se naučili pri osnovnem luščenju smo morali uporabiti tudi tukaj, saj se v drugem delu iskanja spet znajdemo pri luščenju rezultatov na enak način kot na strani nepremicnine.net. Ker knjižnica Mink omogoča Xpath sintakso, nam ta del ni bil tehnično zahteven. Povezava Mink knjižnice z orodjem Sahi in Tor omrežjem je prikazana na Sliki 4.2

Ker se te akcije izvajajo v ozadju v obliki prikritih procesov, potrebujemo čelni del sistema, ki bi te akcije nadzoroval. Tako smo za napredno luščenje implementirali tudi časovni razporejevalnik akcij z administrativnim vmesnikom. Omogoča nam vnos iskanj v bazo, po katerih bi procesi v ozadju vršili luščenja podatkov na straneh, saj nam s stališča uporabnosti dosti bolj poenostavi upravljanje s sistemom. Lupinska orodja so lahko avtomatizirana s pomočjo orodij crontab ali v procesih screen.

4.2 Anonimizacija

Ker smo želeli biti pri našem delovanju čimmanj opazni, smo se poskušali povezati na spletne strani prek posredniških strežnikov. Javnih ter brezplačnih je na internetu ogromno. Slaba stran uporabe teh strežnikov je pomanjkljiva varnost podatkov, saj nihče ne preprečuje administratorjem man-in-the-middle napada. Naš promet in podatki pa nam niso ključnega pomena, saj se ne prijavljamo v zaščiten del z uporabniškim imenom in geslom. Seznam posredniških strežnikov dobimo kar z luščenjem spletne strani cool-proxy.net, kjer je redno osvežen seznam prosto dostopnih strežnikov iz celega sveta. Te strežnike zaradi pogostih izpadov tudi mi redno osvežujemo, mehanizem smo implementirali kot lupinsko skripto, ki jo zaganjamo na določen interval, najraje prek sistema crontab.

Napredno luščenje pa smo namesto prostih posredniških strežnikov uporabl-

jali SOCKS posredniške strežnike v omrežju Tor. Že samo Sahi orodje deluje na principu posredniškega strežnika, ima pa dodatno možnost nastavljanja prehoda prometa prek zunanjega posredniškega strežnika. Na lokalnem računalniku smo namestili orodje Tor, se povezali na omrežje in celotno luščenje uporabljali prek teh usmerjevalnikov.

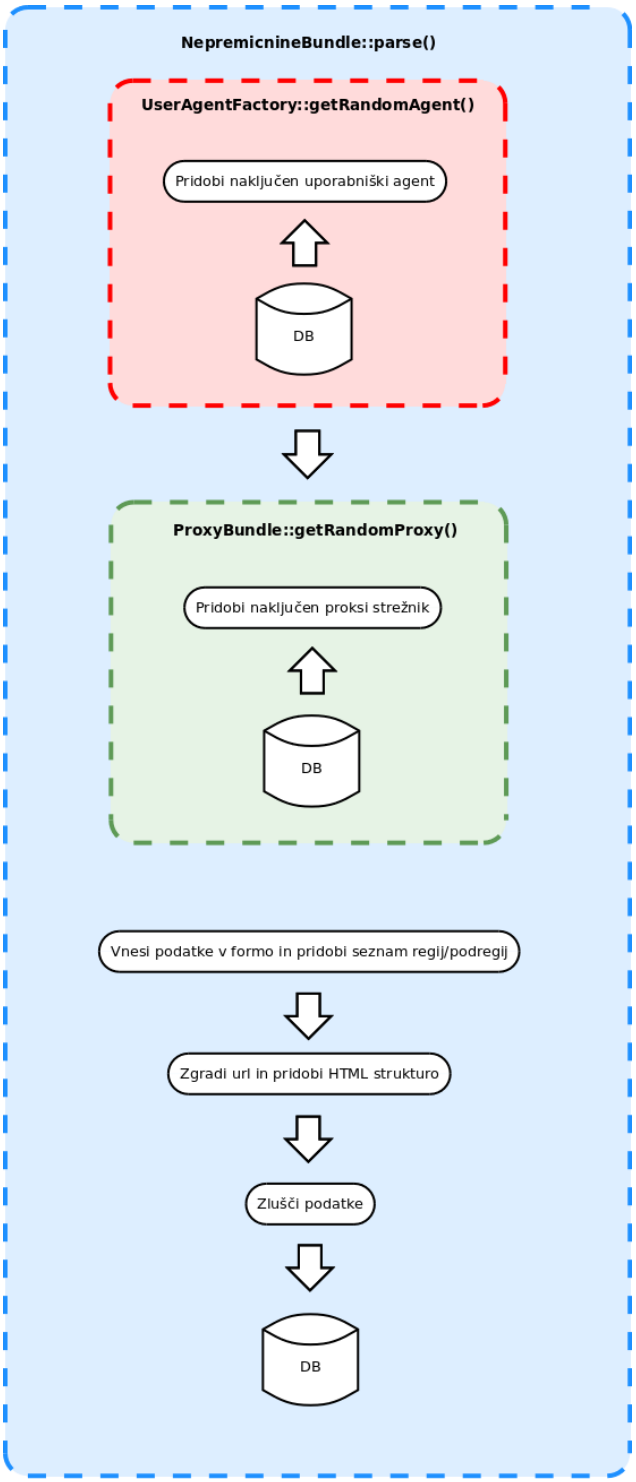
Del anonimizacije je tudi uporabniški agent. Kot pri posredniških strežnikih smo si tudi tu izbrali spletno stran, ki ponuja ogromno informacij in podatkovno bazo z agenti vseh večjih spletnih brskalnikov - www.useragentstring .com. Ker orodja, ki jih uporabljamo, privzeto uporabljajo svoj uporabniški agent (tako Curl kot Sahi), smo morali te agente luščiti in pozneje tudi implementirati. Uporaba teh naključnih agentov je bolj v prikaz zmožnosti uporabljene tehnologije, kot anonimizacija, saj smo agentu pripisali lastne podatke v vednost skrbnikom sistemov. Teh agentov ni potrebno osveževati tako pogosto kot posredniških strežnikov, ker se dodajajo samo z novimi verzijami brskalnikov. Uporabili smo samo večje brskalnike, kot so Internet explorer, Google Chrome in Mozilla Firefox.

4.3 Uporabniški vmesnik

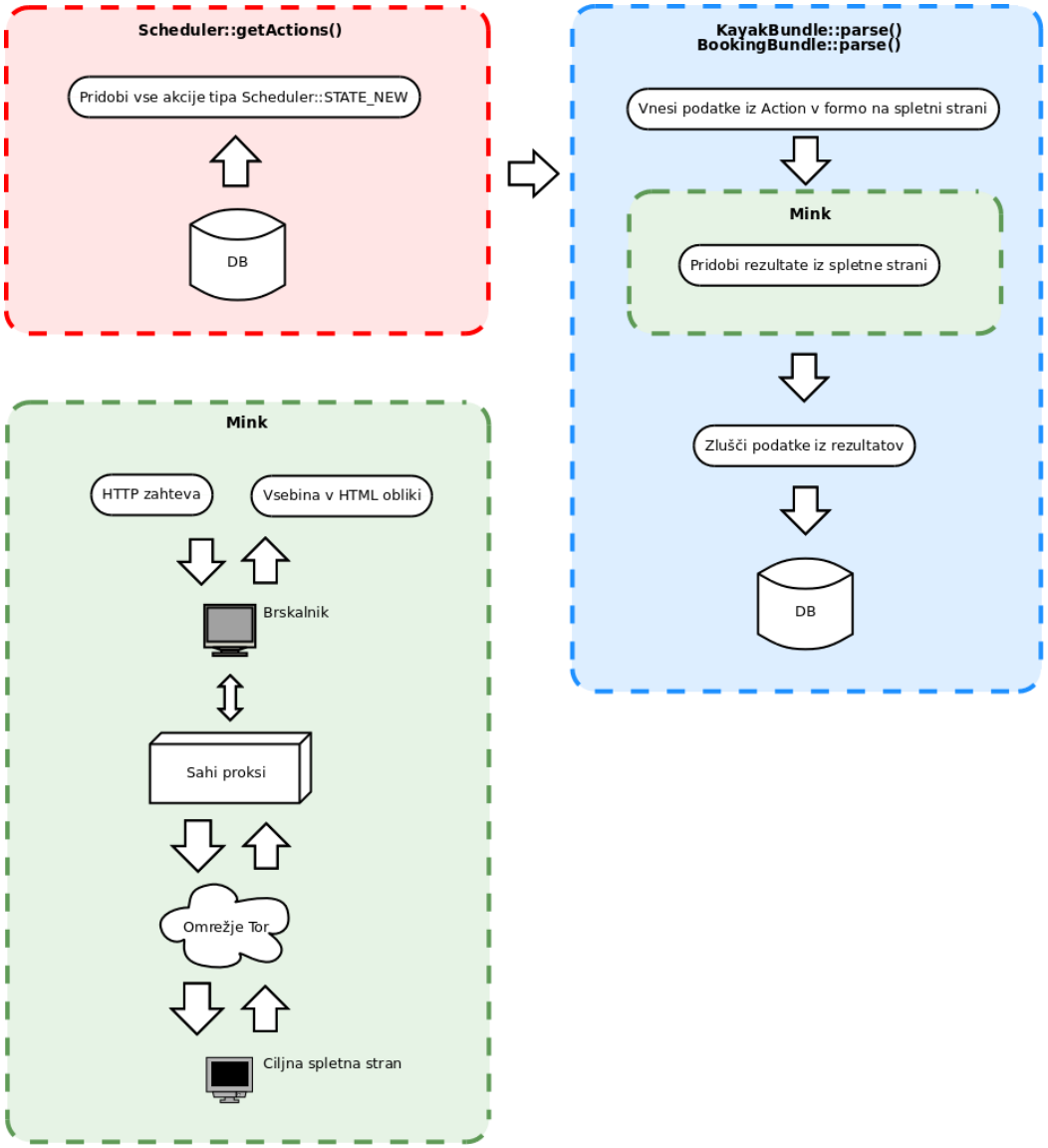
Za lažji pregled prenesenih podatkov smo realizirali poglede v podatkovno bazo s pomočjo Twitter Bootstrap paketa za ogrodje Symfony2. Prepričala nas je predvsem preprosta namestitev in velik nabor že vnaprej pripravljenih gradnikov uporabniškega sistema, nekaj bolj uporabnih:

- uporaba css mreže za lažjo umestitev gradnikov v dele strani in s tem odzivni dizajn
- tipografije, stilizirani seznami in tabele
- razporeditev gradnikov vnosnih form
- nivoji navigacije, meniji
- javascript implementacije modalnih oken in ostalih gradnikov strani

S samo implementacijo smo se poskušali približati osnovni ideji diplomske naloge in hkrati izboljšati pomanjkljivosti open source in komercialnih rešitev spletnih luščilcev. Pomembno nam je bilo predvsem optimizirano delovanje sistema, simulacija uporabniške interakcije z vmesnikom in uporaba različnih tehnologij. Temelj našega sistema za luščenje so lupinske skripte, ki dostopajo do spletnih strani nemoteno in brez potrebe po grafičnem vmesniku, kar nam poveča portabilnost implementacije na praktično kakršenkoli Unix strežnik.



Slika 4.1: Shema osnovnega luščenja podatkov



Slika 4.2: Shema naprednega luščenja podatkov

Poglavje 5

Implementacija prototipa luščenja podatkov iz spletnih strani

Luščilec, ki smo si ga zamislili, bi lahko realizirali s pomočjo bash lupinskih skript, algoritma v obliki spletne strani ali kot unit test v PHPUnit orodju. Izbrali smo ogrodje Symfony zaradi dobre podpore lupinskih skript v konzoli, enostavne implementacije programskih paketov in namestitve podpornih knjižnic. Vsak tip luščenja smo zapakirali v ločen paket, s pripadajočimi zagonskimi skriptami, podatkovnimi entitetami, grafičnim vmesnikom in lastnimi knjižnicami ter predstavili potek osnovne namestitve.

5.1 Zgradba Symfony projekta

etc/

Nastavitve strežnika nginx, v glavni konfiguracijski datoteki nastavimo preverjanje vseh nginx.conf datotek v določeni mapi. V teh specifičnih datotekah nastavljamo poti do korenske mape tega projekta, kot tudi dnevniške mape. Zaradi varnosti je potrebno te datoteke hraniti izven korenskega imenika spletnega strežnika.

log/

Dnevniške datoteke o dostopih do spletne aplikacije in sporočila o napakah pri delovanju (PHP in nginx napake). Smotrna je tudi uporaba logrotate orodja, saj se podatki v te dnevniške datoteke dodajajo in pridobivajo na velikosti.

opt/

Ostala programska oprema, ki ni del Symfony ogrodja. Sem spada sahi posredniški strežnik, katerega zaženemo na lokalni napravi, nastavi pa se ga lahko tudi za delo na oddaljenem strežniku.

Symfony/

Celotno ogrodje se nahaja v tem imeniku, potrebne programske odvisnosti in namestitev se vrši prek orodja Composer. Vsa uporabniška izvorna koda se nahaja v podimeniku src/, zunanje knjižnice pa v podimenik vendors/.

5.2 Podpora lupinskim ukazom

Komponenta Console omogoča enostavno pisanje lupinskih ukazov za implementacijo ponavljajočih se opravil, namestimo jo lahko v posebnem paketu kot knjižnico ali kot del celotnega ogrodja. Kontekstualno smo jih postavili v pripadajoče pakete.

5.3 Namestitev

Namestitev celotnega Symfony ogrodja in dodatnih paketov se izvaja prek sistema Composer. Potrebna je le namestitvena datoteka v JSON formatu v datoteki composer.json. Tu so definirane vse odvisnosti s strani knjižnic in paketov ter minimalne verzije programske opreme. S to datoteko smo vključili Twitter Bootstrap, Doctrine, okolje za predloge Twig, knjižnico za podporo logiranju Monolog, Mink in povezavo med Mink ter Sahi testnim orodjem,

PHP Simple HTML Dom knjižnico in KNP Paginator paket za enostavno uporabo prikazovanja rezultatov po straneh.

5.3.1 Potek namestitve

Koda 5.1 : Projekt kloniramo v mapo

```
1 git clone https://github.com/biftek/wsd.git scraper
```

Koda 5.2 : Posodobimo vse zunanje odvisnosti

```
1 cd symfony/
2 curl -s http://getcomposer.org/installer | php
3 php composer.phar install
```

Koda 5.3 : Namestimo phantomJS

```
1 cd ../
2 wget -O - https://phantomjs.googlecode.com/files/phantomjs-1.9.0-linux-
      x86_64.tar.bz2 | tar xvjf - && mv phantomjs-1.9.0-linux-x86_64 opt/
      phantomjs
```

Koda 5.4 : Posodobimo podatkovno bazo

```
1 php symfony/app/console doctrine:schema:update
```

Projekt namestimo iz oddaljenega git repozitorija, tu je shranjena celotna implementacija našega luščilca skupaj z lupinskimi zagonskimi skriptami, lastnim naborom luščilne knjižnice, predlogami čelnega dela sistema, administrativnim vmesnikom in vmesnikom med entitetami v podatkovni bazi ter implementacijo. Ostale zunanje knjižnice posodabljamo prek sistema composer, saj nam tako ni potrebno skrbeti za novejše verzije v sistemu git, ampak periodično preverjamo nove izdaje s programom composer.phar. V mapo opt/ shranimo tudi Sahi emulator in Phantom.js brezglavi brskalnik.

Shemo podatkovne baze nam ni potrebno shranjevati v sql datoteko, saj doctrine samodejno prevede kodo iz mapirnih razredov v sql sintakso.

5.3.2 Grafični vmesnik

Zaradi potrebe po bolj agilnem pristopu k realizaciji implementacije smo za čelni del administracije uporabili ogrodje Twitter Bootstrap, saj nam je znatno razbremenil čas razvoja vmesnika in zahtevanih znanj CSS sintakse. Symfony2 omogoča veliko uporabnih orodij v lupini, ki nam poenostavijo postavitev grafičnih vmesnikov med entitetami in podatkovno bazo s pomočjo CRUDL (Create, Read, Update, Delete, List). V večini primerov smo lahko že s samodejno zgenerirano kodo in Twitter Bootstrapa upravljali s podatki. Celoten čelni del lahko z lahkoto nadgrajujemo z ogrodjem Assetic, ki prevaja LESS datoteke v CSS stile, sistem Twig, pa skrbi za generiranje HTML vmesnika prek predlog .html.twig.

Slika 5.1: Primer vmesnika, ki ga zgeneriramo z lupinsko skripto in obogatimo s Twitter Bootstrap ogrodjem

Koda 5.5 : Kreiranje vmesnika za entiteto Action paketa SchedulerBundle

```
1 php app/console generate:doctrine:crud --entity=ScraperSchedulerBundle:
      Action
```

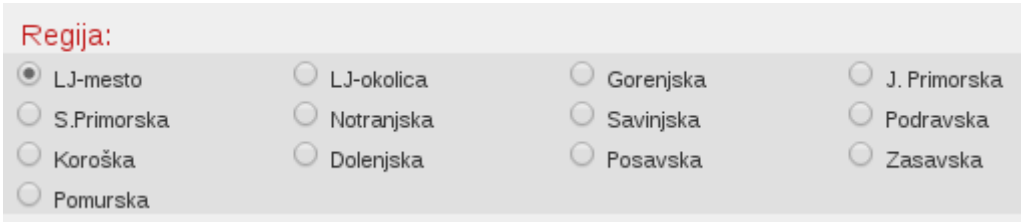
5.4 Osnovno luščenje

5.4.1 NepremicnineBundle

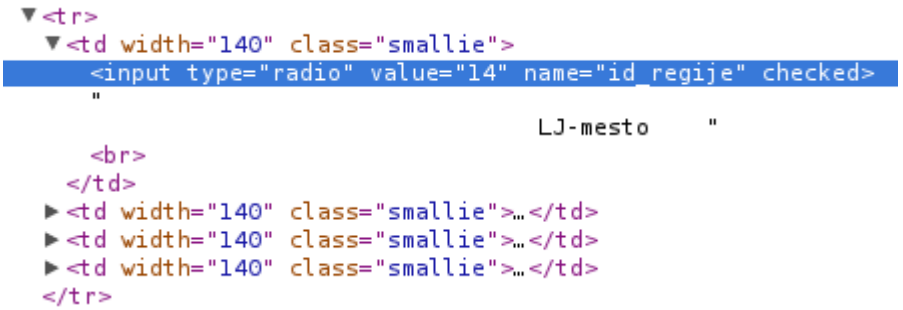
Za osnoven prikaz luščenja smo si izbrali stran nepremicnine.net, predvsem zato, ker ne uporablja dinamičnega prikaza podatkov na strani s tehnologijo AJAX. Uporablja namreč HTML strukturo, ki je na začetku izgledala enostavna, a se je včasih izkazala za trd oreh, predvsem zaradi nepravilno gnezdenih elementov. Ko smo pridobili vse identifikacijske številke ID posameznih regij, smo jih vključili v iskalnem nizu spletnega naslova, kjer smo za vsako regijo pridobili še ves nabor prej definiranih področij. Le-te smo nato v dodatnem zahtevku uporabili pri iskanju vseh možnih oglasov v tej regiji.

Za prvi primer luščenja smo si pomagali s knjižnico PHP Simple HTML DOM Parser, katera skrbi za prenos podatkov in dostop do DOM elementov s preprosto PHP sintakso. Omogoča nam tudi luščenje podatkov iz nepravilno strukturiranih HTML dokumentov.

Nepremicnine.net smo vzeli kot primer dveh orodij, s katerimi lahko luščimo podatke v prostokodni domeni. Simple HTML DOM Parser vsebuje tako celoten nabor funkcionalnosti, s katerimi lahko črpamo podatke. Alternativo v tem paketu pa smo realizirali s pomočjo privzetih orodij v programskem jeziku PHP. Tako smo s knjižnico XPath in Curl dostopali do spletnih strani, jih prenesli na lokalni strežnik in obdelali brez dodatne namestitve zunanje programske opreme.



Slika 5.2: Izbira regij na strani nepremicnine.net



Slika 5.3: Izbira regij v HTML kodi

Koda 5.6 : Izbira regij s SimpleHTMLDom

```
1  foreach ($this->dom->find('input[name=id_regije]') as $radio) {
2        $regions[$radio->value] =
3          rtrim(
4            preg_replace(
5              "/^\s+(.*)\s+/",
6              '$1',
7              $radio->parent()->plaintext
8            )
9          );
10 }
```

Koda 5.7 : Izbira regij z XPath

```
1  $regionList = $xpath->query('//input[@name="id_regije"]');
2      foreach ($regionList as $node) {
3        $regions[$node->getAttribute('value')] =
4          rtrim(
5            preg_replace(
6              "/^\s+(.*)\s+/",
7              '$1',
8              $node->nextSibling->nodeValue)
9          );
10     }
```

Časovni razmak med zahtevki lahko nadziramo s konstantama DELAYRANDOMIZE in DELAYTTL, saj v nasprotnem primeru dostopamo z maksimalno pasovno širino, to pa je z lahkoto izsledljivo, predvsem pa preveč obremeni

ciljne strežnike. Smotrno je torej uporabljati nek naključen razmak, za fiksno časovno pavzo, pa lahko preverimo datoteko robots.txt na nepremicnine.net in se navedene zakasnitve tudi držimo.

5.4.2 ProxyBundle

Posredniške strežnike uporabljamo takrat, ko nas oddaljeni strežniki omejijo na število zahtevkov na določeno časovno enoto, ali ko želimo biti anonimni. Tak prenos podatkov sicer ni varen, saj deluje posredniški strežnik kot "middle man". V našem primeru pa tega problema ni, saj prenašamo prosto dostopne in javne podatke. Tu uporabljamo isto tehniko luščenja kot pri paketu NepremicnineBundle, saj se je xpath v povezavi s Curl izkazal za najenostavnejšo rešitev. Glede na to, da vsebuje stran več kot 40 strani strežnikov, smo se omejili le na določeno število, saj so razvrščeni po hitrosti in stabilnosti.

To je tudi prva stran, pri kateri se je na sami strukturi HTML dokumenta poznalo omejevanje dostopa spletnih pajkov. Pri izpisu IP številke posredniškega strežnika uporabniku, so si avtorji pomagali s skrivanjem določenih cifer (ki ne spadajo v IP naslov) s CSS stili. Ko smo pa pričeli z luščenjem, je IP naslov nenadoma pridobil preveč podatkov. Rešitev je bila v bolj specifični uporabi HTML atributov.

Koda 5.8 : Dostop do IP številke

```
1 $ipSpans  = $serverStats->item(0)->childNodes->item(0)->childNodes;
2 $ipValues = array();
3
4 foreach ($ipSpans as $span) {
5   if ($span->nodeName == 'span' && $span->getAttribute('class') !== '')
        {
6     $ipValues[] = $span->nodeValue;
7   }
8 }
```

Slika 5.4: Struktura IP številk posredniških strežnikov v HTML kodi in tehnika omejevanja luščenja

5.4.3 UserAgentBundle

Velik delež spletnih aplikacij uporablja med drugim tudi najenostavnejše in najbolj razširjeno preverjanje, ali se skriva za zahtevkom človek ali avtomatizirano orodje. To je pregledovanje vsebine uporabniškega agenta. Če ta podatek vsebuje nestandardno orodje ali je celo brez vsebine, aplikacija zavrne dostop ali prikaže drugačen izpis. Tako se poleg IP naslovov legitimira tudi spletne pajke, saj ti zapišejo identifikacijo v to polje.

Kot primer dobre prakse smo tudi mi uporabili prilagojenega uporabniškega agenta, tudi zaradi morebitnega odziva strani, ki smo jih luščili. Omejili smo se tudi na družine najbolj znanih brskalnikov, tako smo zbrali le uporabniške agente Mozille, Internet explorerja, Google chroma in Safarija. Do agentov smo nato prek naše knjižnice naključno dostopali in jim dodali osebni podpis.

Koda 5.9 : Izbira naključnega uporabniškega agenta s podpisom

```
1 $UAFactory = new UserAgentFactory($em);
2 $UAFactory->setPersonalInfo(" - [Web scraping test, contact peter.
      grlica@gmail.com for more info.]");
3
4 $agent = $UAFactory->getRandomAgent();
```

5.5 Primer naprednejšega luščenja podatkov

5.5.1 KayakBundle

Pri naprednem luščenju smo se osredotočili na strani z več dinamične vsebine, predvsem strani, prikazanimi s pomočjo AJAX tehnologije. Stran kayak.com je iskalnik letalskih prevoznikov, hotelov in ostale turistične ponudbe. Tu smo se posvetili predvsem iskanju kart na določene destinacije.

V bonton spletnega luščenja spada tudi pregledovanje robots.txt datoteke, na strani kayak je omenjena nam najbolj pomembna podstran – flights/, za katero je prepovedan dostop za spletne pajke. Tako bi z dostopom naše implementacije spletnega luščilnika kršili pravila uporabe spletne strani. Svetla točka se nam nakaže v teh pravilih uporabe, saj je navedeno, da je edina izjema uporaba teh podatkov v lastne namene, kjer se vsebina ne bo v nikakršnem primeru uporabljala v komercialnih rešitvah.

Do iskalnika lahko dostopamo kar prek naslovnega polja, saj lahko iščemo po letališčih z njihovimi oznakami, npr. /flights#/LON-REK/2013-07-03/2013-07-25. Za prikaz možnosti uporabe te tehnologije pa smo avtomatizirali tudi samo iskanje prek obrazca, iz katerega smo potem preusmerjeni na prej omenjeni spletni naslov.

Prek knjižnice Mink preprosto dostopamo do emulatorja brskalnikov Sahi. Ta emulator lahko konfiguriramo na katerikoli napravi v dosegu našega programa, saj se vsi zahtevki vršijo prek Sahi posredniškega strežnika. Tako imamo v tej knjižnici orodja za delo z vnosnimi elementi (klik na gumb, vnos v polje, ...), akcijo primi in spusti itd. Predvsem smo uporabljali vnos v polja (določili smo datume) ter čakanje na izid neke akcije. Pri iskanju kart smo morali namreč počakati na dokončen izpis vseh rezultatov. Tu spet naletimo na omejevanje spletnih pajkov, saj se najcenejše karte načeloma prikazujejo na začetku strani, a so s tehnologijo AJAX prikazane proti koncu zahtevka. Tako se celotna stran v brskalniku izriše v manj kot 5 sekundah, preostali AJAX klici pa se vršijo še minimalno naslednjih 50 sekund. V takšnih situacijah je najlažje preverjati, ali se prikaže/skrije HTML element z obvestilom o številu

zadetkov iskanja, Sahi to rešuje s funkcijo wait(). Sprejema dva argumenta, po katerih spremlja časovno omejitev prikaza elementa. Čaka torej tako dolgo, dokler ni eden od pogojev zadoščen, bodisi je to število milisekund zakasnitve, ali pa preveri javascript pogoj v drugem argumentu.

Koda 5.10 : Wait funkcija pregleduje pojavnost HTML elementa

```
1 $session->wait(100, "$('.introtext').length > 0");
```

Tako je spletna stran kayak.com že z načinom prikaza vsebine omejila večjo količino zahtevkov na iskalno stran, saj se ob prenosu strani s tekstualnim brskalnikom (ali v primeru Mink – Goutte headless brskalnikom) prikaže samo stran s filtri brez kakršnihkoli rezultatov. Povprečni čas iskanja določene relacije tako traja dobro minuto.

Ob koncu iskanja se rezultati izpišejo v samo strukturo strani, tako da se pri večstranskih rezultatih spletna stran ne osveži, ampak z uporabo Javascript jezika preprosto nadomesti. Tako je za stran z 10 rezultati potrebno rahlo manj izvedbenega časa kot za stran s 500 rezultati. Najpočasnejši del celotnega luščenja te strani je čakanje na rezultate, končni prenos HTML in luščenje z Xpath sintakso sta z uporabnikovega stališča zanemarljiva.

Ker se pri tem programskem paketu ne povežemo na spletno stran prek Curl, moramo nastaviti posredniški strežnik, prek katerega dostopamo do spletne strani, v Sahi nastavitvah. Le-ta pa ne omogoča dinamičnega naslavljanja posredniških strežnikov (vsaka sprememba v konfiguracijskih xml datotekah zahteva vnovični zagon Sahi strežnika). Tako smo se v želji za anonimizacijo zadovoljili s Tor omrežjem. Tor strežnik smo postavili na lokalno napravo na vrata 9050 in ga nastavili kot izhodni strežnik v Sahi nastavitvah. Če bi pa hoteli s pomočjo Tor omrežja pri vsaki zahtevi dostopati prek naključnega strežnika, bi morali avtomatizirati še ta del. Potrebovali bi programsko spremembo Sahi nastavitev v xml datoteki in ponoven zagon samega Tor strežnika. S tem bi pa predvsem izgubili dragocen čas pri inicializaciji omrežja (postaviti se mora handshake z vsemi klienti in celotna pot do zadnjega usmerjevalnika).

Koda 5.11 : Tor posredniški strežnik v Sahi nastavitvah

```
1 # Use external proxy server for http
2 ext.http.proxy.enable=true
3 ext.http.proxy.host=localhost
4 ext.http.proxy.port=9050
```

5.5.2 BookingBundle

Primer luščenja spletne strani kayak.com smo še nadgradili z iskanjem hotelov v končni destinaciji iskalnika letov. Tu je bila sama zasnova problema enaka kot na kayak.com, spet smo uporabili Sahi posredniški strežnik, ampak z drugim emulatorjem brskalnika – Phantom.js.

Uporaba knjižnice Mink je zelo olajšala delo, saj je bilo v pri headless emulatorju Phantom.js nemogoče vizualno spremljati dogajanje ob dostopu do spletne strani. Ob izrednih napakah smo si pomagali z avtomatsko generacijo zaslonskih slik, pozitivna plat Mink knjižnice, pa se je prikazala prav pri transparentnosti uporabe brskalnikov. Skozi celotno implementacijo smo uporabljali navaden brskalnik kot pri KayakBundle, ko je luščilec pravilno deloval, pa smo pričeli z uporabo Phantom.js

Ker je bilo samo delo z brskalnikom počasno (zagon in emulacija dela z gradniki strani), smo si s hitrostjo "headless" brskalnika Phantom.js opazno optimizirali luščenje. Za most med Sahi in Phantom.js brskalnikom smo morali implementirati svojo skripto v javascriptu in jo povezati z obema orodjema. Brskalnike lahko dodajamo v Sahi xml konfiguracijsko datoteko, kjer lahko tudi sami definiramo, s katerimi orodji se bomo povezovali na specifično spletno stran.

Koda 5.12 : Konfiguracija spletnega brskalnika v nastavitvah Sahi

```
1 <browserType>
2   <name>phantomjs</name>
3   <displayName>PhantomJS</displayName>
4   <icon>safari.png</icon>
5   <path>/usr/share/nginx/www/scraper/opt/sahi/extlib/phantomjs/bin/
        phantomjs</path>
```

```
6     <options>--proxy=localhost:9999 /usr/share/nginx/www/scraper/opt/sahi/
          extlib/phantomjs/bin/phantom-sahi.js</options>
7     <processName>phantomjs</processName>
8     <capacity>100</capacity>
9     <useSystemProxy>false</useSystemProxy>
10  </browserType>
```

Tu smo specificirali, naj se Phantom.js brskalnik povezuje prek vrat 9999, na katerih smo postavili Sahi strežnik. Skripto smo napisali predvsem zaradi tega, da lahko po želji razpolagamo s posredniškimi strežniki, ter se s tem ne zavežemo na uporabo Tor omrežja. Vse zahteve zatem pošiljamo iz lupinskih skript prek Sahi knjižnice in povezovalne skripte v Phantom.js brskalnik.

Ker uporabljamo po meri napisano skripto, ki uporablja brskalnik, smo si lahko tudi sami definirali uporabniškega agenta. Tu smo si pomagali že z uporabljeno kodo, ki pri ostalih luščenjih pomaga pri dostopu do naključnega uporabniškega agenta. Da bi pa lahko Phantom.js uporabil ta podatek, smo morali napisati lupinsko skripto v Symfony projektu, ki te podatke pridobi, ter bash komandno skripto, katera se nanjo sklicuje. V skripti, ki povezuje Sahi in Phantom.js, pa smo prek child_process modula klicali bash skripto. Nekatere spletne strani preverjajo tudi sumljive podatke v zahtevah, zato smo v želji za čimvečjo transparentnost uporabili tudi posredniško polje (referer).

Koda 5.13 : Skripta ki dostopa do podatkov o uporabniških agentih v bazi

```
1  #!/bin/bash
2
3  HOME="/usr/share/nginx/www/scraper/Symfony"
4  AGENT=‘/usr/bin/php $HOME/app/console scrape:get_user_agent‘
5
6  echo -n $AGENT
```

Koda 5.14 : Sahi-phantom.js povezovalna skripta

```
1  if (phantom.args.length === 0) {
2      console.log('Usage: phantom-sahi.js <Sahi Playback Start URL>');
3      phantom.exit();
4  } else {
5      // argument, ki ga dobimo iz Sahi - zahtevana spletna stran
```

```
6       var address = phantom.args[0];
7       var page    = new WebPage();
8
9       // nakljucni uporabniski agent
10      var spawn = require('child_process').spawn;
11      var ua    = spawn('/usr/share/nginx/www/scraper/Symfony/bin/
            get_random_ua.sh', []);
12
13      ua.stdout.on('data', function (ua) {
14          page.settings.userAgent = ua;
15
16          // izvor zahtevka
17          page.customHeaders = {
18              "referer": "http://www.google.com/"
19          };
20          page.open(address, function(status) {
21              if (status === 'success') {
22                  return true;
23              } else {
24                  return false;
25              }
26          });
27      });
28  }
```

5.5.3 SchedulerBundle

Za upravljanje z lupinskimi skriptami v ozadju smo se odločili za nov paket, ki vsebuje entitetni tip Action. Z vnosom nove akcije v bazo v administracijskem vmesniku lahko razpolagamo z odhodi in prihodi v specifično mesto na določen datum vzleta in pristanka. Akcijam lahko specificiramo terminske razporede, tako lahko luščimo strani na točno določen datum.

Pripravili smo razred, ki s temi akcijami razporeja - tako lahko v KayakBundle in BookingBundle dobimo vse akcije, ki še niso bile prožene. Ko nam Scheduler::getActions vrne celoten seznam novih akcij jih uporabimo v obeh paketih. Tako KayakBundle uporabi datum odhoda in prihoda in jih vnese s pomočjo Sahi knjižnice v čelni sistem spletne strani kayak.com. V ozadju tudi BookingBundle vsakih 30 sekund s pomočjo akcij preverja statuse in

na podlagi obeh datumov začne iskati proste hotele v ciljni destinaciji. Če katerakoli lupinska skripta konča delovanje brez napak, nastavi vrednost akciji na Scheduler::STATE_FINISHED.

Poglavje 6

Ugotovitve in zaključek

Spletno luščenje je zaradi socialnih omrežij, spletnih trgovcev in ponudnikov vsebin v neprestanem porastu in naj bi dosegalo prek 5% spletnega prometa (od skupno 51% prometa, ki ni generiran s strani človeka)[30]. S tem porastom se pojavljajo vedno nove tehnike zaviranja takšnih dostopov do vsebine, ter spletnih storitev, ki te zahteve identificirajo in onemogočajo (Distil.it, SiteBlackBox, Sentor Assassin in Pubcrawl).

Pri tej diplomski nalogi smo poskušali pokazati, kakšne prepreke nas čakajo pri spletnem luščenju podatkov in samo vprašanje, ali je naše početje sploh legalno. Zavedati se moramo namreč, da naš primer spletnega luščenja teh spletnih strani še zdaleč ne sega v sposobnosti komercialnih luščilcev, pajkov spletnih iskalnikov ali blackhat orodij. Pri naši implementaciji smo bili omejeni predvsem pri pasovni širini dostopa do strani nepremicnine.net (sposobni smo bili luščiti 30 oglasov na 5 sekund pri minimalni zakasnitvi dostopa). Dnevno smo tako sposobni dostopati do 20.000 podstrani rezultatov. Hitrost pa je odvisna tudi od pasovne širine in obremenjenosti naključnih posredniških strežnikov, ki jih uporabljamo pri dostopu. Osnovno luščenje bi lahko še razširili s pomočjo curl_multi_init() klica, ki kreira vzporedne povezave z enim curl oprimkom.

Prekršili smo tudi eno prvih pravil bontona spletnega luščenja, saj se nismo držali navodil datoteke robots.txt, kjer je definirana zakasnitev pajkov

minimalnih 10 sekund.

6.1 Nadgradnje

Pri naprednem luščenju smo prišli do drugačnega problema, lokalno nas je zaviral emulator spletnega brskalnika v želji za čimvečjo simulacijo uporabnika. Spletna stran kayak.com namreč umetno zavira izpis rezultatov s tehnologijo AJAX in s tem podaljša čas luščenja za 10-kratni faktor. V primeru preverjanja uporabniške interakcije na strani, kot so vpisovanje vrednosti v vnosna polja črko za črko (in ne copy/paste, kot smo uporabljali z orodjem Mink) ter premikanje miškinega kurzorja (xdotool ukaz v linux lupini) bi samo delovanje le še upočasnilo.

Za napredno luščenje je možnosti nadgradnje več:

- uporaba podatkovne baze spletnih naslovov
- naprednejša uporaba Tor omrežja
- več Phantom.js luščilnih delavcev
- najem instance strežnika v oblaku
- luščenje strani v geografski bližini

S podatkovno bazo bi si lahko zagotovili paralelnost zahtev, saj bi glavni algoritem dostopal do strani, vnašal podatke v vnosna polja in naslove teh rezultatov zapisal v bazo. Preostali del implementacije bi predstavljali delavci, ki bi vsak spletni naslov v podatkovni bazi rezervirali z atomarno transakcijo (do strani moramo dostopati samo enkrat). S to idejo se izognemo večkratnim zaporednim dostopom do strani (in s tem lažjo detekcijo našega orodja) ter naključno zakasnitvijo med zahtevki. Pri avtomatiziranih orodjih se tudi pri naključnem dostopu lahko opazi trende dostopov (predvsem časovne), saj so le-ti pri uporabnikih veliko bolj naključni kot pri pajkih. Glavne razlike so pri amplitudi in mejnih vrednostih, uporabniški zahtevki se zgostijo ob jutrih

in popoldnevih, pajki in luščilci pa čez celoten dan. Razlike med dostopi se obravnava na podlagi izvora zahteve in frekvence dostopa. Minimalno število dostopov se obravnava regularno, saj je število informacij, ki grejo ven iz sistema zanemarljivo. Te zahteve predstavljajo veliko večino uporabniških dostopov. Nadpovprečno število dostopov iz izvora in visoka frekvenca zahtev neželene pajke postavi na črni seznam, velikokrat pa zaradi tega trpijo tudi uporabniki za posredniškimi strežniki.

Na lokalno napravo lahko namestimo tudi Tor strežnik v več primerkih, ter jih povežemo na določen razpon vrat. Na njih se lahko po želji povežemo prek naše programske opreme, zakasnitev ob vnovičnem povezovanju na Tor usmerjevalnike ni. Omejuje nas pasovna širina in velikost pomnilnika na lokalni napravi.

Phantom.js se lahko v povezavi s Sahi posredniškim strežnikom postavi tudi na več oddaljenih strežnikov, dodajanje novih delavcev, pa je preprosto, saj potrebujemo le dodatno nastavitev v xml konfiguracijski datoteki. Ta rešitev bi delovala na osnovi podatkovne baze, saj bi bilo potrebno v nasprotnem primeru implementirati sinhronizacijski mehanizem. V tej fazi bi lahko uporabljali tudi rotacijo že obstoječih javnih posredniških strežnikov, ki jih že imamo v bazi. Spletne strani z veliko prometa imajo te strežnike na črni listi, zato seznam vodimo prioritetno. Ob prvem prikazu varnostnega sistema CAPTCHA pa označimo uporabljen posredniški strežnik za neuporaben in se nekaj časa ne usmerjamo prek njega ali ga celo brišemo iz podatkovne baze. V izogib zakasnitvam pa bi lahko tudi izbirali posredniške strežnike v geografski bližini ciljne spletne strani.

Pogosta rešitev pri spletnem luščenju je tudi najem EC2 instance v Amazonovem oblaku (Amazon Web Services AWS). Amazon Elastic Compute Cloud EC2 omogoča najem navidezne naprave, na katero lahko preko spletne storitve namestimo kakršnokoli programsko opremo, administracija naprave pa je namenjena izključno naročniku. Najbolj primerne navidezne naprave za takšne rešitve so predvsem paketi z veliko vhodno/izhodno zmogljivostjo (predvsem pasovno širino), kot so paketi m1.large ali m1.xlarge, kjer je

pasovna širina 500 Mbps do 1000 Mpbs. Elastičnost zagotavljajo pri najemu, saj se stroški gibljejo od 0.24$ do 0.48$ na uro. Ker s tako veliko pasovno širino kmalu obremenimo ciljne spletne strežnike, je pomembno celotno delo porazdeliti. V primeru luščenja večih spletnih strani, pa nam paketi s premalo pasovne širine ne zadoščajo več, zato lahko najamemo več manjših instanc in paraleliziramo zahteve. Ker AWS ponuja veliko različnih paketov, pa ob primanjkovanju procesorske moči ali pomnilnika lahko posežemo po večjih instancah, kot je hi1.4xlarge. Tu si lahko še dodatno pomagamo z NoSQL podatkovnimi bazami, kot so Cassandra ali MongoDB in SSD diski ter 10Gbps pasovne širine.[12] Zaradi izredne priljubljenosti storitev v oblaku v tej panogi, pa veliko spletnih strani že omejuje promet iz IP številk amazonovih strežnikov, zato nekateri luščilci usmerjajo svoj promet prek Tor omrežja kar na EC2 instancah.

Literatura

[1] Matthew Turland, "php|architect's Guide to Web Scraping with PHP", *Introduction - Web Scraping Defined*, str. 2

[2] Paul Ford, "How Google beat Amazon and Ebay to the Semantic Web". Dostopno na:
`http://www.ftrain.com/google_takes_all.html`

[3] Johan Hjelm, "Creating the Semantic Web with RDF" *Languages and Ontologies*, str. 201–206

[4] T. Berners-Lee, MIT/W3C, D. Connolly, "Hypertext Markup Language - 2.0". Dostopno na:
`http://tools.ietf.org/html/rfc1866`

[5] "Introduction to HTML 4". Dostopno na:
`http://www.w3.org/TR/html401/intro/intro.html`

[6] Anne van Kesteren, Simon Pieters, "HTML5 differences from HTML4". Dostopno na:
`http://www.w3.org/TR/html5-diff/`

[7] Bill Evjen, Kent Sharkey, Thiru Thangarathinam, Michael Kay, Alessandro Vernet, Sam Ferguson, "Professional XML" *XML Syntax*, str. 3–17

[8] Daniel Stenberg, "550M users". Dostopno na:
`http://daniel.haxx.se/blog/2012/05/16/300m-users/`

[9] Fabien Potencier, "Why Symfony?". Dostopno na:
http://fabien.potencier.org/article/65/why-symfony

[10] Matthew Turland, "php|architect's Guide to Web Scraping with PHP", *Legality of Web Scraping*, str. 165–167

[11] Ronald M. Whyte, "EBAY, INC., Plaintiff, vs. BIDDER'S EDGE, INC., Defendant.". Dostopno na:
https://www.law.upenn.edu/fac/pwagner/law619/f2001/week11/bidders_edge.pdf

[12] Amazon, "Amazon EC2 Instance types". Dostopno na:
http://aws.amazon.com/ec2/instance-types/

[13] Symfony, dostopno na:
http://symfony.com

[14] Selenium, dostopno na:
http://seleniumhq.org

[15] Selenium2, dostopno na:
http://code.google.com/p/selenium

[16] Sahi, dostopno na:
http://sahi.co.in

[17] Watir, dostopno na:
http://watir.com

[18] Phantom.js, dostopno na:
http://phantomjs.org

[19] Goutte, dostopno na:
http://mink.behat.org/#gouttedriver

[20] Zombie.js, dostopno na:
http://zombie.labnotes.org

[21] Visual Web Ripper, dostopno na:
`http://visualwebripper.com`

[22] WebSundev Extractor, dostopno na:
`http://websundew.com`

[23] Easy Web Extractor, dostopno na:
`http://webextract.net`

[24] Web Content Extractor, dostopno na:
`http://newprosoft.com/web-content-extractor.htm`

[25] Mozenda, dostopno na:
`http://mozenda.com`

[26] OutWit Hub, dostopno na:
`http://outwit.com/products/hub`

[27] Helium Scraper, dostopno na:
`http://heliumscraper.com`

[28] Scraper extension, dostopno na:
`https://chrome.google.com/webstore/detail/scraper/mbigbapnjcgaffohmbkdlecaccepngjd?hl=en`

[29] Državni zbor, "Kazenski zakonik RS". Dostopno na: `http://www.dz-rs.si/wps/portal/Home/deloDZ/zakonodaja/izbranZakonAkt?uid=1B04368E4254FD4BC12575C40026DF98&db=urad_prec_bes&mandat=VI`

[30] Tom Foremski, "51% Of Internet Traffic Is 'Non-Human"', dostopno na:
`http://www.zdnet.com/blog/foremski/report-51-of-web-site-traffic-is-non-human-and-mostly-malicious/2201`