UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matjaž Horvat

Nadgrajeno brskanje

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Saša Divjak

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01595/2009
Datum: 15.10.2009

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

Kandidat: **MATJAŽ HORVAT**

Naslov: **NADGRAJENO BRSKANJE**
AUGMENTED BROWSING

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Diplomsko delo obravnava področje nadgrajenega brskanja, ki obiskovalcem spletnih strani, ki nimajo skrbniškega dostopa, omogoča spreminjanje spletnih strani.

Analizirajte tehnologije in programska orodja za tako nadgrajevanje spletnih strani. Podajte ključne pasti, na katere motra biti pozoren razvijalec. Podajte primer nadgradnje spletne strani in svoje delo primerno dokumentirajte.

Mentor:

prof. dr. Saša Divjak

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Franc Solina

IZJAVA O AVTORSTVU
diplomskega dela

Spodaj podpisani Matjaž Horvat

z vpisno številko 63030124

sem avtor diplomskega dela z naslovom

Nadgrajeno brskanje.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Saše Divjaka;
- 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. 12. 2009 Podpis avtorja _____

Zahvaljujem se svoji najmlajši sestri Tini Horvat Vidovič, ki me je pred zagovorom diplomskega dela vztrajno vabila na obisk v Rim. Ker sem si jo na njenem domu v večnem mestu želel obiskati šele po pridobitvi naziva univerzitetni diplomirani inženir računalništva in informatike, je bilo njeno vztrajno ponavljanje vabila najboljša motivacija za pisanje diplomskega dela.

Kazalo

Seznam uporabljenih kratic in simbolov

HTML	HyperText Markup Language (jezik za označevanje nadbesedila)
URL	Uniform Resource Locator (enolični krajevnik vira)
URI	Uniform Resource Identifier (enotni označevalnik vira)
DOM	Document Object Model (objektni model dokumenta)
HTTP	HyperText Transfer Protocol (protokol za prenos hiperteksta)
CSS	Cascading Style Sheets (prekrivni slogi)
XML	Extensible Markup Language (razširljivi označevalni jezik)
XPCOM	Cross-Platform Component Object Model
XPConnect	Cross Platform Connect
XPI	Cross-Platform Installer
XUL	XML User Interface Language
W3C	World Wide Web Consortium
API	Application Programming Interface (programski vmesnik)
JSON	JavaScript Object Notation
RDF	Resource Description Framework
CMS	Content Management System (sistem za upravljanje vsebine)

Povzetek

Uporabniki si pri brskanju po spletu vse bolj pogosto želijo uporabniško izkušnjo, ki je prilagojena njihovim osebnim željam. Diplomsko delo obravnava področje nadgrajenega brskanja, ki obiskovalcem spletnih strani, ki nimajo skrbniškega dostopa, omogoča spreminjanje spletnih strani.

S tehnologijami in programskimi orodji, ki jih predstavimo v poglavju Pregled tehnologij, lahko obiskovalci spletne strani prilagodijo svojih zahtevam. Ker so prilagoditve lahko zelo raznolike, so na voljo številne tehnologije, med katerimi ni mogoče enolično izluščiti najboljšega, saj ima vsaka svoje prednosti in slabosti.

Poleg omenjenih tehnologij in programskih orodij mora razvijalec pri nadgrajevanju spletnih strani obvladati tudi osnovne spletne programske jezike: HTML, CSS in JavaScript. Zato v poglavju Pogosti problemi spoznamo nekaj ključnih pasti, na katere mora biti pozoren razvijalec, predvsem pri uporabi jezikov CSS in JavaScript.

Osrednji del diplomskega dela predstavlja sistematični pregled izdelave nadgradnje spletne strani na primeru spletne aplikacije Zemanta. Z aplikacijo nadgradimo različne spletne strani s pomočjo bookmarkleta, uporabniške skripte za Greasemonkey, razširitve za Firefox in Jetpacka, povežemo pa jo tudi s storitvijo Ubiquity.

Ključne besede:

Spletna stran, nadgrajeno brskanje, Zemanta

Abstract

When surfing the web users more and more often want the personalized user experience. Diploma thesis addresses augmented browsing, which describes the experience of using a system that can automatically improve the information on web pages, without having administrator access rights.

With technologies and software tools, which are described in the section Pregled tehnologij (Technology overview), visitors can adapt web pages to their personal preferences. Since the adjustments can be very diverse, a number of technologies is made available, each with its advantages and disadvantages.

In addition to these technologies and software tools to augment web pages developer also needs to have basic knowledge of web programming languages: HTML, CSS and JavaScript. Therefore we get to know some key traps in the section Pogosti problemi (Common problems), which developer should pay attention to, especially when using CSS and JavaScript.

The main part of the thesis presents a systematic review of augmenting web pages with a web application called Zemanta. We augment various web pages using the Bookmarklet, Greasemonkey user script, Firefox extension and Jetpack, and we also integrate Zemanta using Ubiquity.

Keywords:
Web page, augmented browsing, Zemanta

1 Uvod

1.1 Problemsko področje

Po oceni iskalnika Wolfram Alpha iz aprila 2009 se na internetu nahaja več kot 230 milijonov različnih spletišč[1], ki jih obiskuje 1,4 milijarde uporabnikov interneta po vsem svetu [5, 6]. Ob tako visokem številu spletišč in še večjem številu njihovih uporabnikov je povsem samoumevno, da bi si marsikdo na marsikateri spletni strani želel marsikaj spremeniti.

Spremembe so lahko zelo različne: od manjših posegov, kot je modra namesto črne barve naslova v bralniku novic Google Reader, do prikazovanja števila neprebranih elektronskih sporočil v ikoni Gmaila. Med izboljšavami lahko najdemo tudi gumb za prenos videoposnetkov iz YouTuba, pa tudi čisto prave spletne aplikacije, ki služijo kot nadgradnja obstoječih.

A večina uporabnikov nima niti potrebnega znanja za razvoj in spreminjanje spletnih strani. Pa tudi tisti, ki ga imajo, ne morejo prilagajati strani, do katerih nimajo skrbniškega dostopa. Izjema so le posebne spletne strani wiki, katerih obiskovalci so hkrati tudi ustvarjalci.

Skrbniški dostop izbranemu uporabniku omogoča, da deloma ali v celoti vpliva na vsebino in izgled spletišča. To pomeni, da lahko zgolj uporabnik s skrbniškim dostopom dodaja, odstranjuje in spreminja besedila, slike, fotografije, zvočne posnetke, video in vse ostale vsebine. Poleg tega lahko vpliva tudi na postavitev spletišč in na njihovo grafično podobo. Velikokrat ima skrbniški dostop več uporabnikov in je razdeljen v več nivojev z različnimi pravicami, npr. urednik, novinar in vzdrževalec spletnega časopisa. V vseh primerih pa velja, da navadni uporabniki spletnih strani nobene izmed teh pravic nimajo. Kako si lahko torej uporabniki prilagodijo spletno stran svojim željam in kako lahko svoje spletne aplikacije vključijo v obstoječe spletne strani? Na to vprašanje odgovarja pričujoče diplomsko delo. Obravnava različne probleme, s katerimi se pri tem srečujemo, in različne tehnike za spopadanje z njimi.

V nadaljevanju naloge v podpoglavju 1.2 predstavimo zahteve in cilje diplomskega dela, v 1.3 pa sorodna dela, saj je marsikateri problem in njegova rešitev že opisan v literaturi. Poglavja 2, 3 in 4 sestavljajo osrednjo vsebino dela. V 2. predstavimo najbolj razširjene tehnologije in programska orodja za nadgrajevanje spletnih strani, pogoste probleme pri nadgrajevanju opišemo v 3. poglavju, 4. poglavje pa natančno pojasnjuje celoten postopek nadgrajevanja strani s primerom iz prakse. Na koncu v podpoglavju 5 podamo še oceno opravljenega dela, predstavimo težave, na katere smo naleteli, in ideje, ki bi lahko bile predmet novih raziskav.

---

[1] Spletišče (tudi spletno mesto) je več spletnih strani povezanih v celoto. Spletišče je nameščeno na spletnem strežniku in dostopno preko interneta.

1.2 Cilji

Temeljni cilj diplomskega dela je izdelati dokumentacijo, ki bo služila kot pripomoček razvijalcem za nadgrajevanje spletnih strani, do katerih nimamo skrbniškega dostopa. Dokumentacijo dopolnjuje tudi primer programske rešitve, s pomočjo katere lažje predstavimo najpogostejše probleme in različne tehnološke rešitve, ki so primerne za nadgrajevanje. Tako diplomsko delo ni samo teoretični priročnik za nadgrajevanje spletnih strani, ampak ponuja tudi številne primere iz prakse, ki jih lahko razvijalci ponovno uporabijo ali prilagodijo za ponovno uporabo v svojih rešitvah.

1.3 Sorodno delo

Nadgrajevanje spletnih strani, do katerih nimamo skrbniškega dostopa, je razmeroma nov pojav. Pojem nadgrajenega brskanja (augmented browsing [1]), ki govori o nadgrajevanju spletnih strani s strani navadnih uporabnikov, se je prvič pojavil leta 1997. Nadgrajevanje razlaga kot možnost, pri kateri si lahko uporabniki brez skrbniškega dostopa prilagodijo izgled in obnašanje spletnih dokumentov po svojih željah.

Med uporabniki se je nadgrajevanje začelo množično širiti leta 2005, ko je Google predstavil Greasemonkey. Ta razširitev za brskalnik Firefox namreč vsem uporabnikom omogoča nameščanje skript, s pomočjo katerih lahko spreminjajo spletne strani v jeziku HTML. Ozadje nadgrajevanja z Greasemonkeyjem in nasvete za izdelavo uporabniških skript je že v letu izida samega Greasemonkeyja izdal O'Reilly v svojem priročniku [2].

Če želimo narediti večji poseg v spletno stran, kot je npr. integracija spletnih aplikacij, in ga ponuditi več uporabnikom, se je potrebno zavedati, da vsi nimajo nameščenega Greasemonkeyja. Zato je namesto uporabniških skript bolj primerno izdelati kar samostojno razširitev za brskalnik. Tega se med drugim poslužuje podjetje Zemanta, kar je na predavanju ob izidu Firefox 3 v ljubljanski Kiberpipi podrobno predstavil Marko Samstur [4].

Na spletnih straneh Zemante [3] je na voljo tudi dokumentacija o integraciji Zemantine spletne aplikacije v platforme za pisanje blogov in sisteme za upravljanje vsebine. Napisana je za uporabnike s skrbniškim dostopom, vendar služi tudi kot odličen pripomoček za izdelavo spletnih aplikacij in njihovo integracijo v obstoječe spletne strani.

2 Pregled tehnologij

Že v uvodu smo omenili, da si lahko uporabniki želijo zelo različnih prilagoditev spletnih strani: od spreminjanja pisave, ki razvijalcu vzame nekaj vrstic programske kode, do pravih spletnih aplikacij, ki dopolnjujejo osnovno spletno stran in jih razvijajo cela podjetja. Zato ne preseneča, da je v kratkem času od pojava nadgrajevanja spletnih strani nastalo več tehnologij, ki služijo temu namenu.

Med številnimi orodji ni mogoče enolično izluščiti najboljšega, saj ima vsako svoje prednosti in slabosti. V tem poglavju bomo spoznali vsa najpomembnejša programska orodja in tehnologije za nadgrajevanje spletnih strani, ki so danes v uporabi.

2.1 Bookmarklet

Decembra 1995 je Netscape Navigator 2.0B3 kot prvi brskalnik ponudil tolmač za skriptni programski jezik JavaScript, prvotno imenovan Mocha in kasneje LiveScript. Istoimensko podjetje je jezik razvilo z namenom, da pomaga programerjem pri ustvarjanju dinamičnih spletnih strani in naprednejših uporabniških vmesnikov. Ker je tolmač del brskalnika, se izvorna koda izvaja na strani odjemalca, kar pred prihodom JavaScripta ni bilo mogoče; vse podatke je bilo treba najprej poslati na strežnik, kjer so se obdelali, rezultati pa so se v obliki statičnih spletnih strani v jeziku HTML vrnili do odjemalca.

Že od prve različice naprej je JavaScript omogočal tudi uporabo protokola javascript: URL v brskalniku po vzoru ostalih protokolov, kot so http:, file: in ftp:. Tako lahko z ukazom javascript: 'Živjo, svet!' ustvarimo nov dokument z želeno vsebino, ukaz javascript: alert(document.links[0].href) pa za izvedbo skripte uporabi DOM trenutnega dokumenta. Spletni naslov oblike javascript: URL lahko shranimo tudi med zaznamke (angl. bookmark) in na ta način dobimo bookmarklet[2]. Razlog za obstoj bookmarkletov se torej skriva v JavaScriptovi shemi URI, ki omogoča, da so programi v JavaScriptu shranjeni v obliki URL-jev, slednje pa lahko shranimo tudi kot zaznamke.

Beseda bookmarklet je zloženka besed bookmark (zaznamek) in aplet (manjši program), torej označuje v JavaScriptu napisane programčke, shranjene v obliki zaznamkov. Za nadgrajevanje spletnih strani je ključnega pomena drugi primer uporabe iz zgornjega odstavka, s katerim vplivamo na DOM dokumenta, ki ga imamo trenutno odprtega. Bookmarkleti lahko trenutno stran preiskujejo in spreminjajo, zato so primerni predvsem kot hitro dostopna orodja, ki nadgradijo posamezno spletno stran ali razširijo zmogljivost brskalnika.

---

[2] Terminološki slovar informatike Islovar (http://islovar.org/) kot poslovenjen izraz angleškega izvirnika bookmarklet predlaga aktivni zaznamek, a ga v svojem delu zaradi slabe razširjenosti ne uporabljam.

Bookmarklete ponavadi namestimo tako, da obiščemo spletno stran, ki ponuja URL s programsko kodo v JavaScriptu, in slednjega dodamo med zaznamke z desnim klikom ali kakšnim drugim ukazom, ki je odvisen od brskalnika. Druga možnost je, da v brskalnik dodamo nov zaznamek, kot URL navedemo programsko kodo v JavaScriptu in shranimo zaznamek.

Uporabljamo jih kot vse ostale zaznamke, le da se po kliku ne odpre nova stran, ampak ostanemo na obstoječi. Ena izmed glavnih pomanjkljivosti bookmarkletov v primerjavi z večino ostalih tehnologij za nadgrajevanje spletnih strani je prav zahteva po kliku pred izvedbo kode, ki je nikakor ne moremo zagnati samodejno, ko se naloži sama spletna stran.

Tudi to je eden izmed razlogov, zakaj bookmarkleti kakšnega večjega razcveta nikoli niso dosegli, čeprav jih poznamo že skoraj 15 let. Tudi spletna stran bookmarklets.com, ki naj bi bila zbirališče bookmarkletov, se lahko pohvali z le nekaj več kot 150 bookmarkleti v svoji ponudbi. Med njimi najdemo bookmarklete, ki spreminjajo izgled spletnih strani, pridobivajo podatke iz strani, izvajajo iskanje nad izbranimi besedami, pošiljajo vsebino zunanjim storitvam (npr. prevajalnikom) in nastavljajo pogoste nastavitve.

Slika 1: Primer bookmarkleta, ki v pogovornem oknu izpiše število besed na trenutni spletni strani. Na sliki je brskalnik Firefox 3.0 v Ubuntu Linuxu. Izvorna koda:

```
javascript:alert("There%20are%20about%20"+document.body.textContent.split(/[\s./\u2000-\u206F\u2E00-\u2E7F\u3000-\u303F]+/).length+"%20words%20on%20this%20page.")
```

2.2 Firebug

Firebug je razširitev za brskalnik Firefox, ki ni tipičen predstavnik tehnologij za nadgrajevanje spletnih strani, saj z njim ne moremo doseči trajnih sprememb na spletnih straneh. A ker velja za de facto standard na področju orodij za spletni razvoj, o njegovi priljubljenosti pa priča tudi okoli 20 milijonov prenosov [7], ga mora vsak spletni razvijalec dobro poznati. Še posebej, če se ukvarja z nadgrajevanjem strani.

Med drugim na enem mestu ponuja urejanje, razhroščevanje in nadziranje programske kode v jezikih CSS, HTML in JavaScript. Vse to lahko počnemo "v živo", tako da so spremembe takoj vidne na spletni strani. A če smo kot eno izmed glavnih pomanjkljivosti bookmarkletov navedli nezmožnost samodejnega izvajanja kode ob nalaganju strani, se ta pomanjkljivost v še večji meri odraža v Firebugu. Potem ko stran ponovno naložimo, so vse spremembe izgubljene.

Med razvojem spletišč je seveda povsem dovolj, da razvijalec npr. s spremembo vrednosti enega izmed parametrov CSS preveri, kako se ta sprememba pozna na strani. V prihodnjih različicah Firebuga pa se lahko nadejamo tudi možnosti shranjevanja sprememb v datoteke CSS, HTML in JavaScript.

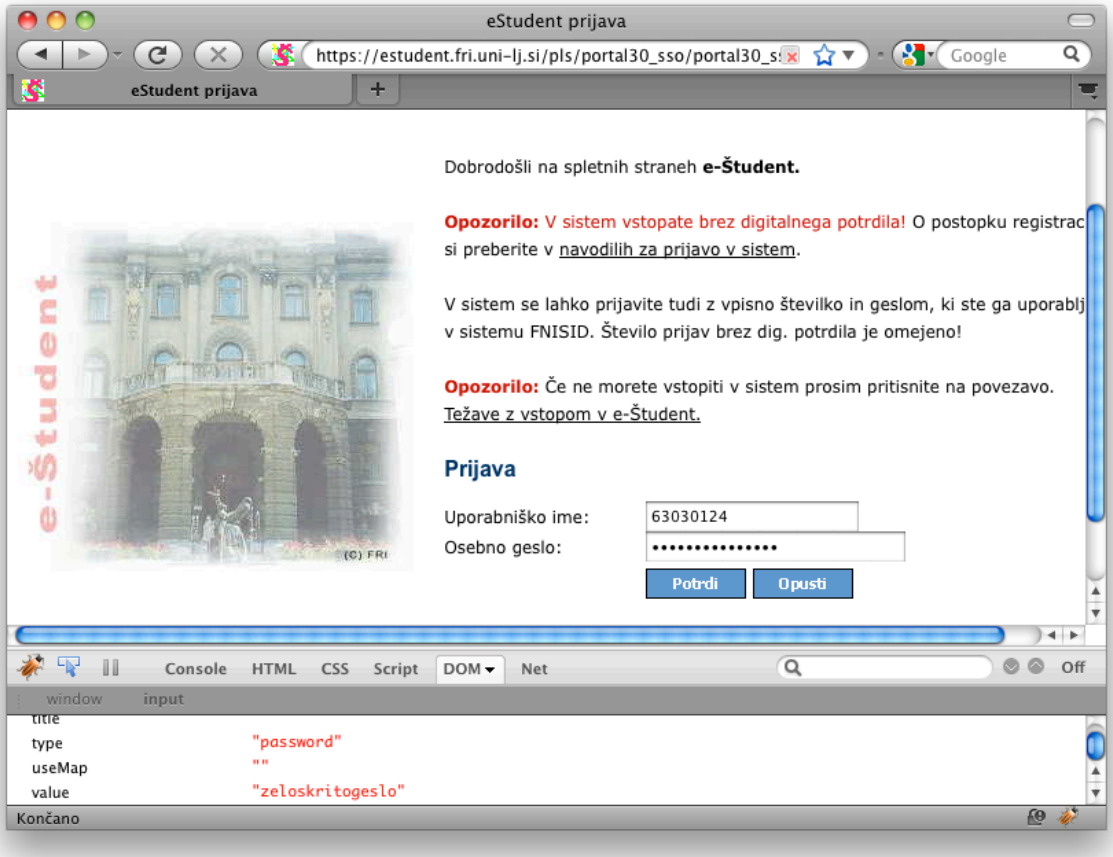
Kaj vse zmore Firebug? Med uporabniki je najbolj razširjenja možnost pregledovanja elementov HTML, saj nam Firebug zgolj s klikom nanje pokaže, kje v izvorni kodi se nahajajo. Grafično označi tudi vse elemente ali dele elementov, ki jih spreminja JavaScript, omogoča pa tudi neposredno urejanje kode HTML, vključno z dodajanjem in brisanjem elementov.

Za vsak element HTML si lahko ogledamo tudi parametre CSS, ki so nastavljeni v različnih datotekah, spreminjamo lahko njihove vrednosti ter dodajamo in odstranjujemo obstoječe parametre. Vse te spremembe so takoj vidne na spletni strani. Velikokrat nam Firebug pomaga tudi pri določanju položaja elementov, saj na pregleden način prikazuje odmike, robove in okvire, hkrati pa si lahko pomagamo tudi z navideznimi ravnili.

Kot smo že omenili, Firebug služi tudi nadziranju, saj lahko v grafično bogatem prikazu spremljamo nalaganje vsake datoteke, ki sestavlja spletno stran. To je še posebej uporabno, če želimo ugotoviti razloge za počasno nalaganje strani in poiskati ukrepe za optimizacijo. Pri vsaki zahtevi si lahko ogledamo glavo HTTP, za vsak zahtevek XMLHttpRequest pa nam Firebug pokaže tako besedilo, ki se je poslalo na strežnik, kot tudi odgovor, ki smo ga prejeli.

Document Object Model si lahko predstavljamo kot hierarhično urejene objekte in funkcije, ki jih spreminjamo z JavaScriptom. Firebug omogoča hitro iskanje objektov DOM in njihovo urejanje v živo, kjer nam je v veliko pomoč samodejno dopolnjevanje vrednosti.

Zelo zmogljiva je tudi Firebugova konzola, ki podrobno prikazuje napake v izvajanju JavaScripta, CSS in XML. Dopolnjuje jo ukazna vrstica, s pomočjo katere lahko na trenutni strani izvajamo poljubno kodo v JavaScriptu, ki jo lahko s klikom na Copy skopiramo v odložišče v obliki bookmarkleta. Razhroščevalnik JavaScripta omogoča prekinjanje izvajanja in pregled vrednosti spremenljivk ob poljubnem času, za nadzor zmogljivosti in odkrivanje ozkih grl pa je na voljo tudi profilirnik (profiler) JavaScripta.



Slika 2: Firefox z vključenim Firebugom, ki ima odprt zavihek DOM. Kljub temu da brskalnik ne prikazuje gesla ob prijavi v sistem e-Študent, ga lahko s Firebugom zlahka odkrijemo.

O razširjenosti Firebuga priča tudi dejstvo, da si lahko namestimo celo razširitve zanj, torej razširitve za razširitev za Firefox. Na voljo jih je več deset, najbolj znan med njimi pa je Yahoojev YSlow, ki analizira spletne strani in na osnovi Yahoojevih standardov zmogljivosti prikazuje razloge za počasnost.

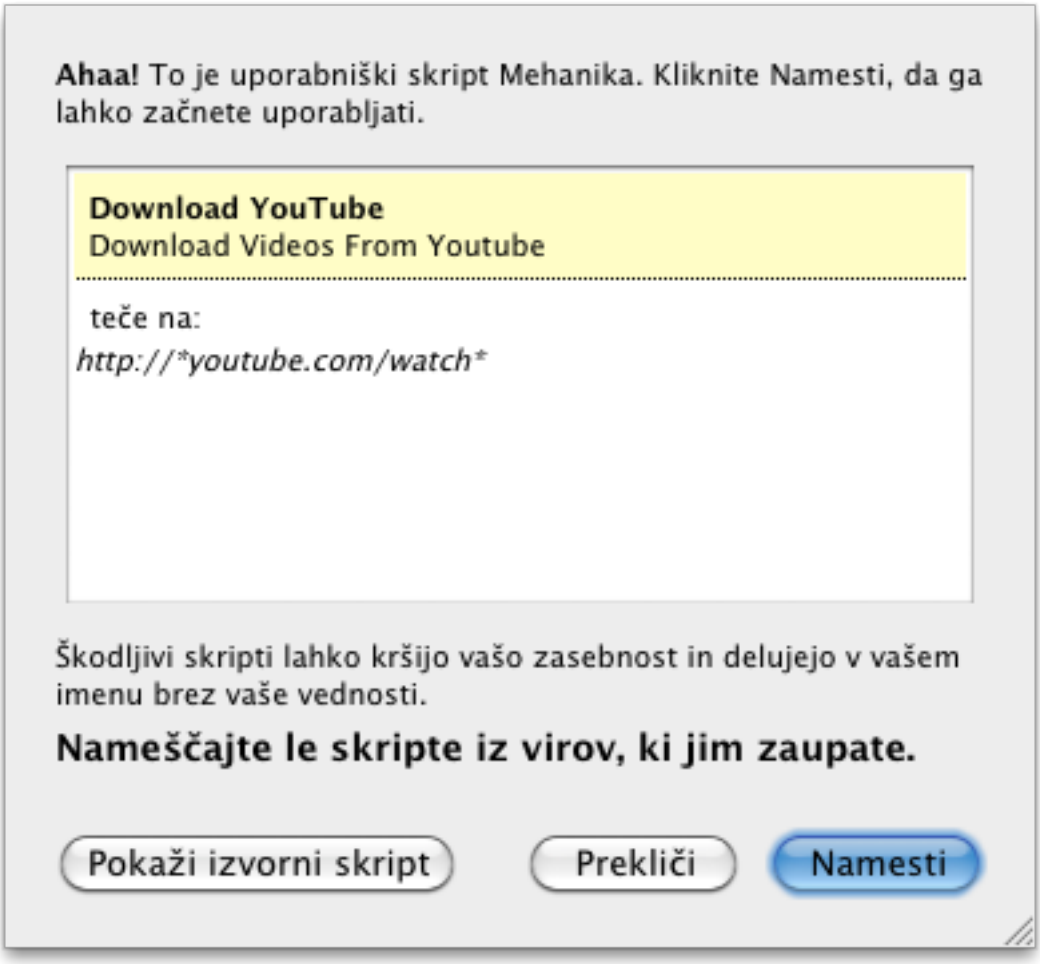
V okviru projekta Firebug poteka tudi razvoj Chromebuga, ki služi razvijalcem razširitev za Firefox, o katerih bomo več povedali kasneje. Nekoliko manj pozornosti pa je zadnje čase deležen še drugi sestrski projekt - Firebug Lite, ki poskuša čim večji del zmogljivosti Firebuga ponuditi izključno v obliki JavaScripta, da ga lahko uporabimo tudi v drugih brskalnikih.

Slednji so bili namreč dolgo časa brez primernih orodij za spletne razvijalce, a se na srečo stvari premikajo na bolje. Brskalniki, ki temeljijo na WebKitu (predvsem Applov Safari in Googlov Chrome) imajo vgrajeno zbirko orodij Web Inspector, ki omogoča ogled izvorne kode, hierarhije DOM, razhroščevanje skript, profiliranje in drugo. Podobno orodje je tudi Opera Dragonfly, vendar tako kot Internet Explorer Developer Toolbar za Internet Explorer 6 in 7 precej zaostaja za Firebugom. Internet Explorer 8 je na voljo skupaj z razhroščevalnikom JScript, ki je primerljiv z Web Inspectorjem.

2.3 Greasemonkey

Zapisali smo že, da se je nadgrajevanje spletnih strani začelo množično širiti leta 2005, ko je Google predstavil Greasemonkey. Ko razširitev namestimo v brskalnik Firefox, potrebujemo še vsaj eno uporabniško skripto. Slednja spreminja strani HTML ob dogodku DOMContentLoaded, ki se sproži ob koncu nalaganja dokumentove vsebine DOM. Pomembna razlika med dogodkom DOMContentLoaded in onLoad je, da se prvi lahko sproži še preden so naložene vse slike, drugi pa šele ob koncu nalaganja slik.

Kot smo že zapisali, je na bookmarklets.com zbranih okoli 150 bookmarkletov, in čeprav jih je v resnici na internetu raztresenih bistveno več, so Greasemonkeyjeve uporabniške skripte neprimerljivo bolj razširjene. Najdemo jih na strani userscripts.org, njihovo skupno število pa presega mejo 40.000. Namestimo jih z enim klikom, vmes se iz varnostnih razlogov pojavi pogovorno okno s podatki o skripti, ogledamo pa si lahko celo izvorno kodo (Slika 3).



Slika 3: Preden namestimo uporabniško skripto za Greasemonkey, se nam predstavi z imenom, opisom in masko podprtih spletnih naslovov. Za napredne uporabnike je na voljo tudi ogled izvorne kode skripte. Primer na sliki prikazuje namestitev skripte za prenos videov iz YouTuba na računalnik.

Greasemonkey odpravlja že večkrat omenjeno pomanjkljivost bookmarkletov in Firebuga, ki ne omogočata samodejnega zaganjanja kode, ko se naloži spletna stran. Ko skripto enkrat naložimo, se nadgradnje spletne strani izvedejo vsakič, ko se stran naloži. Greasemonkey lahko uporabimo za dodajanje novih zmogljivosti spletnim stranem (kot je zgoraj prikazani prenos videov iz YouTuba), združevanje podatkov z več strani, odpravljanje napak pri izrisovanju strani, odstranjevanje neželenih elementov (npr. oglase ali rezultate reševanja kvizov na Facebooku) in za mnoga druga opravila.

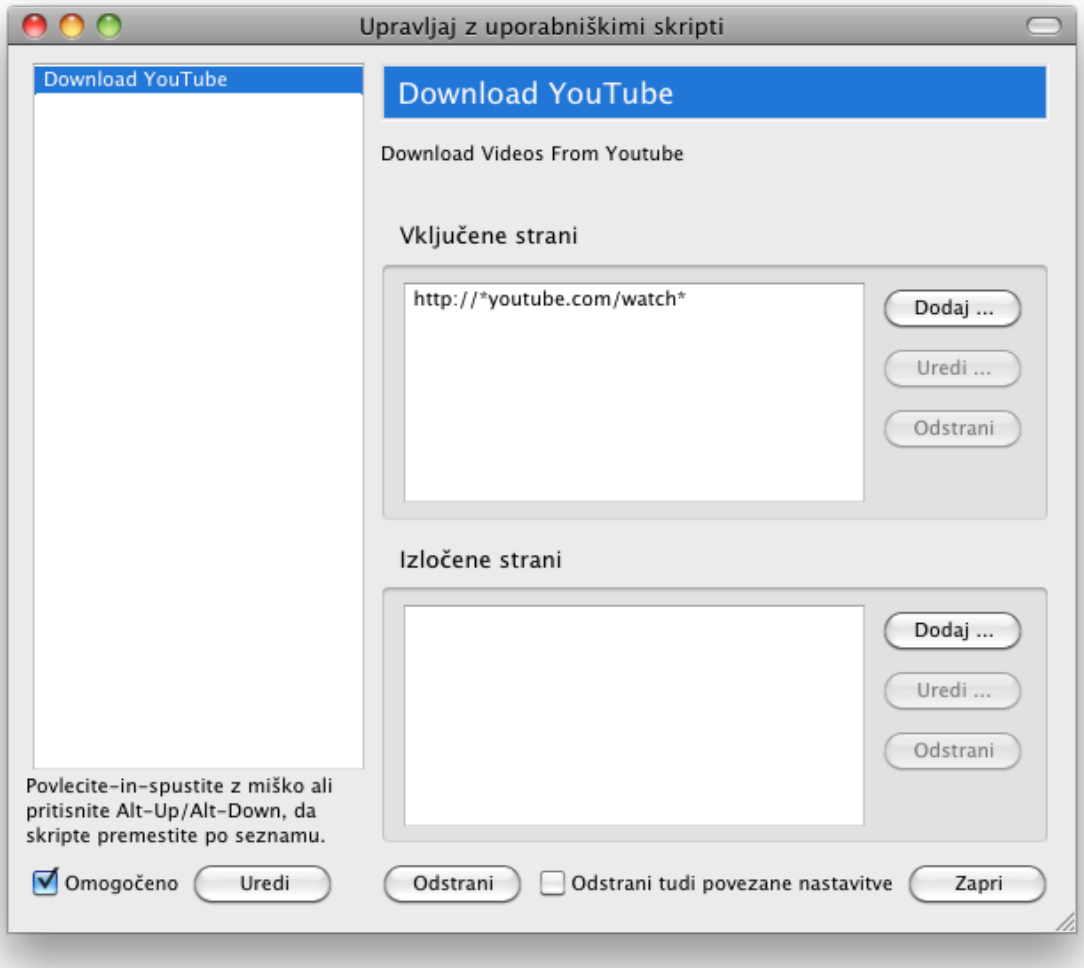
Vsaka skripta ima v glavi poleg imena, opisa, avtorja in nekaterih drugih podatkov obvezno tudi masko, s katero določimo naslove spletnih strani, na katerih se bo skripta izvajala (parameter @include). V telesu sporočila pa se nahaja programska koda v JavaScriptu z nekaterimi omejitvami, ki se izvede na vnaprej določenih straneh.

```
// ==UserScript==
// @name                Download YouTube
// @namespace           DownloadYoutube
// @description         Download Videos From Youtube
// @version             1.0
// @date                2009-05-10
// @creator             downloadmp4.info
// @include             http://*youtube.com/watch*
// ==/UserScript==

// JavaScript code is placed here
```

Slika 4: Ob naslovu videoposnetka na YouTubu je dodana povezava za prenos videa na računalnik. Video se nahaja na naslovu: http://www.youtube.com/watch?v=KIV4FUwLi7g

Uporabniške skripte imajo imena oblike imeskripte.user.js, zato Greasemonkey ob zahtevku za URL, ki se konča v taki obliki, ponudi namestitev skripte (Slika 3). Greasemonkey vsebuje tudi upravljalca skript, s katerim lahko dodajamo vključene in izključne strani, na katerih se skripta izvede oz. ne, ter spreminjamo izvorno kodo.

V slednji se lahko poleg običajnih lastnosti JavaScripta poslužujemo tudi nekaterih dodatnih zmogljivosti, med katerimi je zahtevek XMLHttpRequest iz drugega izvora (cross-site). Pomembno je tudi upoštevati, da se uporabniške skripte za Greasemonkey zaženejo ob vsakem primerku spletne strani. Zato je oteženo globalno upravljanje seznamov, kar lahko rešimo s piškoti ali Greasemonkeyjevimi klici GM_getValue in GM_setValue.



Slika 5: Upravljanje z uporabniškimi skriptami za Greasemonkey

Greasemonkey ni omejen samo na Firefox; deluje še v dveh brskalnikih družine Mozilla - Flock in SeaMonkey, pa tudi v glasbenem predvajalniku Songbird. Deloma je podprt v Epiphanyju, brskalniku namiznega okolja GNOME, skripte pa lahko poganjamo tudi v Operi od različice 8 naprej in Internet Explorerju z nameščeno razširitvijo IE7Pro. Za uporabo v Safariju in nekaterih drugih na WebKitu temelječih programih moramo namestiti razširitev GreaseKit, podpora za Googlov Chrome pa je še v razvoju.

2.4 Razširitev za brskalnik

Ključ do množičnega uspeha Greasemonkeyja je v uporabi programskega jezika JavaScript, ki ga obvladajo vsi spletni razvijalci, in odprava pomanjkljivosti nadgrajevanja strani z bookmarkleti.

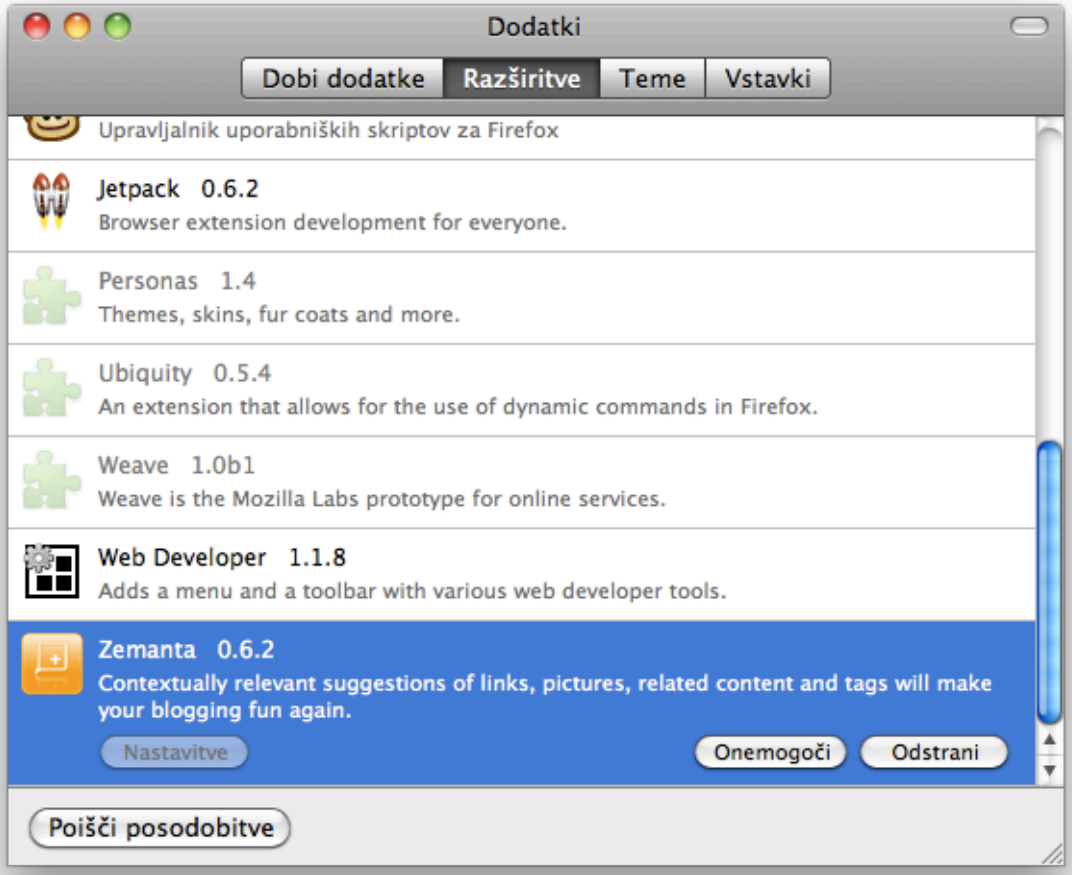
Toda število prenosov Greasemonkeyja za Firefox je okoli 25 milijonov [8], kar pomeni, da ima ta razširitev bistveno manj uporabnikov kot sam Firefox, ki jih šteje kar 330 milijonov [9]. Pri Internet Explorerju in ostalih brskalnikih s pomembnim tržnim deležem je razmerje med številom vseh uporabnikov in uporabnikov s podporo za izvajanje uporabniških skript še večje. Firefox ima namreč daleč najbolje razvito infrastrukturo za razvoj razširitev, saj se je pri vseh ostalih brskalnikih pri razvoju razširitev potrebno posluževati stranskih poti. In razširitve so nujno potrebne, če želimo zagotoviti izvajanje uporabniških skript (razen pri Operi in predvidoma v Chromu).

Število uporabnikov z možnostjo izvajanja uporabniških skript je torej majhno v primerjavi s celotnim številom uporabnikov interneta. Če želimo svojo nadgradnjo spletnih strani ponuditi vsem, jo moramo izdelati kot samostojen programski paket. To nam omogočajo razširitve za brskalnike, ki prinašajo še eno pomembno prednost: uporabniki za namestitev namesto dveh korakov (namestitev Greasemonkeyja in nato uporabniške skripte) potrebujejo le enega (namestitev razširitve).

A tudi odločitev za razširitve ni optimalna. Prvo težavo smo že omenili: kakovostno infrastrukturo za razvoj razširitev za brskalnik ponuja le Firefox, ki obvladuje zgolj četrtino trga brskalnikov [10]. Pri drugih brskalnikih je zato velikokrat potrebno zaobiti varnostni sistem oz. hekati, dodaten problem pa predstavlja tudi pomanjkanje dokumentacije. Te je pri Firefoxu sicer dovolj, vendar za razvoj razširitev ne zadošča poznavanje progamskih jezikov spletni razvoj, kot pri Greasemonkeyju, ampak je potrebno obvladati tudi tehnologije XPCOM, XPConnect, XPI in XUL.

Slika 6: Razmerje med brskalniki z infrastrukturo za razširitve in brez znaša 1: 3.

Strnemo lahko, da imajo razširitve za brskalnik prednosti v primerjavi z Greasemonkeyjem in uporabniškimi skriptami predvsem za končne uporabnike, pomanjkljivosti pa najbolj občutijo razvijalci.



Slika 7: Firefox vsebuje tudi posebno pogovorno okno za upravljanje razširitev, kjer lahko razširitve poiščemo, namestimo, odstranimo, vključimo ali izključimo, v nekaterih primerih pa tudi nastavljamo.

Seveda pa je ključna razlika med obema tehnologijama v njunem dosegu: Greasemonkey je iz varnostnih razlogov omejen skoraj izključno na posege v spletne strani, razširitve pa lahko spreminjajo tudi brskalnikovo kramo, tj. uporabniški vmesnik (orodne vrstice, menije, stransko vrstico, pogovorna okna itd.). Varnostna omejitev je v Firefoxu implementirana s tehnologijo XPCNativeWrapper, ki "ovije" objekte in dovoli dostop samo do izbranih lastnosti in metod, do vseh ostalih pa lahko dostopamo le iz priviligirane kode.

Kako priljubljene so razširitve? O tem pričata naslednja podatka: število prenosov vseh razširitev za Firefox skupaj s temami doslej znaša 1,7 milijarde, v uporabi pa je trenutno 175 milijonov razširitev [11].

2.5 Jetpack

Prednost razvoja uporabniških skript za Greasemonkey v primerjavi z razvojem razširitev za brskalnik je nedvomno v lažjem razvoju, četudi se omejimo zgolj na razširitve za Firefox, za razvoj katerih hekanje ni potrebno. Zato skupnost Mozilla, ki razvija Firefox, že pripravlja naslednjo generacijo razširitev, ki jih bodo lahko razvijali vsi spletni razvijalci, saj bo zadoščalo poznavanje jezikov JavaScript, HTML in CSS.

Razširitve 2.0 ali lahke razširitve so na voljo že danes, le da moramo Firefox najprej nadgraditi z razširitvijo Jetpack. Od Firefoxa 4.0 naprej, ki naj bi izšel konec leta 2010, pa bo ta zmogljivost vgrajena že v sam brskalnik. Dokončna odločitev sicer še ni sprejeta, vendar naj bi po sedanjih načrtih Mozilla kmalu zatem začela opuščati podporo za klasične razširitve, kot jih poznamo danes.

Izraza Jetpack ne uporabljamo samo za razširitev, ki jo moramo namestiti za uporabo lahkih razširitev, ampak tudi za tehnologijo samo ter za posamezne razširitve, napisane v tej tehnologiji. Poleg spletnih programskih jezikov JavaScript, HTML in CSS se moramo za pisanje Jetpackov naučiti samo še razmeroma preprostega programskega vmesnika, poslužujemo pa se lahko celo programskega ogrodja jQuery.

2.5.1 jQuery

jQuery 1.3.2 obsega manj kot 20 kB, z njegovo pomočjo pa je pisanje JavaScripta bistevno poenostavljeno. Podpira izbirnike elementov jezikov CSS 1-3, spreminjanje in prečkanje DOM, upravljanje z dogodki, delo z atributi, spreminjanje CSS, vizualne učinke, animacije, Ajax in različne pripomočke, kot je prepoznava uporabnikovega brskalnika.

Velja za najbolj razširjeno programsko ogrodje za JavaScript med spletnimi razvijalci, k čemur je v veliki meri pripomoglo tudi enakovredno delovanje v brskalnikih Internet Explorer 6.0+, Firefox 2+, Safari 3.0+, Opera 9.0+ in Chrome. jQuery UI, ki ni del Jetpacka, dopolnjuje osnovno programsko ogrodje z dodatnimi zmogljivostmi za razvoj interaktvnih spletnih aplikacij s poudarkom na interakciji z uporabniškim vmesnikom, naprednih učinkih in animacijah.

jQuery vsebuje dva načina uporabe. Prvi poteka preko funkcije $, s katero ustvarimo objekt tipa jQuery. Tem funkcijam pravimo tudi ukazi in jih lahko verižimo, saj vsaka izmed njih vrača objekt tipa jQuery. Pogosto jih uporabljamo pri delu z vozlišči DOM. Spodnji primer s funkcijo $ in izbirnikom CSS ustvari objekt tipa jQuery, ki se sklicuje na nič ali več značk tipa div na strani HTML, ki so razreda “test”. Doda jim razred “blue” in jih odkrije s počasno animacijo.

```
$("div.test").addClass("blue").slideDown("slow");
```

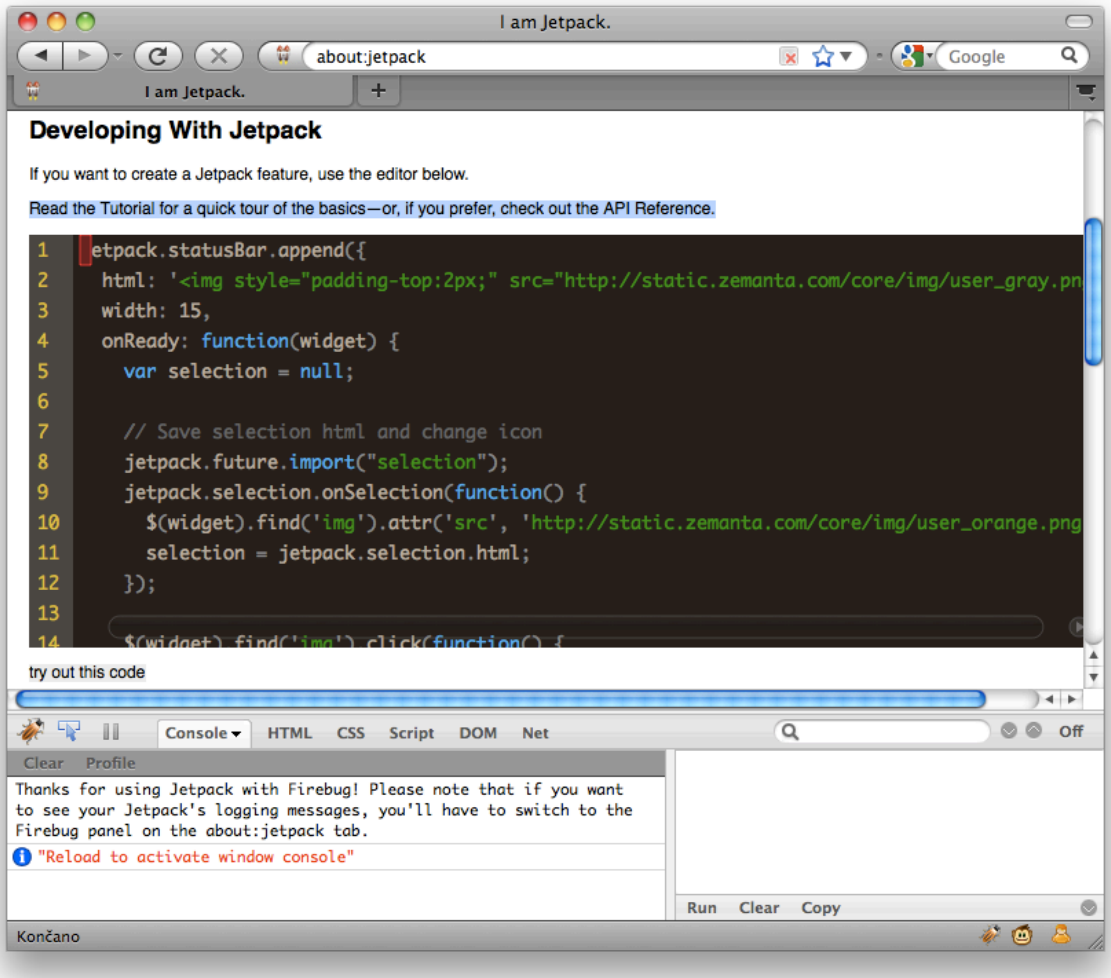
Drugi način poteka preko funkcij s predpono $., ki služijo kot pripomočki in ne delujejo na samih objektih tipa jQuery. Spodnji primer v dokument zapiše število 234.

```
$.each([1,2,3], function() {
      document.write(this + 1);
});
```

2.5.2 Razvoj in uporaba Jetpackov

Ko imamo dovolj znanja JavaScripta, HTML, CSS in jQueryja, se lahko lotimo razvoja Jetpackov. Začnemo z namestitvijo razširitev Jetpack in Firebug v Firefox in obiskom strani about:jetpack. Na začetku je zelo priporočljivo prebrati vsebino v zavihku Tutorial, ki skozi preproste primere razloži bistvo razvoja Jetpackov. V zavihku API Reference je predstavljen programski vmesnik Jetpacka, v zavihku Develop pa lahko začnemo s programiranjem.

Osrednji del zaseda vgrajeni spletni urejevalnik za programiranje Bespin (bespin.mozilla.com), ki je sicer samostojni projekt Mozilla Labs. Njegov osnovni namen je izdelati spletni urejevalnik za programerje, ki je dostopen iz različnih računalnikov in posledično omogoča tudi skupinsko delo. Podpira označevanje kode za spletne programske jezike in uporablja tehnologijo HTML 5, predvsem element canvas.

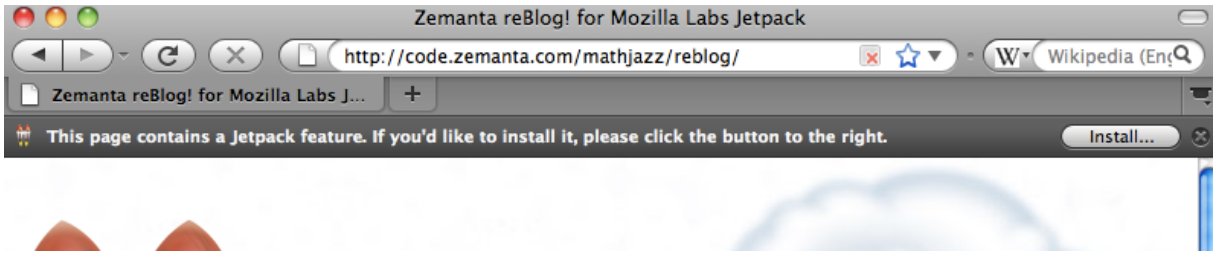


Slika 8: Okolje za razvoj Jetpackov na strani about:jetpack. V osrednjem delu je spletni urejevalnik Bespin s programsko kodo, ki se izvede ob kliku na "try out this code". V spodnjem delu okna vidimo Firebug z vključeno konzolo, v kateri se izpisujejo morebitne napake, skrajno desno v vrstici stanja pa je ikona, ki jo je vstavil Jetpack, ki ga trenutno razvijamo.

Ko dokončamo z razvojem Jetpacka, ga lahko ponudimo uporabnikom. To lahko storimo na spletni strani jetpackgallery.mozillalabs.com, ki je čisto sveže zbirališče Jetpackov, podobno strani Add-ons for Firefox za klasične razširitve. Uporabniki namestitev izvedejo s preprostim klikom, uporabljati pa ga lahko začnejo takoj. Druga možnost je, da Jetpack ponudimo na lastni spletni strani preko značke link v glavi.

```
<html>
	<head>
		<title>My First Jetpack</title>
		<link rel="jetpack" href="my-first-jetpack.js"/>
	</head>
	<body>This is my first Jetpack!</body>
</html>
```

Vsi uporabniki, ki imajo nameščeno razširitev Jetpack (od vključno Firefoxa 4.0 naprej to seveda ne bo več potrebno), bodo ob obisku naše strani prejeli obvestilo, da je na strani na voljo Jetpack in da ga lahko namestijo s preprostim klikom na gumb.



Slika 9: Primer obvestila na strani, ki vsebuje Jetpack.

Lahke razširitve se od običajnih razlikujejo tudi po tem, da jih lahko začnemo uporabljati takoj po namestitvi, brez predhodnega ponovnega zagona brskalnika. Ta ni potreben niti po vklopu ali izklopu posameznega Jetpacka, ampak se sprememba uveljavi takoj.

Poleg tega naj bi Jetpack končno odpravil tudi nevšečnosti z zastarelimi razširitvami. Doslej je namreč veljalo, da je ob vsaki novi veliki različici Firefoxa bilo potrebno počakati, da so tudi avtorji razširitev pripravili nove različice, ki so delovale z novim Firefoxom. Ko bo Jetpack postal del Firefoxa, takšnih težav naj ne bi bilo več, tako da bodo lahko uporabniki tudi hitreje nadgradili na sodobnejši spletni brskalnik.

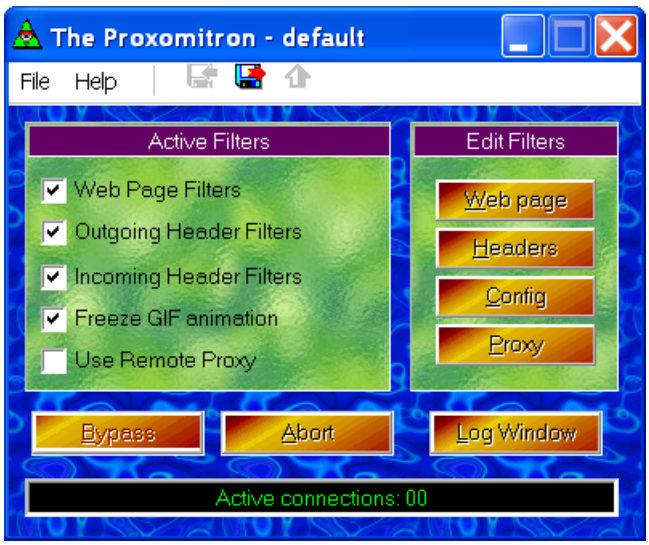
Predvidevamo lahko, da je prihodnost razvoja razširitev za Firefox in tudi nadgrajevanja spletnih strani v Jetpacku in v Jetpacku podobnih tehnologijah, če jih bodo razvili tudi ostali brskalniki. Google že pripravlja podobno rešitev za Chrome, ki temelji na Greasemonkeyju in bo tako kot Jetpack znala dostopati tudi do krame brskalnika.

Jetpack je sicer še vedno v intenzivnem razvoju, zato lahko zaenkrat dostopa zgolj do nekaterih delov brskalnika, npr. do vrstice stanja, drsne stranske vrstice in menija, prikazuje pa lahko tudi sistemska obvestila ter bere in piše v odložišče.

2.6 Druge tehnologije

Predstavili smo le najbolj razširjene tehnologije in programska orodja za nadgrajevanje spletnih strani. Če bi želeli enako podrobno predstaviti tudi vse ostale, bi presegli obseg diplomskega dela, zato smo izbrali nekatere izmed njih, ki jih bomo na hitro preleteli.

Najstarejši predstavnik je Proxomitron, ki je na voljo že od leta 1999, ko se je JavaScript šele dobro uveljavljal. Zasnovan je bil za Windows 95, vendar deluje tudi na novejših različicah Microsoftovega operacijskega sistema, čeprav se je njegov razvoj končal že leta 2003. Služi predvsem preprečevanju odpiranja neželenih oken in oglasov, odstranjevanju v spletne strani vgrajenih zvokov in animacij, spreminjanju in onemogočanju JavaScripta, pa tudi prilagojevanju izgleda in vsebine spletnih strani.



Slika 10: Proxomitron, univerzalni spletni filter

Proxomitron je samostojni program, ki prestreza promet med poljubnim brskalnikom in spletnimi strežniki. Uporablja poseben jezik za ujemanje, ki je podoben regularnim izrazom, s katerim spreminja vsebino spletnih strani, tako da dodaja, odstranjuje ali spreminja besedilo.

Podobnih rešitev, ki delujejo kot posredni strežniki na strani odjemalca, je veliko. Prva med njimi je Proximodo, ki poskuša nadaljevati razvoj Proxomitrona. Čeprav je program napisan na novo, je združljiv s konfiguracijskimi datotekami (filtri) svojega predhodnika. Na voljo so še v Rubyju napisan MouseHole, javanski Muffin, BFilter in Privoxy, ki podpirata različne operacijske sisteme, in drugi.

Tudi na drugi strani, torej na spletnem strežniku, lahko uporabimo posebno programsko opremo za nadgrajevanje spletnih strani. Seveda se takoj pojavi vprašanje, zakaj bi to počeli, saj v tem primeru potrebujemo skrbniški dostop in lahko spremenimo samo stran. Izkaže se, da imamo velikokrat opravka z več spletnimi stranmi, ki bi jih radi na enak način posodobili, zato iščemo neko sistemsko rešitev. To nam med drugim omogoča Monkeygrease.

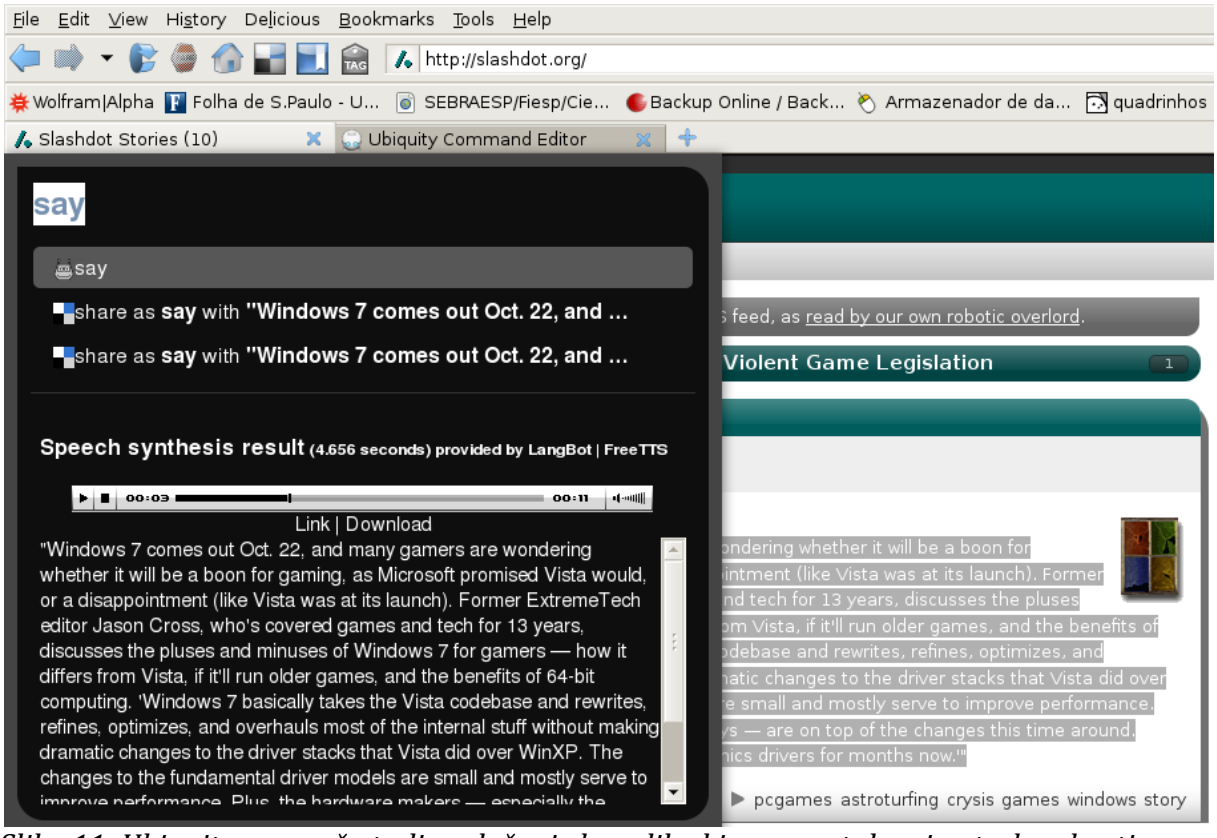
Precej drugačna rešitev je Stylish. Gre za še en dodatek za družino Mozillinih brskalnikov in Thunderbird, ki s pomočjo jezika CSS spreminja izgled spletnih strani in uporabniških vmesnikov aplikacij, napisanih v tehnologiji XUL. Za razliko od Greasemonkeyja namesto uporabniških skript uporablja uporabniške sloge, ki jih najdemo na strani userstyles.org.

Tudi tehnologija iMacros je v primerjavi z ostalimi rešitvami za nadgrajevanje spletnih strani povsem samosvoja. Dodatek za brskalnike Firefox, Chrome in Internet Explorer prinaša možnost snemanja in ponovnega predvajanja zmogljivosti, kot jo poznamo v orodjih za testiranje aplikacij. Za nadzor nad makroji skrbi JavaScript, zanimiva pa je tudi možnost izmenjave makrojev in skript s prijatelji.

2.6.1 Ubiquity

Če si katera izmed doslej še nepredstavljenih tehnologij zasluži več pozornosti od ostalih, je to prav gotovo Ubiquity. Projekt se je iz Mozillinih laboratorijev javnosti prvič pokazal avgusta 2008 kot poskus ponovne uporabe ukazne vrstice v sodobnih uporabniških vmesnikih, ki naj bi uporabnikom prihranil precej časa.

Vsebuje okoli 80 ukazov za iskanje, prevajanje, pošiljanje elektronske pošte, iskanje zemljevidov itd., hkrati pa ponuja tudi programski vmesnik za razvoj lastnih ukazov. Številni med njimi služijo tudi nadgrajevanju spletnih strani, npr. že omenjeno prevajanje, saj lahko označimo poljuben del besedila na strani, uveljavimo ukaz za predvajanje in po potrditvi je izbrani del strani preveden v ciljni jezik.



Slika 11: Ubiquity omogoča tudi poslušanje besedila, ki nam ga tako ni potrebno brati

3 Pogosti problemi

Tehnologije in programsko opremo, ki smo jih spoznali v drugem poglavju, mora spletni razvijalec za nadgrajevanje spletnih strani nujno obvladati. Seveda pa je obvezno tudi poznavanje osnovnih spletnih programskih jezikov, ki se izvajajo na strani odjemalca: HTML za določanje semantične strukture in hierarhije dokumenta, CSS za nastavitev izgleda in postavitve ter JavaScripta za izvedbo interaktivnosti.

Zato si bomo v nadaljevanju ogledali nekaj ključnih pasti, na katere mora biti pozoren razvijalec pri nadgrajevanju spletnih strani. Predstavili bomo težave pri uporabi jezikov CSS in JavaScript, medtem ko uporaba HTML pri nadgrajevanju ne prinaša dodatnih nevšečnosti v primerjavi s samim razvojem spletnih strani. Poleg tega vlogo določanja vsebine in strukture dokumentov pri zahtevnejših nadgradnjah opravlja JavaScript, kar bomo podrobno prikazali v četrtem poglavju.

3.1 CSS

Namen HTML je določanje vsebine dokumenta, torej ne potrebuje značk za oblikovanje. Zato je postala specifikacija HTML 3.2 nočna mora za spletne razvijalce, saj je uvedla značke, kot je font (pisava), in lastnosti, kot je color (barva). Razvoj obsežnih spletišč, kjer je bilo potrebno vsaki strani dodati podatke o barvah in pisavah, je namreč postal dolgotrajen in drag proces. Zato je W3C ustvaril CSS in od vključno HTML 4.0 lahko vse podatke o oblikovanju odstranimo iz dokumenta HTML in jih shranimo v ločeno datoteko CSS. Sestavljena je iz seznama pravil naslednje oblike:

```
selector1 [, selector2, ...][:pseudo-class] {
      property: value;
      ...
}
/* comment */
```

Z izbirnikom, ki jih je lahko več, določimo seznam elementov, na katere se nanaša pravilo, nato pa določimo še vrednosti posameznim lastnostim. Izbirnike lahko dopolnimo tudi s psevdorazredi, kot je npr. :hover, s katerim pravilo omejimo samo na trenutek, ko je miškin kazalec na elementih, določenih z izbirnikom.

Datoteko CSS je potrebno povezati z dokumentom HTML, kar storimo z značko link v datoteki HTML. Velikokrat se tudi zgodi, da na obliko enega elementa vpliva več pravil, ki so lahko shranjena v različnih datotekah. In tu se srečamo z glavnim problemom pri nadgrajevanju spletnih strani, povezanih s CSS.

Izbirniki spletnih strani (lokalni) velikokrat ustrezajo tudi elementom, ki jih v dokument HTML vstavimo pri nadgrajevanju, zato vplivajo na njihov izgled. Če želimo enako nadgradnjo ponuditi na več spletnih straneh, bomo tako zelo verjetno potrebovali več različnih datotek CSS, saj bodo sicer nekateri elementi imeli različen slog.

Vzdrževanje različnih datotek, ki služijo enakemu namenu, še posebej če so večje, je časovno potratno. Zato si lahko pomagamo tako, da uporabimo dve datoteki CSS: splošno, ki je enaka za vse spletne strani, in specifično, ki se razlikuje od strani do strani in odpravi "napake", ki jih povzroča CSS spletne strani same.

Druga rešitev pa je, da elemente, ki jih vstavimo v dokument HTML, ovijemo v več nivojev elementov div. Tako bodo lokalni izbirniki zgrešili vse elemente, ki jih želimo prikazati na spletni strani, zato za posamezno spletno stran specifične datoteke CSS v tem primeru ne potrebujemo. Primer strukture datoteke HTML in verige izbirnikov v CSS je prikazan spodaj:

```
<div id="app1">
      <div id="app2">
            <div id="app3">
                  <!-- Here starts and ends the actual content -->
            </div>
      </div>
</div>

#zem1 #zem2 #zem3 h3 {

}
```

3.2 Razširjanje objektov Object in Array

JavaScript je objektno usmerjen programski jezik, ki temelji na prototipih. To pomeni, da ne uporablja razredov, zato spreminjanje in dodajanje lastnosti ter metod objektov poteka s kloniranjem obstoječih objektov, ki služijo kot prototipi (objekt.prototype).

Spodnji primer prikazuje dodajanje lastnosti populationCount konstruktorju objekta Person s pomočjo prototipa in njegovo spreminjanje med izvajanjem kode, ki se konča z izpisom "Število ljudi v mojem svetu: 2."

```
Person.prototype.populationCount=0;
function Person(name, sex){
      Person.prototype.populationCount++;
      this.getName=function(){ return name }
      this.getSex=function(){ return sex }
}
var gk = new Person('Janez', 'moški');
var lrk = new Person('Marija', 'ženski');
alert("Število ljudi v mojem svetu: " + gk.populationCount + ".");
```

V JavaScriptu so vsi objekti dediči objekta Object, zato podedujejo vse lastnosti in metode njegovega prototipa Object.prototype (seveda jih je kasneje moč tudi povoziti). Zato je še toliko bolj nevarno razširjati objekt Object, saj to lahko vpliva na delovanje kode na številnih mestih. Pojav je še posebej neprijeten pri nadgrajevanju spletnih strani, saj se razvijalec nadgradnje pri razvoju svoje kode tega velikokrat ne zaveda in mora svojo kodo naknadno prilagajati spremembam.

Oglejmo si primer preproste programske kode v JavaScriptu, ki deluje različno, če je objekt Object razširjen ali ne.

```
var obj = {a: "A", b: "B", c: "C", d: "D"};
for (var key in obj) {
      doSomething(key, obj[key], obj);
}

if ("b" in obj) {
      doSomethingElse();
}
```

Če je bil Object.prototype spremenjen, bi se zanka sprehodila tudi skozi na novo dodane lastnosti in bi tako delovala drugače, kot v primeru izvirnega objekta Object. Zato si velja zapomniti pravilo, da objektov Object ne razširjamo, če le želimo da naša stran omogoča nadgrajevanje.

Enaka prepoved velja tudi za objekt Array, ki ga tako kot Object pogosto uporabljajo programska ogrodja za JavaScript (tudi jQuery) pri prenašanju lastnosti med funkcijami in svojevrstno shranjevanje nastavitev.

3.3 Zasebni imenski prostori

Zasebni imenski prostori so v splošnem dobra ideja. V JavaScriptu jih običajno ustvarjamo na dva načina: s pomočjo anonimnih funkcij ali s hierarhično urejenim drevesom objektov. V drugem primeru metode, ki jih želimo ohraniti zasebne, implementiramo v objektu znotraj hierarhične strukture, in tako ne umažemo globalnega imenskega prostora. Tako kot za uporabniški vmesnik je namreč tudi za globalni imenski prostor JavaScripta priporočljivo, da ne zaseda več prostora, kot je potrebno.

Po drugi strani pa nam lahko zasebni imenski prostori včasih povzročajo precejšnje težave. Če želimo uporabiti kakšno funkcionalnost spletne strani, ki je na voljo le znotraj zasebnega imenskega prostora, praktično nimamo na razpolago načina, kako dostopiti do nje.

Tak primer najdemo npr. v starejših različicah Wordpressa, natančneje v uporabniškem vmesniku za pisanje blogov. Vmesnik je sestavljen iz različnih gradnikov, ki jih lahko razporejamo po strani. Toda če želimo dodati svojega, mu zmožnosti premikanja po strani ne moremo dodati na enostaven način. Potrebne funkcije in lastnosti so namreč skrite v zasebnem imenskem prostoru, do katerega nimamo dostopa, zato nam ne preostane drugega, kot da celotno funkcionalnost nadomestimo z lastno kodo.

Včasih si lahko prihranimo precej časa, če namesto razvoja lastnih nadomestkov funkcij poslušamo zanimiv dogodke, npr. onMouseIn, onMouseOut, onMouseMove, onMouseOver, onMouseClick itd. Na ta način lahko poskušamo prestreči funkcije, ki se pokličejo ob proženju teh dogodkov.

Posebna vrsta dogodkov so mutacijski dogodki (angl. mutation events), ki se prožijo ob spremembah strukture dokumenta, npr. ob dodajanju in odstranjevanju vozlišč dokumenta ali ob spreminjanju atributov. Tudi na te dogodke lahko prežimo, vendar je njihova velika pomanjkljivost, da jih ne podpira najbolj razširjen brskalnik Internet Explorer.

V splošnem torej velja, da ustvarjalci spletnih strani lahko uporabljajo zasebne imenske prostore oz. je to velikokrat celo zaželeno. Vendar pa morajo paziti, da vanje ne shranjujejo funkcinalnosti, ki jih bodo drugi razvijalci morebiti potrebovali pri nadgrajevanju strani.

3.4 Časovna neusklajenost

Programska koda v JavaScriptu se v brskalniku izvaja v eni niti. Zato se zdi predpostavka, da je izvajanje celotne kode serializirano, na prvi pogled povsem sprejemljiva. Hkrati bi pričakovali, da se lahko v kodi zanašamo na druge dele kode, ki naj bi se v skladu z zgornjo predpostavko izvedli prej.

Takšna razlaga izvajanja programske kode v JavaScriptu je zelo preprosta, lahko razumljiva, a žal tudi povsem napačna. Drži sicer, da tolmač JavaScripta teče v eni niti, a upoštevati moramo tudi druge programske vmesnike. Čeprav gre za zelo zmogljiv programski jezik, ki je tudi izrazno dovolj močan, ima JavaScript namreč omejene zmogljivosti pri rokovanju s spletnimi stranmi.

Zato uporabljamo tudi nekatere druge programske vmesnike, ki so določeni znotraj brskalnika. Tipičen primer je DOM API, ki predstavlja javni programski vmesnik za dostop do DOM dokumenta HTML. In prav tu se skriva pomembna podrobnost – ti programski vmesniki se namreč ne izvajajo vedno v eni niti.

Oglejmo si preprost primer iz prakse, pri katerem nam napačno razumevanje izvajanja programske lahko povzroči veliko preglavic. Recimo, da želimo nekemu vozlišču naključne spletne strani dodati nekoliko bolj zapleteno vsebino, npr. neko drugo spletno stran v celoti.

Razčlenjevanje izvorne kode in vstavljanje v dokument traja nekaj časa, kar je še posebej opazno na nekoliko manj zmogljivih računalnikih. Zato brskalnik ne čaka do konca izvajanja tega opravila, ampak še pred tem nadaljuje z naslednjo vrstico v programski kodi. Tak primer je prikazan spodaj z uporabo knjižnice jQuery:

```
var element = document.getElementById('someID');
element.after('<div id="button"></div>' + someLongHTMLstring);
$('div#button').click(function () {
      // doSomething();
});
```

V praksi se tako lahko zgodi, da ob izvajanju vrstice, ki na element #button obesi dogodek click, ta element sploh še ni na voljo, in posledično se ob kliku na element ne bo zgodilo nič. Zato se je potrebno pred takšnimi težavami preventivno zaščititi, in sicer najlažje z uporabo funkcije setTimeout, ki najprej preveri, ali je ustrezen objekt ali element na voljo (v našem primeru #button), in šele nato nadaljuje z izvajanjem kode.

3.5 Sporazumevanje iz različnih izvorov

Politika enakega izvora (same origin policy) je pomemben varnostni koncept številnih programskih jezikov, ki se izvajajo na strani brskalnika, tudi JavaScripta. Dokumentom in skriptam iz enega izvora onemogoča komunikacijo z dokumenti in skriptami, ki izvirajo od drugod. Enak izvor v tem primeru pomeni enako ime domene, vrat in protokola.

Ciljni spletni naslov	Odgovor
http://store.company.com/dir2/other.html	Uspeh
http://store.company.com/dir/inner/another.html	Uspeh
https://store.company.com/secure.html	Napaka: različen protokol
http://store.company.com:81/dir/etc.html	Napaka: različna vrata
http://news.company.com/dir/other.html	Napaka: različno ime domene

Tabela 1: Kakšen bi bil odgovor politike enakega izvora, če bi s katerim izmed ciljnih spletnih naslovov poskušal komunicirati naslov http://store.company.com/dir/page.html?

Varnostni koncept, ki se je pojavil že v Netscape Navigatorju 2.0 in je bil od takrat naprej prisoten v vseh pomembnejših brskalnikih, preprečuje dve nevarnosti. Obe sta povezani s piškoti, ki hranijo podatke o sejah in uporabnikih. Prva nevarnost je oponašanje uporabnika, kar lahko napadalec doseže, če ugrabi uporabnikovo sejo in v njegovem imenu pošilja zahtevke HTTP, druga pa oponašanje spletnih strani, kar napadalcu omogoča krajo identitete s pomočjo lažnega predstavljanja (angl. phishing).

Omejitve enakega izvora zato ne veljajo povsod. V dokumentih HTML lahko namreč brez težav uporabljamo slike, prekrivne sloge (CSS) in skripte (JavaScript) iz drugih domen. Pravzaprav omejitve pri nalaganju dokumentov srečamo le ob uporabi zahtevka XmlHttpRequest.

In prav XmlHttpRequest pogosto uporabljamo pri nadgrajevanju spletnih strani, saj želimo velikokrat podatke iz spletne strani poslati na svoj strežnik ali ločeno spletno storitev, jih tam obdelati in rezultate obdelave vrniti nazaj na izhodiščno stran. Takšne nadgradnje so postale priljubljene že ob razcvetu tehnologij AJAX in Web 2.0.

Kako torej obiti zaščito, ki jo predstavlja politika enakega izvora? Dolgoročno rešitev predvideva osnutek specifikacije HTML 5, vendar so nov standard zaenkrat implementirali le najnovejši brskalniki, npr. Firefox 3.5, ki še niso dovolj razširjeni. Zato je v zadnjem času vse bolj pogosta metoda prenosa podatkov preko lastnosti name objekta window, ki jo uporabljata tudi Google in Facebook. Na voljo je tudi kot dodatek (plug-in) za različna programska ogrodja JavaScripta, zato jo razvijalci zlahka uporabijo v svoji rešitvi in je ne rabijo niti podrobno razumeti.

Deluje tako, da ustvari skriti element iframe, ki je tarča zahtevka. Če uporabljamo zahtevek tipa GET, v iframu odpre naslov s podatki v obliki niza, če uporabljamo POST, pa najprej ustvari obrazec s skritimi polji, ki predstavljajo podatke. Ko se poslana datoteka naloži, se nastavi lastnost window.name in sproži dogodek onLoad, ki v iframe naloži prazno lokalno datoteko. Zatem lahko preberemo window.name, v katerem je shranjen odgovor.

4 Postopek nadgrajevanja

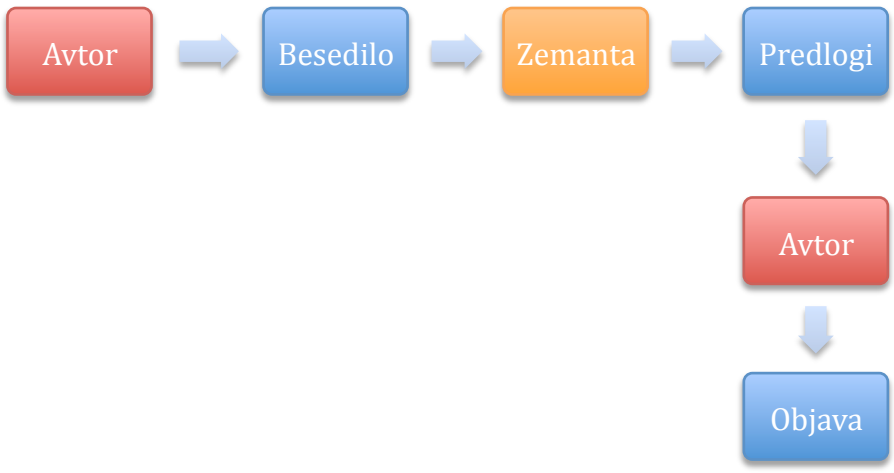
Pregledali smo najpomembnejša programska orodja in tehnologije za nadgrajevanje spletnih strani in se seznanili s ključnimi pastmi, na katere mora biti pozoren razvijalec pri nadgrajevanju s CSS in z JavaScriptom. S tem smo osvojili dovolj znanja, da se lahko lotimo nadgrajevanja.

V tem poglavju si bomo zato sistematično ogledali potek izdelave nadgradnje spletnih strani. Kot primer bomo uporabili spletno aplikacijo Zemanta istoimenskega podjetja in jo vgradili v različne spletne strani s pomočjo bookmarkleta, uporabniške skripte za Greasemonkey, razširitve za Firefox in Jetpacka. Ker Zemanta ponuja tudi programski vmesnik za uporabo storitve v neodvisnih rešitvah, jo bomo povezali tudi s storitvijo Ubiquity.

Obenem bomo podrobno prikazali tudi način, kako smo rešili težave s CSS, časovno neusklajenostjo in sporazumevanjem iz različnih izvorov. Tako bomo Zemanto uporabili kot šolski primer za študij nadgrajevanja, ki je kljub kompleksnosti aplikacije primeren tudi za razvijalce manj zahtevnih nadgradenj.

4.1 Zemanta API

Preden se lotimo samega nadgrajevanja, si oglejmo, kaj sploh Zemanta je in kaj vse lahko z njo počnemo. Razvijalci lahko uporabijo programski vmesnik (API) do Zemante za pridobivanje sorodnih vsebin iz besedila, ki ga pošljejo skozi vmesnik. Odgovor Zemanta API je sestavljen iz štirih glavnih (članki, ključne besede, slike in povezave v besedilu) in ene izbirne komponent (kategorije).



Slika 12: Ko avtor napiše besedilo, s pomočjo Zemante pridobi predloge sorodnih vsebin (članke, ključne besede, slike in povezave v besedilu), s katerimi ga pred objavo obogati.

Pred uporabo programskega vmesnika se je potrebno registrirati na spletnih straneh Zemante za razvijalce [12] in pridobiti ključ za API. Ta je namreč potreben pri vsaki uporabi storitve Zemanta. Za brezplačno uporabo storitve moramo sprejeti omejitev največ tisoč klicev na Zemanta API dnevno, po dogovoru pa nam je podjetje omejitev pripravljeno povečati na deset tisoč.

Besedilo, ki ga pošljemo, je lahko navadno besedilo ali HTML, vendar lahko slednji vsebuje le "čisto vsebino". To pomeni, da so sicer dovoljene značke za povezave ali krepko besedilo, ne smemo pa poslati npr. celotne spletne strani skupaj z navigacijo, glavo, nogo itd. S tem bi namreč poslabšali kakovost predlogov, ki jih v odgovoru dobimo od Zemante.

Pri klicu na Zemanta API moramo obvezno navesti tudi eno izmed treh podprtih metod:

- **zemanta.suggest**: vrača članke, ključne besede, slike in povezave v besedilu,
- **zemanta.suggest_markup**: vrača samo povezave v besedilu,
- **zemanta.preferences**: vrača nastavitve za posameznega uporabnika.

Določiti moramo tudi obliko, v kateri želimo dobiti odgovor. Izbiramo lahko med štirimi vrednostmi:

- **xml**: najpogostejša oblika za izmenjavo podatkov na internetu,
- **json**: JSON je podoben XML, vendar lažji za razčlenjevanje v JavaScriptu,
- **wnjson**: podobno kot JSON, vendar uporablja prenos preko window.name,
- **rdfxml**: semantična aplikacija ponuja tudi odgovor v obliki strukture RDF/XML.

Parameter	Opis
api_key	Vaš ključ za API
text	Vhodno besedilo (navadno ali HTML)
method	Metoda na strežniku
format	Zahtevana oblika odgovora

Tabela 2: obvezni parametri, ki jih moramo z vsakim zahtevkom poslati na Zemanta API

Končna točka klica Zemanta API je http://api.zemanta.com/services/rest/0.0/, zaradi velikosti besedila in omejitev zahtevka GET pa je treba vedno uporabiti POST. S tem imamo dovolj podatkov, da pošljemo zahtevek na Zemanta API. Spodnji primer prikazuje klic v programskem jeziku Python.

```
import urllib

gateway = 'http://api.zemanta.com/services/rest/0.0/'
args = {'method': 'zemanta.suggest',
        'api_key': 'key1234',
        'text': '''The Phoenix Mars Lander has successfully deployed its
robotic arm and tested other instruments including a laser designed to
detect dust, clouds, and fog. The arm will be used to dig up samples of the
Martian surface which will be analyzed as a possible habitat for life.''',
        'format': 'xml'}

args_enc = urllib.urlencode(args)
print urllib.urlopen(gateway, args_enc).read()
```

Zemanta API na klic odgovori v obliki, ki smo jo določili s parametrom format. Strukturo odgovora prikazuje spodnja tabela.

Parameter	Opis
status	Označuje stanje zahtevka
rid	Enolična oznaka zahtevka
articles	Seznam objektov s sorodnimi članki
keywords	Seznam objektov s ključnimi besedami
images	Seznam objektov s slikami
markup	Objekt s povezavami v besedilu
categories	Seznam objektov s kategorijami
signature	Podpis (koda HTML)

Tabela 3: obvezni parametri, ki jih moramo z vsakim zahtevkom poslati na Zemanta API

Več podrobnosti o Zemanti API je na voljo na Zemantinih straneh za razvijalce [13].

4.2 Spletna aplikacija Zemanta

Semantična tehnologija, ki prepozna bistvo besedila in poišče sorodne vsebine, je uporabna na različnih področjih. Zato ne preseneča, da se je Zemantin API v manj kot dveh letih obstoja precej uspešno razširil in ima čez 1000 uporabnikov. Podjetje Zemanta se je odločilo, da jo ponudi blogerjem, zato razvija spletno aplikacijo, ki se integrira v platforme za pisanje blogov oz. sisteme za upravljanje vsebine (CMS), namenjene bloganju.

Zemanta se pri pisanju bloga pojavi kot poseben gradnik (angl. widget) ob urejevalniku besedila. Med pisanjem analizira besedilo in samodejno prikazuje predloge sorodnih vsebin: slike, članke, ključne besede in povezave v besedilu. Te lahko s klikom vključimo v besedilo, ki ga na ta način obogatimo in naredimo privlačnejšega za bralce.

Zemanta je na voljo kot razširitev za brskalnika Firefox in Internet Explorer, v Safariju, Chromu in Operi pa jo lahko uporabimo s pomočjo bookmarkleta. Poskusno je na voljo tudi Jetpack, razvijalci pa so nekoč internetno uporabljali tudi uporabniške skripte za Greasemonkey. Že iz omenjenih oblik, v katerih je na voljo aplikacija, je povsem jasna izbira JavaScripta kot osrednjega programskega jezika, za postavitev in izgled pa skrbi CSS.

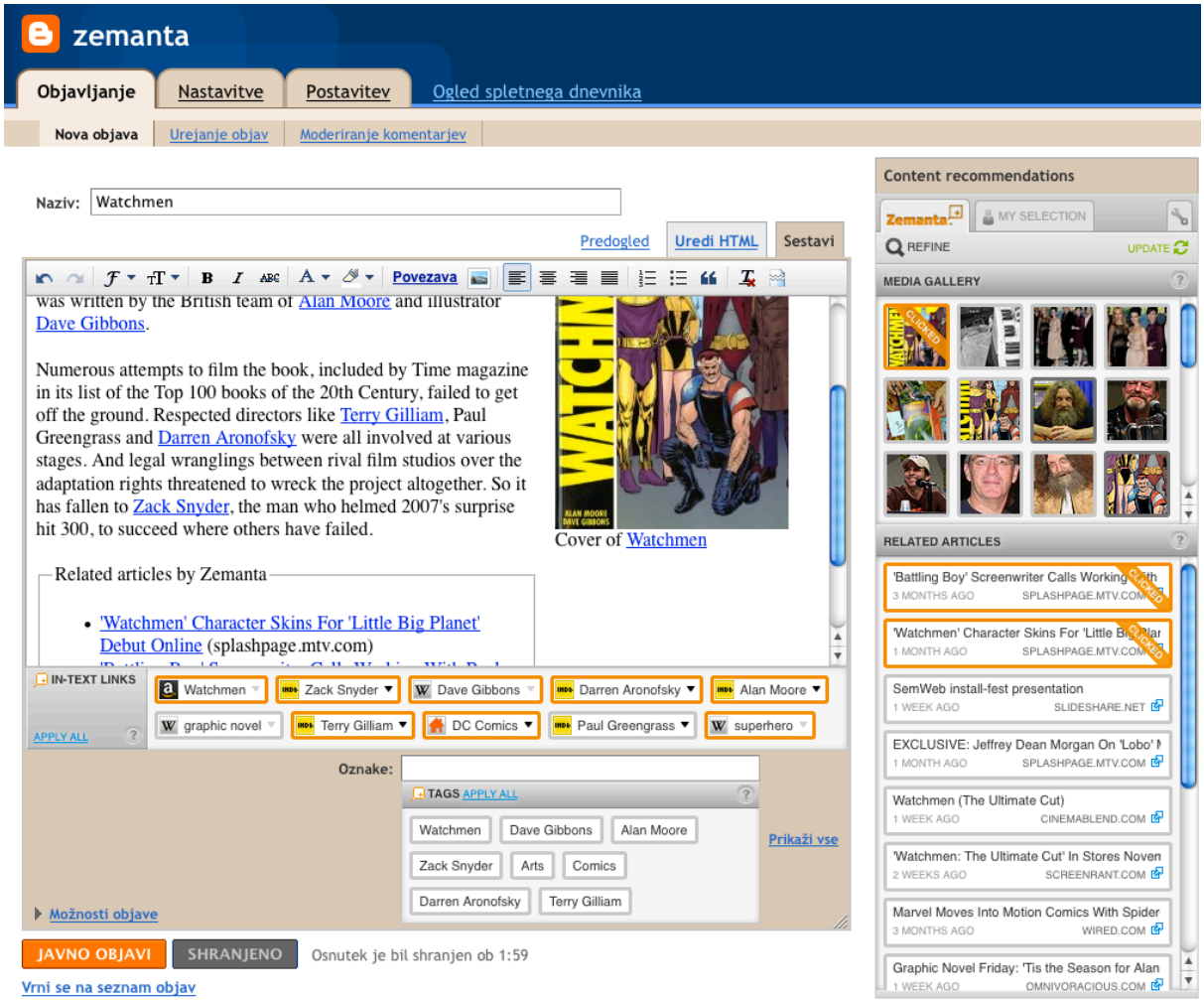
Tako programska koda v JavaScriptu kot tudi prekrivni slogi se nahajajo na Zemantinih strežnikih. Torej je vse, kar mora storiti razširitev (ali katera druga oblika za namestitev Zemante), da v ustrezno spletno strani vstavi vse potrebne datoteke, te pa potem same poskrbijo za delovanje in postavitev aplikacije.

Seznam datotek, ki so potrebne za pravilno delovanje in postavitev spletne aplikacije Zemanta, je prikazan v tabeli 4. Štiri izmed njih so stalne, dve pa sta drugačni za vsako podprto platformo za pisanje blogov.

Datoteka	Opis
loader.js	Naloži vse ostale datoteke
jquery.js	Programsko ogrodje jQuery
jquery.zemanta.js	Vse temeljne funkcije
<platform>.js	Prilagoditve funkcij za platformo
zemanta-widget.css	Osnovni izgled in postavitev
zemanta-widget-<platform>.css	Prilagoditve izgleda in postavitve platformi

Tabela 4: Datoteke, ki so potrebne za pravilno delovanje in postavitev spletne aplikacije Zemanta.

Kot je razvidno iz tabele, mora razširitev naložiti samo datoteko loader.js, ki potem naloži vse ostale in še pred tem vstavi vse potrebne elemente HTML, na katere se datoteke zanašajo. Spodnja slika prikazuje, kako izgleda integracija Zemante v platformo Blogger.



Slika 13: Pisanje bloga v platformi Blogger z nameščeno Zemanto. Pod besedilom je seznam povezav, ki jih lahko vstavimo v besedilo, še nižje so ključne besede, ki jih s klikom dodamo v polje Oznake. Na desni strani je stranska vrstica s slikami, članki in ostalimi zmogljivostmi aplikacije.

A preden se izvede loader.js, moramo preveriti, ali smo na pravi spletni strani. Večina nadgradenj namreč ni vezanih na vse spletne strani, ampak zgolj na nekatere. Zato je že Greasemonkey kot eden izmed pionirjev nadgrajevanja v uporabniških skriptah dovoljeval uporabo parametra @include, s katerim določimo naslove spletnih strani, na katerih se bo skripta izvajala. Bolj prefinjeno rešitev predstavlja spodnji primer iz Zemante:

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
	<release_id>1251366222</release_id>

	<rule>
		<name>Drupal</name>
		<regexp>https?://.*/index\.php\?q=node/add/.+</regexp>
		<regexp>https?://.*/\?q=node/add/.+</regexp>
		<regexp>https?://.*/node/add/.+</regexp>
		<regexp>https?://.*/index\.php\?q=node/[0-9]+/edit.*</regexp>
		<regexp>https?://.*/\?q=node/[0-9]+/edit.*</regexp>
		<regexp>https?://.*/node/[0-9]+/edit.*</regexp>
		<url>http://static.zemanta.com/widgets/drupal/loader.js</url>
	</rule>

	<rule>
		<name>DotClear 2.0</name>
		<regexp>https?://.*/post\.php</regexp>
		<url>http://static.zemanta.com/widgets/dotclear/loader.js</url>
	</rule>

	// More rules…

</rules>
```

Datoteka XML (recimo ji rules.xml) z regularnimi izrazi določa naslove spletnih strani, za katere naj se uporabi katera izmed datotek loader.js. Kot bomo videli kasneje, za vsako platformo potrebujemo svoj loader.js, strani, ki ne ustrezajo nobenemu izmed pravil, pa so nepodprte in na njih nadgradnja ne bo imela nobenega vpliva.

Takšna rešitev je primerna za uporabo tako v razširitvah za brskalnike kot tudi z ostalimi tehnologijami. Celo bookmarklet, ki ga lahko uporabnik poskuša uporabiti na poljubni spletni strani, lahko najprej na strežniku preveri, ali spletni naslov ustreza kateremu izmed pravil in ga ustrezno obravnava.

Ko se prepričamo, da smo na ustrezni strani, vstavimo loader.js. Ker je datoteka z malenkostnimi spremembami uporabna za različne vrste nadgradenj, ne le za Zemanto, jo spodaj navajamo v celoti. Kako deluje? Najprej poišče element (spremenljivka insertionSpace), ob katerega vstavi začetne elemente HTML (spremenljivka widget). Ko to stori, naloži vse tri skripte in oba prekrivna sloga (funkcija load_file). Iz kode je razvidno, da za vsako platformo potrebujemo svoj loader.js.

Ker se loader.js vstavi med nalaganjem spletne strani, bi se lahko zgodilo, da element insertionSpace med preverjanjem še ne bi obstajal, kasneje pa. Zato loader.js vsebuje varovalko, ki pet sekund preverja, ali je element na voljo, in šele nato vstavi kodo HTML in datoteke (setTimeout). S podobnimi težavami se srečamo pogosto in spadajo v sklop časovne neusklajenosti (glej poglavje 3.4).

```
 (function () {
```

```
function now() {
	return new Date().getTime();
}
function load_file(file) {
	var i = 0, j = 0, obj = null, head = null;
	if (file.constructor === Array) { // array
		for (i = 0, j = file.length; i < j; i += 1) {
			arguments.callee.call(this, file[i]);
		}
		return;
	}
	head = arguments.callee.head = arguments.callee.head ||
		 document.getElementsByTagName("head")[0];
	if (file.indexOf('.js') >= 0) { // JavaScript
		obj = document.createElement('script');
		obj.setAttribute("type", "text/javascript");
		obj.setAttribute("src", file);
	} else if (file.indexOf('.css') >= 0) { // CSS
		obj = document.createElement("link");
		obj.setAttribute("rel", "stylesheet");
		obj.setAttribute("type", "text/css");
		obj.setAttribute("href", file);
	}
	if (head && obj) {
		head.appendChild(obj);
	}
}
var staticDomain = 'http://static.zemanta.com/',
	widget = document.createElement('div'),
	insertionSpace = null,
	t0 = now();
widget.setAttribute('id', 'zemanta-sidebar');
widget.innerHTML = '<div id="zemanta-control" class="zemanta"></div>
	<div id="zemanta-message" class="zemanta">Loading Zemanta...</div>
	<div id="zemanta-filter" class="zemanta"></div><div id="zemanta-
	gallery" class="zemanta"></div><div id="zemanta-articles"
	class="zemanta"></div><div id="zemanta-preferences"
	class="zemanta"></div>';
(function () {
	insertionSpace = document.getElementById('some-element');
	if (insertionSpace) {
		insertionSpace.parentNode.insertBefore(widget, insertionSpace);
		load_file([
			staticDomain + 'core/zemanta-widget.css',
			staticDomain + 'platform/zemanta-widget-platform.css',
			staticDomain + 'core/jquery.js',
			staticDomain + 'core/jquery.zemanta.js',
			staticDomain + 'platform/platform.js'
		]);
	} else if (now() - t0 < 5000) {
		setTimeout(arguments.callee, 100);
	}
})();

})();
```

V naslednjem koraku si oglejmo še primer od platforme odvisne kode, ki glavni Zemantini knjižnici jquery.zemanta.js sporoči vse potrebne podatke o uporabljeni platformi. Zgled uporabe od platforme odvisne kode je primeren za uporabo tudi v drugih nadgradnjah, zato strukturo datoteke <platform>.js navajamo v celoti.

```
(function () {
	var $ = null;
	function setPlatform($, p) {
		return $.zextend(p, {
			widget_version: 3,
			platform: {
				// Platform specific functions
			}
		});
	}
	function waitForLoad() {
		var done = false, t0 = null;
		if (typeof $.zemanta === "undefined") {
			$('#zemanta-message').html('Waiting...');
			setTimeout(arguments.callee, 100);
			return;
		}
		t0 = arguments.callee.t0 = arguments.callee.t0 ||
		new Date().getTime();
		$('#zemanta-message').html('Initializing...');
		try {
			done = $.zemanta.initialize(setPlatform($, {
				interface_type: "YOURPLATFORMNAME",
				tags_target_id: "tags-input"
			}));
		} catch (er) {
			done = true;
		}
		if (!done) {
			if (new Date().getTime() - t0 < 15000) {
				setTimeout(arguments.callee, 100);
			} else {
				$('#zemanta-message').html('Editor not found.');
			}
		}
	}
	try {
		$ = window.jQuery;
		if (!$) {
			throw "No jQuery!";
		}
		if ($('#zemanta-message').html() === 'Loading...') {
			$('#zemanta-message').html('Preparing...');
		}
		waitForLoad();
	} catch (er) {
		window.setTimeout(arguments.callee, 100);
		return;
	}
})();
```

Ker se večji del kode zanaša na programsko ogrodje jQuery, najprej preverimo, ali je ta na voljo, sicer počakamo (setTimeout). V nadaljevanju sprožimo začetek izvajanja kode v osrednji knjižnici jquery.zemanta.js s klicem $.zemanta.initialize, še pred tem pa preverimo, ali se je datoteka v celoti naložila (obsega več tisoč vrstic kode). Kot parameter klica podamo funkcijo setPlatform, ki določa vse od platforme odvisne funkcije, njena parametra pa sta ime platforme in ID elementa, v katerega vpisujemo ključne besede.

Oglejmo si nekaj primerov od platforme odvisnih funkcij. Datoteka jquery.zemanta.js vsebuje prototipe teh funkcij, ki jih lahko v datoteki <platform>.js povozimo. Tipični funkciji, ki ju moramo prepisati skoraj za vsako platformo, sta links_initialize() in tags_initialize(), ki poiščeta elementa ($(izbirnik CSS)), kamor naj se vstavijo predlogi povezav in ključnih besed.

Spodaj je primer funkcije links_initialize():

```
links_initialize: function () {
      $('links-element-selector').append('<h2 class="vertical"><span
      class="zemanta-links-title">In-text Links</span><a class="zemanta-
      help" href="http://zemanta.com/faq/#links">?</a></h2><ul id="zemanta-
      links-div-ul"><li class="zemanta-title"><span>Link recommendations
      will appear here</span></li></ul><p class="zem-clear"> </p>');
}
```

Funkcija tags_initialize() je zelo podobna:

```
tags_initialize: function () {
      $('tags-element-selector').append('<h2 class="vertical"><span
      class="zemanta-tags-title">Tags</span><a class="zemanta-help"
      href="http://zemanta.com/faq/#links">?</a></h2><ul id="zemanta-tags-
      div-ul"><li class="zemanta-title"><span>Tag recommendations will
      appear here</span></li></ul><p class="zem-clear"> </p>');
}
```

Poleg funkcij lahko za vsako platformo hranimo tudi spremenljivke. Spodnji primer določa, ali urejevalnik podpira metodo povleci in spusti za vstavljanje slik:

```
dnd_supported: true
```

Za uporabo Zemante in mnogih drugih nadgradenj je potrebno besedilo, ki ga pisec bloga piše v urejevalnik. Zato potrebujemo funkcijo get_editor, ki vrača primerek urejevalnika za vsako platformo. Če uporabljamo urejevalnik obogatenega besedila, kot sta npr. tinyMCE in FCKeditor, zadošča navesti že ID elementa iframe, v katerem se nahaja urejevalnik.

Če pa ID ne obstaja, ali če imamo opravka z bolj zapletenim scenarijem, ki podpira tudi izbiro med urejevalnikom obogatenega besedila in urejevalnika HTML, potrebujemo funkcijo, ki vrača ustrezno referenco, ki je objekt z naslednjimi lastnostmi:

- **element**: element DOM, v katerem se nahaja besedilo
- **property**: lastnost elementa DOM, ki vrača besedilo kot niz
- **type**: 'RTE' za urejevalnik obogatenega besedila, sicer ime značke (male črke)
- **win**: objekt window, v katerem se nahaja element (samo za type: ‘RTE’)

Primer vrednosti, ki jih mora vrniti funkcija, če uporabljamo urejevalnik obogatenega besedila (predstavitev v obliki JSON):

```
win = document.getElementById('editor-iframe').contentWindow;
return {
      element: win.document.body,
      property: 'innerHTML',
      type: 'RTE',
      win: win
};
```

Primer vrednosti, ki jih mora vrniti funkcija, če uporabljamo urejevalnik HTML (predstavitev v obliki JSON):

```
elm = document.getElementById('editor-textarea');
return {
      element: elm,
      property: 'value',
      type: elm.tagName.toLowerCase(),
      win: null
};
```

Primer celotne funkcije get_editor():

```
get_editor: function () {
      var elm = null, win = null, editor = {element: null, property: null,
                type: null, win: null};
      try {
            elm = $('#editor-iframe').get(0);
            if (elm && elm.contentWindow) {
                  win = elm.contentWindow;
                  elm = null;
            } else {
                  elm = $('#editor-textarea').get(0);
            }
            editor = win && {element: win.document.body, property:
                     'innerHTML', type: 'RTE', win: win} ||
                     elm && {element: elm, property: 'value', type:
                     elm.tagName.toLowerCase(), win: null} ||
                     editor;
      } catch (er) {
            $.zemanta.log(er);
      }
      return editor;
}
```

Ko pripravimo še glavno programsko knjižnico (v primeru Zemante je to jquery.zemanta.js) in sestavimo datoteke CSS, lahko začnemo nadgrajevanje strani s pomočjo tehnologij, ki smo jih spoznali v 2. poglavju.

4.3 Bookmarklet

Kot smo že zapisali, je naloga bookmarkleta, da v ustrezno spletno strani vstavi vse potrebne datoteke. Kasneje smo ugotovili, moramo vstaviti samo datoteko loader.js, ki potem naloži vse ostale. Za nalaganje datoteke lahko uporabimo funkcijo load_file() iz loader.js, vendar jo lahko poenostavimo, saj ni potrebno, da podpira tudi vstavljanje datotek CSS.

```
load_file = function (s) {
      var o = document.createElement('script');
      o.type = 'text/javascript';
      o.src = s;
      document.getElementsByTagName("head")[0].appendChild(o);
};
```

Pred vstavljanjem skripte je še potrebno preveriti, ali naša nadgradnja sploh podpira trenutno spletno stran. Zato Zemanta uporablja posebno storitev na svojih strežnikih, ki za vsak spletni naslov (URL) preveri, ali ustreza kateremu izmed pravil v datoteki rules.xml (glej poglavje 4.2).

```
http://labs.zemanta.com/bookmarklet/?url=' +
encodeURIComponent(window.location.href));
```

V primeru ujemanja storitev vrne klic funkcije load_file(), v parametru pa je nastavljena pot do ustrezne datoteke loader.js, ki ustreza pravilu v datoteki rules.xml, pri katerem je prišlo do zadetka.

```
load_file('http://fstatic.zemanta.com/widgets/blogger.com/loader.js');
```

Če spletni naslov ne ustreza nobenemu izmed pravil, že zunanja storitev izpiše obvestilo, ki ga uporabnik prebere v pogovornem oknu.

```
alert("Please use bookmarklet on Compose screen in your blog or email");
```

Ker nekateri starejši brskalnik dopuščajo le nekaj 100 znakov dolgo programsko kodo bookmarkletov (Tabela 5), moramo včasih najpogostejše besede skrajšati .

```
var   w = window,
      d = document,
      z = 'Zemanta',
      e = 'Element',
      a = 'Load',
```

In jih tako tudi uporabiti v programski kodi:

```
h = d['get' + e + 'sByTagName']('head')[0]
```

Kljub temu, da je kratka, je zato programska koda bookmarkleta lahko nekoliko težje berljiva:

```
(function () {
      var w = window,
            d = document,
            z = 'Zemanta',
            e = 'Element',
            a = 'Load',
            h = d['get' + e + 'sByTagName']('head')[0],
            l = function (s) {
                  var o = d['create' + e]('script');
                  o.type = 'text/javascript';
                  o.src = s;
                  h.appendChild(o);
            };
      w[z + a + 'Script'] = l;
      w[z + 'GetAPIKey'] = function () {
            return '4j45amqay26vcjxyrrsf2jz2';
      };
      if (!w[z + a + 'ed']) {
            w[z + a + 'ed'] = 1;
            l('http://labs.zemanta.com/bookmarklet/?url=' +
            encodeURIComponent(w.location.href));
      }
})();
```

Funkcijo spremenimo v bookmarklet tako, da ji spredaj dodamo niz "javascript:" in odstranimo vse presledke.

```
/* JSMIN
javascript:(function(){var
w=window,d=document,z='Zemanta',e='Element',a='Load',h=d['get'+e+'sByTagNam
e']('head')[0],l=function(s){var
o=d['create'+e]('script');o.type='text/javascript';o.src=s;h.appendChild(o)
;};w[z+a+'Script']=l;w[z+'GetAPIKey']=function(){return'4j45amqay26vcjxyrrs
f2jz2';};if(!w[z+a+'ed']){w[z+a+'ed']=1;l('http://labs.zemanta.com/bookmark
let/?url='+encodeURIComponent(w.location.href));}})();
*/
```

Brskalnik	Največje število znakov
Internet Explorer 6	508
Internet Explorer 6 SP2	508 (presledki niso všteti)
Internet Explorer 7	2084
Internet Explorer 8	> 2000
Firefox	> 2000
Safari	> 2000
Chrome	> 2000
Opea	> 2000

Tabela 5: Datoteke, ki so potrebne za pravilno delovanje in postavitev spletne aplikacije Zemanta.

4.4 Greasemonkey

Glavna pomanjkljivost bookmarkletov je, da ne omogočajo samodejnega zaganjanja programske kode z nadgradnjo, ko se naloži spletna stran, ampak jih je potrebno vsakič ročno vključiti. Težavo odpravlja Greasemonkey, pri katerem za vsako uporabniško skripto določimo, na katerih spletnih straneh naj se samodejno izvede ob vsakem nalaganju.

Vsaka uporabniška skripta je sestavljena iz dveh delov. Prvemu pravimo glava in določa nekatere osnovne podatke o uporabniški skripti, kot so ime, opis, avtor in nekateri drugi podatki. Najpomembnejši del glave pa je, da s parametrom @include določimo naslove spletnih strani, na katerih se bo skripta izvajala. Naštejemo jih lahko v več vrsticah, uporabimo pa lahko tudi regularne izraze, s katerimi določimo večjo množico spletnih strani.

```
// ==UserScript==
// @name           Zemanta Wordpress
// @namespace      Zemanta
// @description    Zemanta Wordpress plugin
// @version        1.0
// @date           2009-05-10
// @creator        Zemanta Team
// @include        http://*.wordpress.com/wp-admin/post-new.php
// @include        http://*.wordpress.com/wp-admin/post.php?action=edit*
// ==/UserScript==
```

Drugi del skripte vsebuje programsko koda v JavaScriptu, ki se izvede na vnaprej določenih straneh. Ta del je še preprostejši kot pri bookmarkletih, saj moramo zgolj vstaviti datoteko loader.js. Pomagamo si lahko s funkcijo load_file(), medtem ko logike z preverjanje ustreznosti spletnih naslovov z datoteko rules.xml ne potrebujemo, saj to lahko opravlja glava.

```
var o = document.createElement('script');
o.type = 'text/javascript';
o.src = 'http://static.zemanta.com/widgets/wordpress.com/loader.js';
document.getElementsByTagName("head")[0].appendChild(o);
```

Greasemonkey omogoča tudi svojevrstno rešitev problema sporazumevanja iz različnih domen:

```
// Evil hack for getting cross-site XMLHttpRequest to our server

unsafeWindow.GMlog = GM_log;
unsafeWindow.GM_xmlhttprequest = GM_xmlhttpRequest;
```

Ker je bila prikazana rešitev podprta le v starejših različicah Greaemonkeyja (pred 0.7.20080121.0) [14], je primerneje, če uporabimo metodo prenosa podatkov preko window.name.

Oglejmo si zdaj, kako izgleda celotna uporabniška skripta za uporabo Zemante v Wordpressu:

```
// ==UserScript==
// @name           Zemanta Wordpress
// @namespace      Zemanta
// @description    Zemanta Wordpress plugin
// @version        1.0
// @date           2009-05-10
// @creator        Zemanta Team
// @include        http://*.wordpress.com/wp-admin/post-new.php
// @include          http://*.wordpress.com/wp-admin/post.php?action=edit*
// ==/UserScript==


// Evil hack for getting cross-site XMLHttpRequest to our server

unsafeWindow.GMlog = GM_log;
unsafeWindow.GM_xmlhttprequest = GM_xmlhttpRequest;

var o = document.createElement('script');
o.type = 'text/javascript';
o.src = 'http://static.zemanta.com/widgets/wordpress.com/loader.js';
document.getElementsByTagName("head")[0].appendChild(o);
```

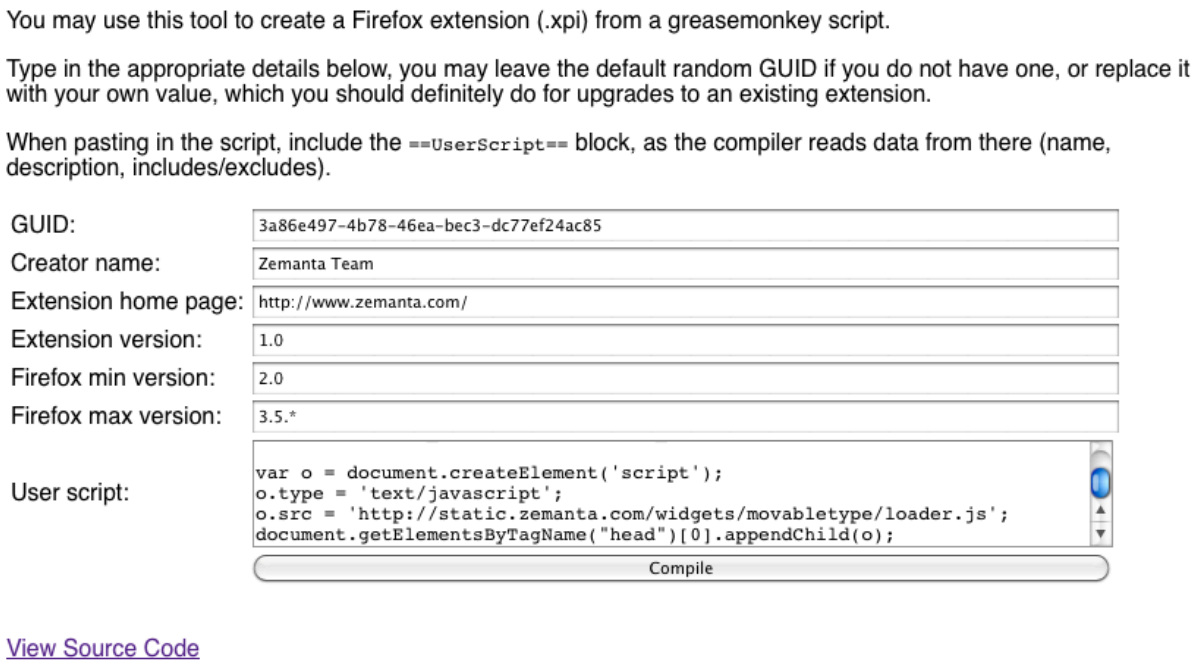
Zelo podobno uporabniško skripto lahko uporabimo tudi za vse ostale platforme. Spodnji primer je npr. za Movable Type in se minimalno razlikuje od različice za Wordpress.

```
// ==UserScript==
// @name           Zemanta Movable Type
// @namespace      Zemanta
// @description    Zemanta Movable Type development plugin
// @version        1.0
// @date           2009-06-04
// @creator        Zemanta Team
// @include        http://*mt.cgi?__mode=view&_type=entry*
// ==/UserScript==


// Evil hack for getting cross-site XMLHttpRequest to our server

unsafeWindow.GMlog = GM_log;
unsafeWindow.GM_xmlhttprequest = GM_xmlhttpRequest;

var o = document.createElement('script');
o.type = 'text/javascript';
o.src = 'http://static.zemanta.com/widgets/movabletype/loader.js';
document.getElementsByTagName("head")[0].appendChild(o);
```

4.5 Razširitev za brskalnik

Število uporabnikov, ki imajo možnost izvajanja uporabniških skript, je razmeroma majhno v primerjavi s celotnim številom uporabnikov interneta. Če želimo svojo nadgradnjo spletnih strani ponuditi čim večjemu krogu uporabnikov, potrebujemo drugačno rešitev. To nam omogočajo razširitve za brskalnike, ki jih uporabniki namestijo v enem koraku.

Če razvijemo uporabniško skripto za Greasemonkey, je prehod na razširitev za Firefox zelo preprosta. Na spletu je brezplačno na voljo storitev [15], ki uporabniško skripto pretvori v obliko razširitve za Firefox (XPI), ki jo preprosto prenesemo na računalnik in jo lahko začnemo širiti naprej.



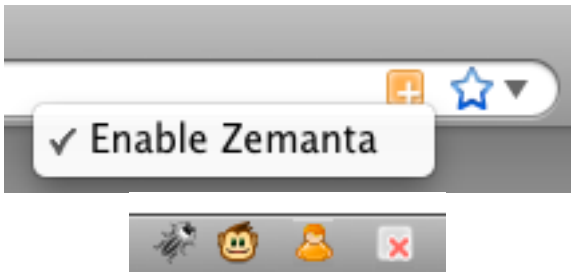
Slika 14: Potem ko izpolnimo nekaj podatkov o razširitvi, prilepimo izvorno kodo uporabniške skripte in jo prevedemo v razširitev.

Podroben (ročen) razvoj razširitve za Firefox presega obseg tega diplomskega dela, še precej več dela pa bi imeli z razvojem razširitve za Internet Explorer. Kot smo omenili že v poglavju 2.4, imajo namreč razširitve za brskalnik prednosti v primerjavi z Greasemonkeyjem in uporabniškimi skriptami predvsem za končne uporabnike, pomanjkljivosti pa najbolj občutijo razvijalci.

Zato si bomo v naslednjem poglavju ogledali izdelavo razširitve s tehnologijo Jetpack, ki je precej podobna tudi novi vrsti razširitev, ki jo Google pripravlja za svoj brskalnik Chrome. Ker gre za nov tip razširitev, ki bodo nadomestile klasične, je njihovo poznavanje zelo pomembno.

4.6 Jetpack

Prednost Jetpacka v primerjavi s klasičnimi razširitvami za Firefox lahko zelo nazorno prikažemo z naslednjim primerom: Zemantina razširitev za Firefox je arhiv trinajstih datotek, ki skupaj obsegajo 180 kB, medtem ko enako zmogljivi jetpack obsega manj kot sto vrsti programske kode.



Slika 15: Zgoraj – razširitev za Firefox v vrstico z naslovom vstavi ikono Zemante, s katero lahko vključimo ali izključimo nadgradnjo. Spodaj – tehnologija Jetpack zaenkrat še ne omogoča dostopa do vrstice z naslovom, zato ikono za vklop in izklop Zemante vstavimo v vrstico stanja.

Sprehodimo se torej skozi Zemantin jetpack.js. Najprej definiramo spremenljivke in v vrstico stanja vstavimo ikono, ki ji določimo tudi ustrezne mere.

```
var domain = 'http://static.zemanta.com/';

jetpack.statusBar.append({
  html: '<img style="padding-top:2px;">',
  width: 14,
  onReady: function(widget){
```

Ali je Zemanta trenutno vključena ali izključena, shranimo v trajno shrambo. Ker ga jetpack privzeto ne podpira, ga moramo najprej uvoziti v imenski prostor.

```
    jetpack.future.import('storage.simple');
    // Store Zemanta status to persistent storage
    if (!jetpack.storage.simple.zemantaStatus) {
      jetpack.storage.simple.zemantaStatus = 'enabled';
    }
```

Glede na stanje nastavimo tudi ikono v vrstici stanja. Da je Zemanta izključena, namreč ponazorimo z rdečim križcem čez ikono.

```
    // Set appropriate Zemanta icon
    if (jetpack.storage.simple.zemantaStatus === 'enabled') {
      $(widget).find('img').attr('src', domain + 'zemicon_normal.png');
    } else {
      $(widget).find('img').attr('src', domain + 'zemicon_disabled.png');
    }
```

Podobno kot stanje v trajno shrambo shranimo tudi ključ za Zemanta API.

```
// Store API key to persistent storage
jetpack.future.import('storage.simple');
if (!jetpack.storage.simple.zemantaAPIKey) {
  jetpack.storage.simple.zemantaAPIKey = '4j45amqay26vcjxyrrsf2jz2';
}
```

Datoteko rules.xml shranimo v shrambo seje, saj se lahko datoteka na strežniku spreminja. Uporabili bi lahko tudi storitev, ki smo jo predstavili v poglavju 4.3.

```
// Store rules.xml to live storage
if (!jetpack.storage.live.zemantaRules) {
  $.ajax({
    url: domain + 'firefox/rules.xml',
    success: function(data, textStatus) {
      jetpack.storage.live.zemantaRules = data.childNodes[0];
    }
  });
}
```

Za vsako pravilo v datoteki rules.xml preverimo, ali morda ustreza trenutnemu spletnemu naslovu.

```
// Check if tab URL matches rules
function matchesRules(location) {
  var url = false;
  $('regexp', jetpack.storage.live.zemantaRules).each(function() {
    var regexp = new RegExp($(this).text());
    if (location.match(regexp)) {
      url = $(this).nextAll('url').eq(0).text();
    }
  });
  return url;
}
```

V primeru ujemanja pravila z naslovom spletne strani vstavimo ustrezno datoteko loader.js in obarvamo Zemantino ikono kot aktivno.

```
// Insert Zemanta scripts
jetpack.tabs.onReady(function (doc) {
  if (jetpack.storage.simple.zemantaStatus === 'disabled') {
    return;
  }
  var url = matchesRules(doc.location.href);
  if (url) {
    var zem = {};
    zem.s = doc.createElement('script');
    zem.s.src = url;
    doc.getElementsByTagName('head')[0].appendChild(zem.s);
    if (jetpack.tabs.focused.url === doc.location.href) {
      $(widget).find('img').attr('src', domain + 'zemicon_active.png');
    }
  }
});
```

Ob vsaki zamenjavi zavihka je potrebno preveriti, ali spletni naslov ustreza kateremu izmed pravil v datoteki rules.xml in ustrezno popraviti ikono.

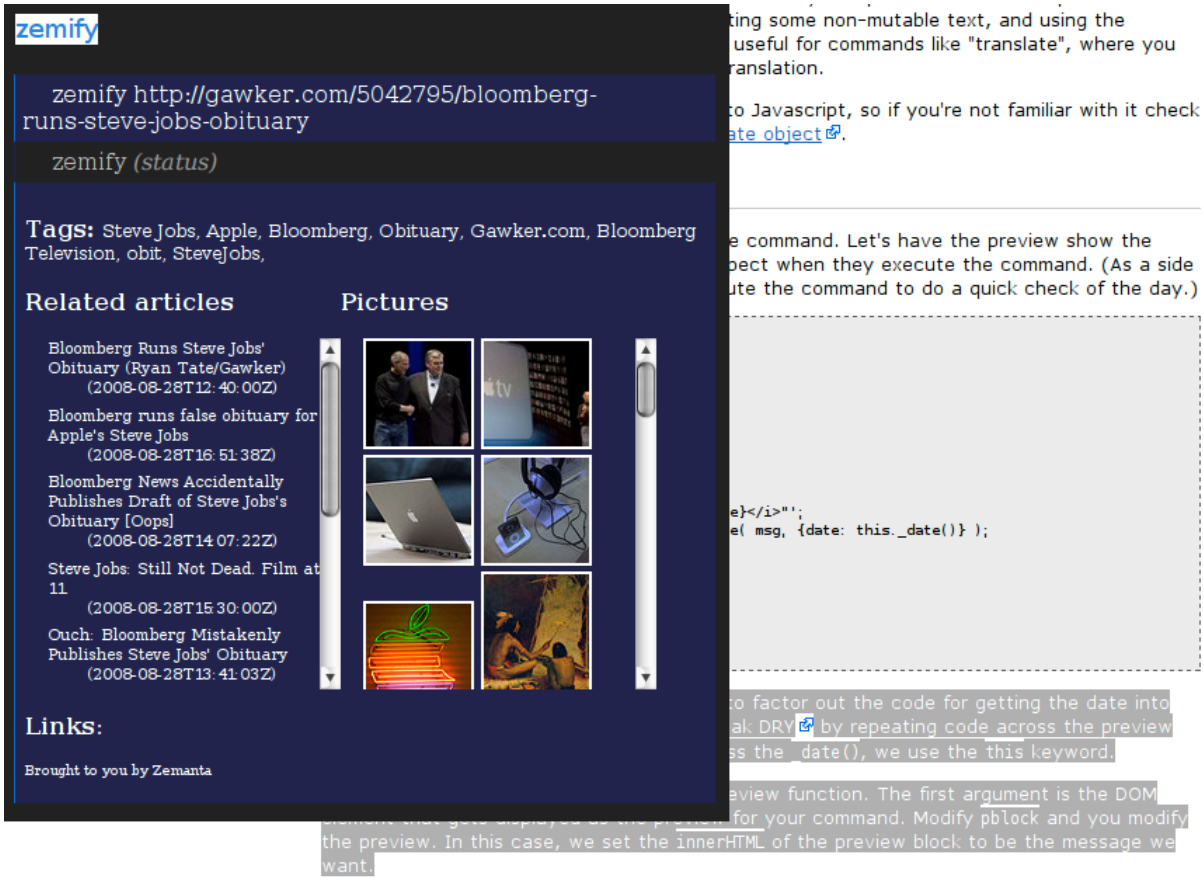
```
    // Set appropriate Zemanta icon for each tab on focus
    jetpack.tabs.onFocus(function() {
      if (jetpack.storage.simple.zemantaStatus === 'disabled') {
        return;
      } else if (matchesRules(jetpack.tabs.focused.url)) {
        $(widget).find('img').attr('src', domain + 'zemicon_active.png');
      } else {
        $(widget).find('img').attr('src', domain + 'zemicon_normal.png');
      }
    });
```

Če uporabnik klikne na ikono Zemante, se Zemanta vključi ali izključi, kar pomeni, da se mora ustrezno spremeniti tudi ikona v vrstici stanja.

```
    // Enable or disable Zemanta on icon click
    $(widget).find('img').click(function() {
      if (jetpack.storage.simple.zemantaStatus === 'enabled') {
        $(this).attr('src', domain + 'zemicon_disabled.png');
        jetpack.storage.simple.zemantaStatus = 'disabled';
      } else if (matchesRules(jetpack.tabs.focused.url)) {
        $(this).attr('src', domain + 'zemicon_active.png');
        jetpack.storage.simple.zemantaStatus = 'enabled';
      } else {
        $(this).attr('src', domain + 'zemicon_normal.png');
        jetpack.storage.simple.zemantaStatus = 'enabled';
      }
    });

  }
});
```

4.7 Ubiquity

Čeprav nima kakšne večje uporabniške vrednosti, je Zemanta na voljo tudi kot ukaz za Ubiquity [16]. Kot zanimivost si ob koncu oglejmo izvorno kodo ukaza.



Slika 16: Zemanto lahko na hitro preizkusimo s pomočjo Ubiquityja. Označimo besedilo, zaženemo Ubiquity in z ukazom zemify prikažemo sorodne članke, slike in ključne besede.

Najprej nastavimo začetne spremenljivke, med drugim tudi vse potrebne parametre za klic Zemanta API.

```
var data = {
      method: 'zemanta.suggest',
      callback: 'none',
      format: 'json',
      text: '',
      api_key: 'g36wbwt2bm835jt43m76snc8',
      zemified: '',
    };
var sendObj = {
      method:"POST",
      url: "http://api.zemanta.com/services/rest/0.0/",
      data: '',
    };
var zemified = "";
```

Izpolnimo nekaj osnovnih podatkov o ukazu, kot so ime, domača stran, avtor itd.

```
CmdUtils.CreateCommand({
  names: ["zemify","zemanta","context"],
  arguments: [ {role: 'object', nountype: noun_arb_text, label:
'selection'}],
  homepage: "http://developer.zemanta.com/",
  author: {name: "Bostjan Spetic", homepage: "http://bostjan.it/"},
  license: "MPL",
```

Definiramo pomožno funkcijo za pretvorbo objekta v niz.

```
  toQueryString: function(params) {
      var qryStr = '';
      for (var key in params) {
        qryStr += key + '=' + encodeURIComponent(params[key]) + '&';
      }
      return qryStr.slice(0,qryStr.length-1);
  },
```

Osrednja funkcija, ki izvede klic na Zemanta API in prikaže rezultat. Predlogo za prikaz rezultata smo zaradi preglednosti kode nadomestili z oznako [PREDLOGA].

```
  preview: function(pblock, args) { //statusText) {
    var statusText = args.object.text;
    var previewTemplate = "Looking for related articles and pictures for
source:<br /><br /> ${status}<br /><br />...";
    var previewTemplate2 = [PREDLOGA];

    var previewHTML = CmdUtils.renderTemplate(previewTemplate, { status:
statusText });
    pblock.innerHTML = previewHTML;

    if(statusText.length < 5) {
      displayMessage("Zemanta requires some text to be selected");
      return;
    }

    data.text = statusText;
    sendObj.data = this.toQueryString(data);

    jQuery.ajax({
                type: sendObj.method, url:sendObj.url, data:sendObj.data,
                success: function(zemdata,statusText){
                   zemified = zemdata;
                   var zdata = eval('(' + zemdata + ')');
                   previewHTML = CmdUtils.renderTemplate(previewTemplate2,
zdata);
                   pblock.innerHTML = previewHTML;
                },
                error: function(data,status){ response = data;
displayMessage("Zemanta error: " + selection); },
                timeout: 30000
              });
  },
});
```

5 Sklepne ugotovitve

V diplomskem delu smo želeli odgovoriti na vprašanje, kako si lahko uporabniki prilagodijo spletno stran svojim željam in kako lahko svoje spletne aplikacije vključijo v obstoječe spletne strani. Obravnavali smo različne probleme, s katerimi se pri tem srečujemo, in različne tehnike za spoprijemanje z njimi. Spoznali smo tudi najbolj razširjene tehnologije in programska orodja za nadgrajevanje spletnih strani ter prikazali njihovo uporabo v praksi.

Ocenjujem, da smo s tem dosegli cilj diplomskega dela. Izdelali smo namreč pripomoček za nadgrajevanje spletnih strani, do katerih nimamo skrbniškega dostopa. Osrednji del dokumentacije predstavlja sistematični pregled izdelave nadgradnje spletne strani na primeru spletne aplikacije Zemanta.

Ker je nadgrajevanje spletnih strani razmeroma nov pojav, je bila največja težava pri nastajanju dela pomanjkljiva dokumentacija. Zato smo si veliko pomagali s tehnično dokumentacijo obstoječih programskih rešitev in tudi s posnetki predavanj razvijalcev teh rešitev.

V splošnem pa je k nadgrajevanju spletnih strani in posledično tudi k nastanku diplomskega dela največ pripomogla skupnost Mozilla. V prvi vrsti z uspehom Firefoxa, ki je zaslužen za razmah odprtih standardov, in z razvojem številnih v delu omenjenih tehnologij, ki te standarde izkoriščajo.

Firefox namreč ni samo brskalnik, ampak je tudi platforma za razvoj razširitev za brskalnik. Ključna programska orodja in tehnologije za nadgrajevanje spletnih strani so tako nastale prav kot razširitve za Firefox – od Greasemonkeyja preko Firebuga do Ubiquityja.

Tudi tehnologija Jetpack, ki predstavlja prihodnost razširitev za Firefox, je zaenkrat na voljo kot razširitev za Firefox. Ker se podobne tehnologije lahkih razširitev, ki sicer temelji na Greasemonkeyju, poslužuje tudi Google v svojem brskalniku Chrome, se odpira vprašanje standardizacije razširitev.

V tem trenutku se sicer zdi utopično, da bi npr. razširitev za Firefox lahko delovala tudi v Chromu, vendar bi s poenotenjem programskih vmesnikov lahko dosegli, da bi bile pri pretvorbi razširitve za en brskalnik v razširitev za drugega potrebne le minimalne spremembe.

Ideje v tej smeri bi prav gotovo zaslužile biti predmet novih raziskav. Te bi lahko v prvi fazi pripeljale vsaj do izgradnje infrastrukture za lahke razširitve tudi v brskalnikih, ki te možnosti zaenkrat ne ponujajo – Safari, Opera in seveda Internet Explorer.

Razmisliti velja tudi o vključitvi programskega ogrodja jQuery v brskalnike. V Firefoxu se bo to zgodilo skupaj z vključitvijo Jetpacka, zelo koristen pa bi bil tudi v ostalih brskalnikih. Nadgrajevanje spletnih strani namreč temelji na JavaScriptu, v katerem bistveno lažje programiramo s pomočjo jQueryjem, obenem pa ta knjižnica zagotavlja tudi uporabo enake kode v različnih brskalnikih.

Tudi za jQuery je zaslužna Mozilla, saj je njegov avtor John Resig, ki je zaposlen v Mozilla Corporation. Enako tudi avtor JavaScripta, Brendan Eich, ki opravlja vlogo tehničnega direktorja.

Seznam slik

Seznam tabel

Literatura

[1] D. Cunliffe, C. Taylor, D. Tudhope, "Query-based navigation in semantically indexed hypermedia", v *Conference on Hypertext and Hypermedia*, New York: ACM, 1997, str. 87-95.

[2] M. Pilgrim, *Greasemonkey Hacks*, Sebastopol: O'Reilly Media, november 2005.

[3] Zemanta JavaScript SDK - Integrating Zemanta. Dostopno na: http://developer.zemanta.com/docs/sdk/

[4] Easy deployment of site-extensions with a browser plugin. Dostopno na: http://video.kiberpipa.org/media/FF3_Marko_Samastur-Browser_plugins/play.html

[5] Wolfram Alpha: number of websites. Dostopno na: http://www.wolframalpha.com/input/?i=number+of+websites

[6] Wolfram Alpha: number of internet users. Dostopno na: http://www.wolframalpha.com/input/?i=number+of+internet+users

[7] Add-ons for Firefox: Firebug. Dostopno na: https://addons.mozilla.org/en-US/firefox/addon/1843

[8] Add-ons for Firefox: Greasemonkey. Dostopno na: https://addons.mozilla.org/en-US/firefox/addon/748

[9] Asa Dotzler: Five years of Firefox. Dostopno na: http://weblogs.mozillazine.org/asa/archives/2009/11/five_years_of_firefo.html

[10] Net Applications: Browser Market Share, November 2009. Dostopno na: http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0

[11] Add-ons for Firefox. Dostopno na: https://addons.mozilla.org/en-US/firefox/

[12] Zemanta Developer Network: Register for an account. Dostopno na: http://developer.zemanta.com/member/register

[13] Zemanta Developer Network: Zemanta Documentation. Dostopno na: http://developer.zemanta.com/docs

[14] GreaseSpot: Version history. Dostopno na: http://wiki.greasespot.net/Version#0.7.20080121.0

[15] User Script Compiler. Dostopno na:
http://arantius.com/misc/greasemonkey/script-compiler

[16] Ubiquity is cool, Zemanta kicks ass! Dostopno na:
http://code.zemanta.com/bostjan/ubiquity/