

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Završni rad br. 1038

**Pozadinsko izvođenje potrošačkih
primjenskih programa udomljenih u web
pregledniku**

Ivan Brkić

Zagreb, veljača 2010.

Zahvaljujem mentoru prof. dr. sc. Siniši Srbliću
na stručnom vodstvu tijekom izrade Završnog rada.

Sadržaj

1	Uvod	4
2	Primjenski programi unutar web preglednika	6
3	Tehnologije za izradu web primjenskih programa	9
3.1	XML	9
3.2	HTML	11
3.3	CSS	12
3.4	JavaScript	14
3.4.1	DOM	15
4	Proširenja preglednika	18
4.1	Preglednik Mozilla Firefox	18
4.2	Mozilla dodaci	18
4.3	Mozilla proširenja	20
4.3.1	XPCOM	20
4.3.2	XUL	22
4.3.3	Programski jezik JavaScript	24
4.3.4	CSS	25
5	Sustav za pozadinsko izvođenje web primjenskih programa	26
5.1	Programska knjižnica <i>Minimize to Tray</i>	26
5.2	Izmjena standardnih funkcionalnosti preglednika	28
5.3	Korisničko sučelje	32
6	Zaključak	34
7	Literatura	35
8	Sažetak	36

1 Uvod

Unazad posljednjih nekoliko godina primjetan je porast udjela primjenskih programa za web preglednike, u odnosu na klasičan oblik primjenskih programa namjenjenih postavljanju na lokalno računalo. Zemljopisna rasprostranjenost, visoka razina pouzdanosti i dostupnosti te porast brzine prijenosa podataka na Internetu samo su neke od brojnih svojstava koja su omogućile da se web primjenski programi po mogućnostima primjene izjednače, pa čak i zamijene klasične primjenske programe.

Primjenski programi napisani za web preglednik gotovo svu primjensku logiku smještaju na udaljene poslužitelje dostupne putem mreže Internet, čime je s korisničke strane potrebna mala količina računalnih sredstava. S druge strane, primjenski program je dostupan u istom obliku bez obzira na vrijeme i mjesto pristupa. Kako se primjenska logika za sve korisnike nalazi na istom poslužitelju, ažuriranje starih i izdavanje novih inačica primjenskih programa vrlo je jednostavno. Ažuriranje se odvija jednokratno, na udaljenom poslužitelju, a korisnici uvijek koriste najnoviju inačicu primjenskog programa.

Kako se primjenski programi za web sve više šire na područja primjene na kojima su se uobičajeno koristili klasični oblici primjenskih programa, sve je veća zastupljenost web primjenskih programa koji se izvode bez stalnog korisničkog nadzora. Takvi primjenski programi sadrže programske funkcionalnosti koje ne zahtijevaju stalno međudjelovanje s korisnikom te funkcionalnosti koje su pokretane vremenom ili događajima iz okoline. Primjeri takvih primjenskih programa su kalendari i podsjetnici, programi za slušanje radijskih ili gledanje televizijskih programa putem Interneta i slično. Takvi web primjenski programi zahtijevaju neprekidan rad web preglednika jer se gašenjem preglednika zaustavlja izvođenje primjenskog programa.

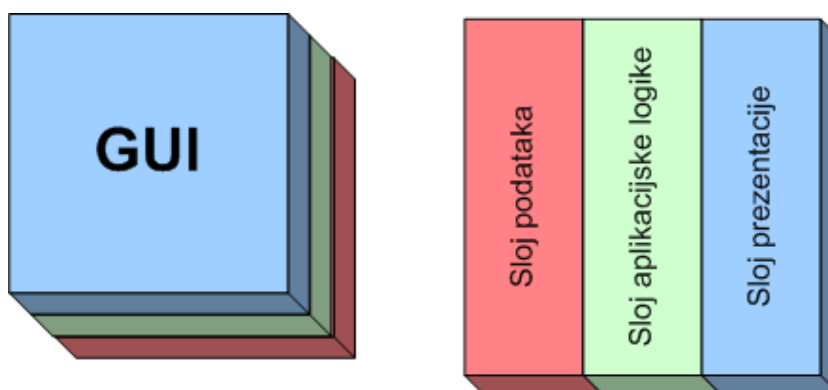
U ovom završnom radu proučeni su mehanizmi pozadinskog izvođenja web primjenskih programa za vrijeme neaktivnosti web preglednika. Programski je ostvaren sustav koji nastavlja izvođenje primjenskih programa učitanih u web preglednik nakon njegova gašenja. Programski sustav ostvaren je u obliku proširenja (eng. *extension*) za preglednik Mozilla Firefox.

Prilikom izrade proširenja koriste se programski jezici bliski operacijskom sustavu te svakodnevno korištene internetske tehnologije. Za ostvarivanje funkcionalnosti na razini operacijskog sustava koriste se XPCOM komponente koje je moguće ostvariti nekim od programskih jezika kao što su C++, Java ili Python. Za programsko ostvarenje funkcionalnosti na razini preglednika koristi se platformski neovisan jezik JavaScript. Za definiranje korisničkog sučelja koristi se derivat jezika XML pod nazivom XUL koji je moguće dodatno opisati pomoću jezika CSS.

2 Primjenski programi unutar web preglednika

Primjenskim programom ili aplikacijom unutar web preglednika podrazumijeva se izvršni softver pokretan pomoću web preglednika dostupan u mrežnom okruženju računala, primjerice putem Interneta ili intraneta. U širu definiciju uvrštava se i softver pokretan u okruženju web preglednika dostupan lokalno koji nema potrebu za korištenjem mreže. Java, JavaScript i Flash samo su neke od najnaprednijih tehnologija korištenih kod izrade aplikacija unutar web preglednika, dok se većina web primjenskih programa najčešće sastoji od više različitih tehnologija kombinirajući optimalnu funkcionalnost svake dostupne tehnologije.

Tipična arhitektura primjenskog programa sastoji se od tri sloja, kao što je prikazano slikom 1: prezentacijskog sloja, tj. korisničkog sučelja, sloja aplikacijske logike te podatkovnog sloja.



Slika 1: Tipična arhitektura primjenskog programa

Preglednik kao okruženje za pokretanje web aplikacija spada pod prezentacijski sloj te mu je primarna svrha interakcija s korisnikom. Kod većine aplikacija podatkovni sloj nalazi se na udaljenom posluživaču. Klijent, točnije aplikacija posluživaču pristupa putem Interneta korištenjem određenog protokola specifičnog za samu aplikaciju. Tako naprednije aplikacije koriste vlastite protokole bazirane na postojećim protokolima niže razine dok jednostavniji primjenski programi koriste već postojeće protokole niže razine kao što su HTTP ili SOAP. Ovisno o količini razmijenjenih podataka s posluživačem, potrebnom rasponu resursa te sadrži li klijent aplikacijsku logiku razlikujemo tanke i debele klijente. Većina web primjenskih programa upravo su tanki klijenti kod kojih se aplikacijska logika nalazi na posluživaču, čime je zauzeće resursa na klijentskoj strani minimalno, te je

primjenskom programu potreban samo prezentacijski sloj za koji je, u konačnici, potrebno razmijeniti poveću količinu informacija s posluživačem. Kod debelog je klijenta aplikacijska logika smještena na klijentu čime je potrebna veća količina lokalnih resursa dok je prijenos podataka gotovo minimalan jer poslužitelj opskrbljuje klijenta sa sirovim i neobrađenim podacima. Kako se aplikacijska logika za sve korisnike nalazi na istom poslužitelju, veoma je jednostavno ažuriranje starih i izdavanje novih inačica primjenskih programa. Ažuriranje se odvija jednokratno na udaljenom poslužitelju, a korisnici uvijek koriste aktualnu najnoviju inačicu primjenskog programa.

Prvi web primjenski programi bili su pravi tanki klijenti, tj. sva obrada podataka odvijala se na strani posluživača koja je klijentskoj strani odgovore slala kao statičke HTML stranice. Prva prava revolucija dogodila se 1995. godine kada tvrtka Netscape objavljuje skriptni jezik JavaScript koji se izvodi na strani klijenta. Već iduće godine Macromedia predstavlja Flash koji posjeduje mogućnost prikazivanja animiranih vektora. Jednostavnom ugradnjom Flasha u bilo koji preglednik na klijentskoj je strani omogućena interakcija s primjenskim programom bez kontaktiranja poslužitelja. Par godina nakon u novoizašloj inačici Microsoftovog web preglednika Internet Explorer 5 u JavaScript se uvodi asinkrona funkcija *XMLHttpRequest*. Iako su i JavaScript i XML postojali dugo prije službena emulzija asinkronog JavaScripta i jezika XML nastaje tek 2005. godine pod nazivom Ajax (*Asynchronous JavaScript and XML*). Kako novonastali Ajax omogućava višu razinu interakcije s korisnikom vrlo je brzo našao primjenu u često korištenim web primjenskim programima kao što su *Gmail* te *Google maps*. Iako JavaScript, Flash i slične moderne tehnologije korištene kod današnjih web primjenskih programa omogućuju da se jednostavniji dio aplikacijske logike odvija na strani preglednika, web primjenski programi i dalje spadaju pod tanke klijente.

Kako se primjenski programi za Web sve više šire na područja primjene na kojima su se uobičajeno koristili klasični oblici primjenskih programa, sve je veća zastupljenost web primjenskih programa koji se izvode bez stalnog korisničkog nadzora. Takvi primjenski programi sadrže programske funkcionalnosti koje ne zahtijevaju stalnu interakciju s korisnikom ili funkcionalnosti koje su pokretane vremenom ili događajima iz okoline. Primjeri najraširenijih takvih primjenskih programa su kalendari i podsjetnici, programi za slušanje radijskih ili gledanje

televizijskih programa putem Interneta itd. Takve web primjenski programi zahtijevaju neprekidan rad preglednika jer se gašenjem preglednika zaustavlja izvođenje primjenskog programa.

Ovisno o programskom jeziku kojim je primjenski program napisan potrebni su mu različiti dijelovi web preglednika. Tako je za programe napisane u JavaScriptu dovoljna samo preglednikova sposobnost prikazivanja (eng. *render*) web stranica. Za Java applete, tj. web primjenske programe napisane u programskom jeziku Java potrebna je funkcionalnost čitavog preglednika koji obično sâm Java applet, radi veće sigurnosti, pokreće u zasebnom programskom okruženju nezavisnom od samog preglednika te s ograničenim pristupom pregledniku i operacijskom sustavu. Takvo se okruženje zove pješčanik (eng. *sandbox*), a ostvaruje se raznim sigurnosnim mehanizmima.

3 Tehnologije za izradu web primjenskih programa

Uobičajene tehnologije za izradu web primjenskih programa su opisni jezici XML i HTML, programski jezik JavaScript i stilski jezik CSS. Ključna svojstva i primjeri uporabe navedenih tehnologija navedeni su u ovom poglavlju.

3.1 XML

XML (Extensible Markup Language) je fleksibilan i jednostavan skup pravila za zapis podataka u obliku obične datoteke. Početkom 1998. godine objavio ga je konzorcij za internetske standarde W3C (*World Wide Web Consortium*), kao jednostavniju i napredniju inačicu starijeg i mnogo šireg jezika SGML (Standard Generalized Markup Language). Danas se koriste peta inačica osnovne XML 1.0 specifikacije te, u manjoj količini, druga inačica XML 1.1 specifikacije.

XML ne propisuje koje se oznake koriste već propisuje pravila definiranja oznaka. XML datoteka obična je tekstualna datoteka koja sadrži XML zaglavlje te korjenski element, dok su podaci zapisani kao stablo elemenata ugniježđeno unutar korjenskog elementa. Prilikom pisanja XML dokumenata koriste se Unicode znakovi, obično UTF-8 kodne stranice. Time je omogućeno korištenje većine znakova iz raznih svjetskih jezika. Imena oznaka, atributa te vrijednosti atributa u jeziku XML razlikuju velika i mala slova te su, primjerice, `<oznaka>` i `<Oznaka>` dvije različite oznake.

Jednostavnost i platformska neovisnost prednosti su jezika XML, koji je danas jedan od najkorištenijih jezika za pohranu podataka. Osim pohrane podataka, XML je, zbog zapisa podataka kao bogato opisanih elemenata, prigodan i za odvajanje sirovih podataka od njihove prezentacije.

Oznaka (eng. *tag*) je definirana kao konstrukcija koja započinje znakom '`<`' te završava znakom '`>`'. XML oznake se dijele na početne oznake, završne oznake te na oznake praznog elementa. Primjeri XML oznaka prikazani su u isječku 1. XML komentari omeđeni su nizom znakova '`<!--`' na početku te nizom znakova '`-->`' na kraju komentara i mogu se protezati kroz više redova.

Logička jedinka koja započinje početnom i završava odgovarajućom završnom oznakom, ili se jednostavno sastoji samo od oznake praznog elementa, zove se element. Svaki element može se detaljnije opisati korištenjem atributa zapisanog

unutar početne oznake u obliku *ključ="vrijednost"*. Isječak 2 prikazuje jednostavne XML elemente opisane atributima.

```
<pocetna_oznaka>
</zavrсна_oznaka>
<oznaka_praznog_elementa/>
<!-- komentar -->
```

Isječak 1: Vrste XML oznaka

```
<oznaka ključ="vrijednost"> Element </oznaka>

<prazan_element verzija="1.0" />
```

Isječak 2: Primjeri elemenata opisanih atributima

Elementi unutar XML dokumenta mogu međusobno biti u odnosu roditelj-dijete, ukoliko je element dijete ugniježđen unutar roditelja, ili, ukoliko posjeduju istog roditelja, u sestrinskom odnosu. U isječku 3 prikazan je međusobni odnos XML elemenata. Elementi unutar XML dokumenta hijerarhijski generiraju objektno stablo XML DOM pri čemu čvor (eng. *node*) stabla predstavlja odgovarajući XML element.

```
<roditelj>
  <dijete> primjer </dijete>
  <prazan_element_dijete sestra_od="dijete" />
</roditelj>
```

Isječak 3: Odnosi između XML elemenata

Na početku svakog XML dokumenta definirano je zaglavlje u kojem se navode korištena verzija i kodna stranica, dok stablo elemenata ugniježđeno unutar jednog korjenskog elementa definira podatke dokumenta. Primjer jednostavnog XML dokumenta s odgovarajućim zaglavljem prikazan je u isječku 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<korjenski_element>
  <rad verzija="1.0">Završni rad</rad>
  <prazan_element />
</korjenski_element>
```

Isječak 4: XML dokument sa zaglavljem i stablom elemenata

Iako po definiciji, XML dokumenti mogu sadržavati proizvoljne oznake, koristeći definiciju tipa dokumenta DTD (*Document Type Definition*), naslijeđenu od jezika

SGML, moguće je definirati raspoloživa imena i raspored ugniježđenosti korištenih oznaka i pripadajućih atributa. Kako je DTD naslijeđen od jezika SGML, nije u potpunosti kompatibilan s novijim XML funkcionalnostima. Koristeći DTD nemoguće je precizno definirati pravila koja XML treba zadovoljavati. Iako danas postoje sheme definirane specifično za XML, DTD se i danas koristi u mnogim web primjenskim programima, prvenstveno zbog veoma jednostavne sintakse te, za razliku od ostalih XML shema, mogućnosti definiranja izravno unutar XML dokumenta.

3.2 HTML

HTML (Hyper Text Markup Language) je standardizirani opisni jezik korišten za zapis internetskih stranica. Sintaksa jezika HTML definirana je po uzoru na standardizirani opisni jezik SGML (Standard Generalized Markup Language). Za razliku od većine ostalih opisnih jezika, HTML se sastoji od konačnog skupa oznaka i pripadajućih atributa.

Imenovanje oznaka i ključeva atributa u jeziku HTML ne razlikuje mala i velika slova. Običaj je pisati imena oznaka velikim slovima te imena atributa i njihove vrijednosti malim slovima. U isječku 5 prikazani su primjeri HTML oznaka.

```
<A href="http://www.fer.hr">Naslovnica fer.hr</A>  
<BR> <!-- html oznaka za novi red -->
```

Isječak 5: Primjer HTML oznaka

Osim konačnog skupa oznaka, HTML dokumenti strogo su definirani osnovnom konstrukcijom gdje su zaglavlje i tijelo dokumenta ugniježđeni unutar korjenskog *<HTML>* elementa.

Isječak 6 prikazuje osnovnu strukturu HTML dokumenta napisanog po HTML 4.01 standardu te HTML oznakama praznog prostora za tekst identificiranog kao *prostor* i neidentificiranog gumba. Izgled HTML stranice iz isječka 6 u internetskom pregledniku Internet Explorer prikazan je na slici 2.

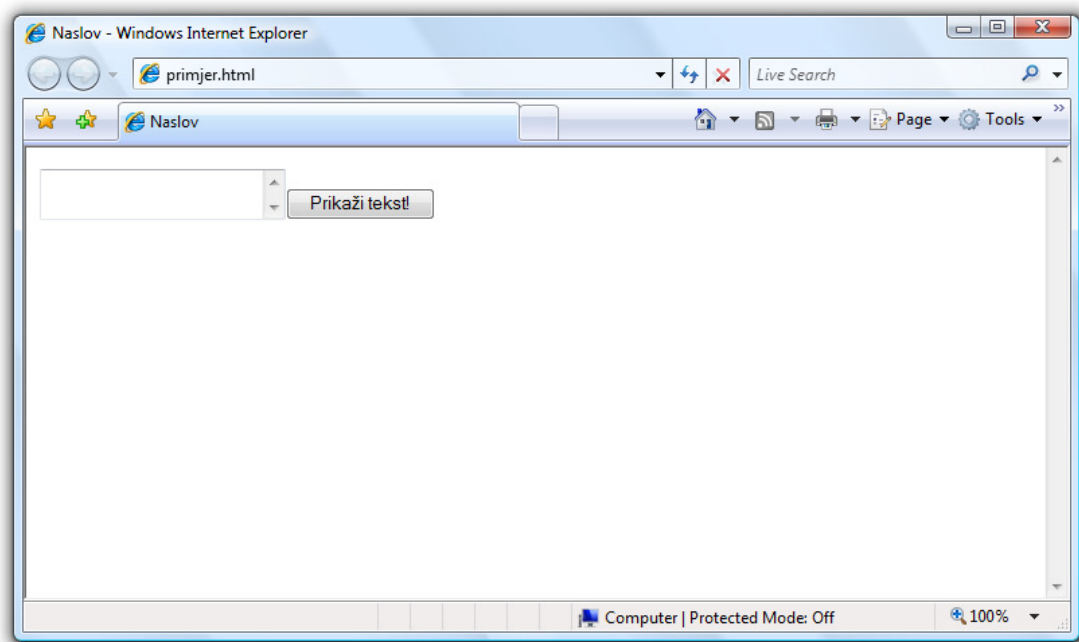
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
  <HEAD>
    <TITLE>Naslov</TITLE>
  </HEAD>

  <BODY>

    <TEXTAREA id="prostor"></TEXTAREA>
    <BUTTON type="button">Prikaži tekst!</BUTTON>

  </BODY>
</HTML>
```

Isječak 6: Osnovna struktura HTML dokumenta



Slika 2: Izgled jednostavne HTML stranice *primjer.html*

3.3 CSS

CSS (Cascading Style Sheets) je stilski jezik za opis prezentacijske semantike dokumenta napisanog u opisnom jeziku, primjerice u jeziku XML ili HTML. Prvenstveno služi za odvajanje prezentacije od sirovih podataka opisnog dokumenta. CSS se može nalaziti u zasebnoj datoteci, ili može biti ugrađen izravno u HTML stranicu. CSS se ugrađuje u HTML elementom `<STYLE type="text/css">` smještenog u zaglavlju dokumenta.

CSS stilske stranice obično bivaju pospremljene u priručnu memoriju web preglednika čime je omogućeno višekratno korištenje stilske stranice što

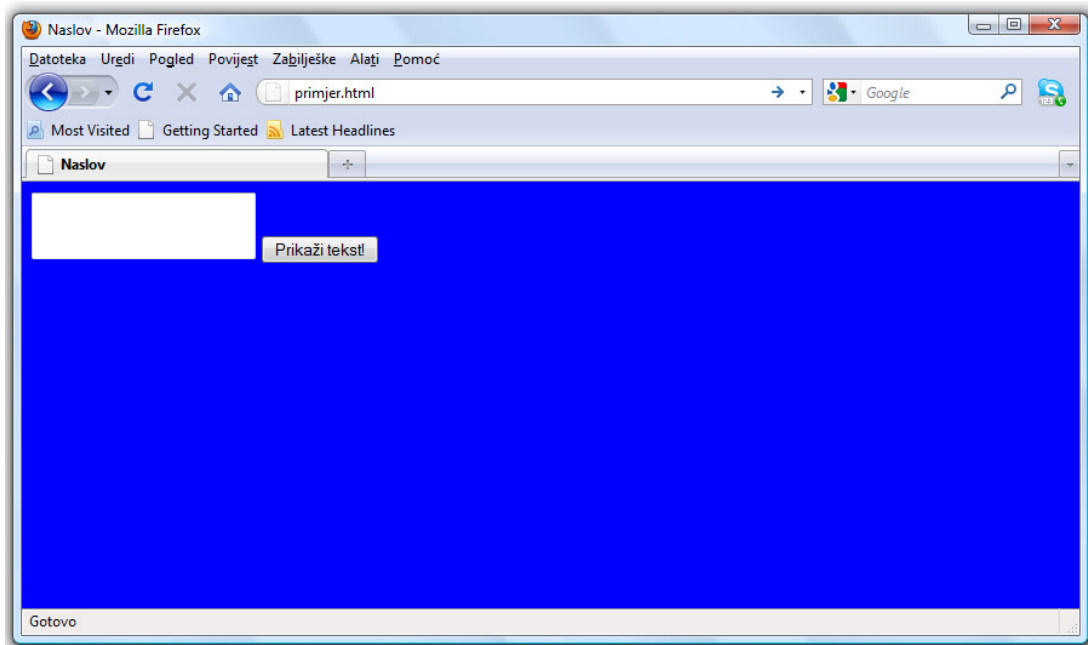
automatski povećava brzinu dohvaćanja podataka i smanjuje količinu zaprimljenih podataka.

CSS datoteka se sastoji od skupa semantičkih pravila. Svako pravilo sastoji se od barem jednog selektora (eng. *Selector*) popraćenog blokom deklaracija. Selektor definira na koji se element opisnog dokumenta pravilo odnosi. Selektori tako mogu definirati oznaku elementa, ime elementa (određeno atributom *name*) element roditelj. Blok deklaracija definiran je kao skup od jedne ili više deklaracija omeđen uglatim zagladama. Svaka deklaracija sastoji se svojstva, dvotočke, vrijednosti te točke-zareza.

U isječku 7 prikazan je primjer CSS pravila. Selektor *body* označava da pravilo odnosi na HTML element *body*, a svojstvo *background-color* označava boju pozadine elementa. Slika 3 prikazuje HTML stranicu iz isječka 6 s pridruženim CSS pravilom iz isječka 7 u okruženju Mozilla Firefox web preglednika.

```
body {  
  background-color:green;  
}
```

Isječak 7: CSS Pravilo



Slika 3: HTML stranica *primjer.html* s pridruženim CSS

3.4 JavaScript

JavaScript je skriptni jezik koji se izvodi na strani klijenta, a razvio ga je Netscape 1995. godine. Prvotno je služio za dinamičku izmjenu HTML dokumenata, međutim, sposobnosti JavaScripta odavno su nadišle njegovu početnu funkcionalnost. Kako ga je većina programera koristila prilikom jednostavnijih zadataka, za JavaScript se dugo mislilo da je beskoristan, a s otkrićem prvih sigurnosnih mana mnogi su odustali od korištenja JavaScripta. Međutim, nastankom Ajaxa 2005. godine lepeza mogućnosti JavaScripta uvelike je uvećana te je JavaScript danas najkorištenija tehnologija Web 2.0 generacije web primjenskih programa. Sposobnosti asinkronih poziva funkcija i korištenja DOM funkcija te neovisnost o platformi samo su neke od prednosti koje JavaScript čine neizostavnim dijelom gotovo svakog modernog web primjenskog programa.

Sintaksa JavaScripta bazirana je na sintaksi programskog jezika C. Za razliku od ostalih jezika baziranih na jeziku C, JavaScript posjeduje dinamičke tipove, tj. sve varijable su tipa *var*, te je u njih moguće pohraniti bilo koji oblik informacije. JavaScript je gotovo u potpunosti objektno orijentiran te koristi prototipove umjesto uobičajenog sustav razreda i njihovog nasljeđivanja. Kako JavaScript ne razlikuje metode od klasičnih funkcija, konstruktori objekata također su obične funkcije.

Iako postoje i druge inačice, JavaScript se i danas najčešće koristi u originalnom obliku izvođenom s klijentske strane, te je implementiran kao integrirana komponenta web preglednika čime je omogućen razvoj kvalitetnijih korisničkih sučelja i izrada dinamičkih web primjenskih programa.

Većina HTML elemenata podržava postavljanje oslušivača događaja (eng. *eventlistener*) koji oslušuju akcije korisnika nad tim elementom. Oslušivači za događaje kao što su klik miša, prijelaz miša preko elementa ili označavanje elementa postavljaju se kao odgovarajući atribut unutar oznake željenog elementa. U isječku 8 prikazan je gumb iz isječka 6 s pridodanim atributom *onclick* koji predstavlja akciju korisnikovog klika na gumb. Vrijednost pridodana atributu je ime JavaScript funkcije koja se poziva prilikom pripadajuće korisnikove akcije, u slučaju iz isječka 8, korisnikovog klika na gumb.

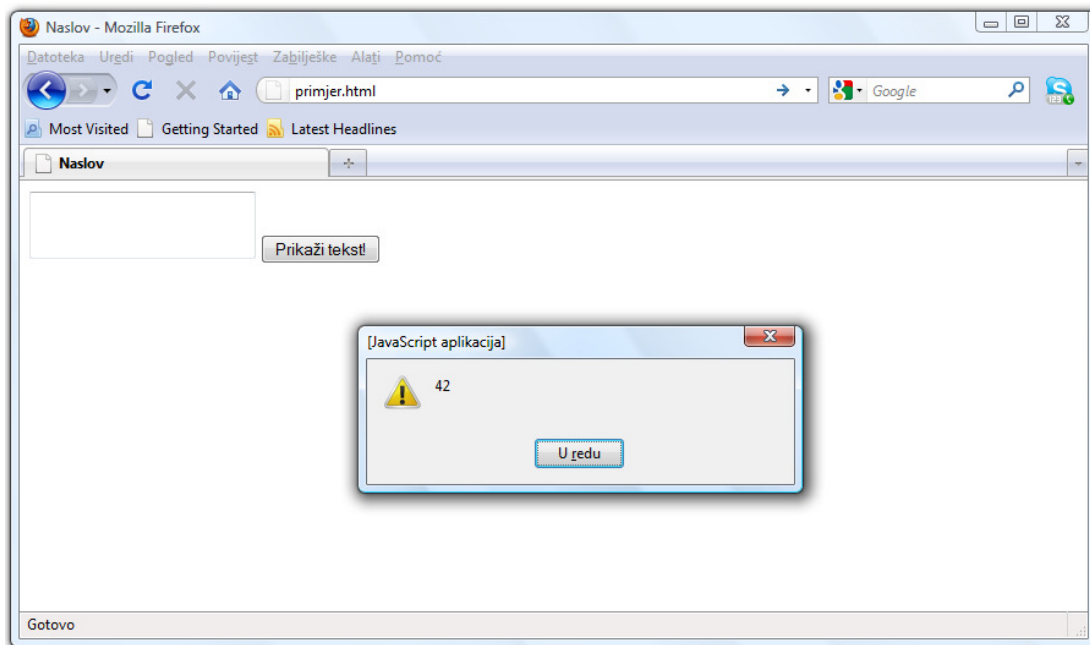
```
<BUTTON type="button" onclick="akcija()">Prikaži tekst!</BUTTON>
```

Isječak 8: Oznaka gumba s pridodanim atributom korisničke akcije

JavaScript kôd se u HTML stranici zapisuje unutar oznake `<SCRIPT type="text/javascript">` smještene u zaglavlju stranice. U isječku 9 prikazana je JavaScript funkcija `akcija()` zapisana unutar odgovarajuće HTML oznake. U varijablu `tekst` pospremljen je broj, a zatim je pozvana standardna funkcija preglednika `alert()`. Funkcija `alert()` prikazuje novi prozor preglednika u kojem se ispisiuje parametar predan toj funkciji. Na slici 4 prikazan je izgled dokumenta `primjer.html` u web pregledniku Mozilla Firefox nakon korisnikovog klika na gumb.

```
<SCRIPT type="text/javascript">
  function akcija(){
    var tekst = 42;
    window.alert(tekst);
  }
</SCRIPT>
```

Isječak 9: JavaScript funkcija kao dio HTML elementa



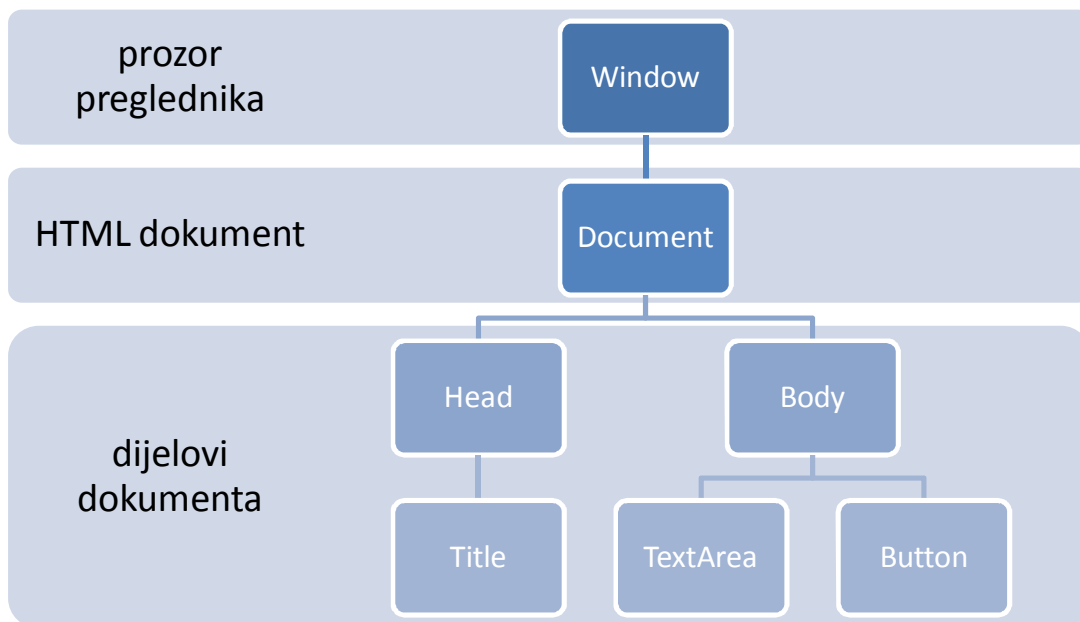
Slika 4: Izgled stranice `primjer.html` nakon korisnikove akcije

3.4.1 DOM

DOM (Document Object Model) je tehnički standard za reprezentaciju dokumenata napisanih u opisnom jeziku, primjerice u jeziku XML ili HTML. Iako su mnogi programski jezici implementirali DOM programsko sučelje, i danas je najčešće korištena izvorna JavaScript implementacija DOM standarda.

Prvi DOM, izdan 1996. godine, zove se *Legacy DOM* ili DOM nulte razine te se razvijao zasebno za svaki web preglednik. Nastao je kao dio JavaScripta koji je omogućavao dohvaćanje vrijednosti određenih elemenata smještenih unutar HTML dokumenta. DOM nulte razine dozvoljava JavaScriptu pristup HTML elementima putem njihovog imena zadanog atributom *name*. W3C 1998. godine donosi prvi DOM standard nazvan DOM razine 1. Za razliku od DOM standarda nulte razine koji je nudio dohvaćanje vrijednosti pojedinih elemenata, DOM prve razine donio je kompletan model za obradu čitavog HTML ili XML dokumenta te metode za dinamičku izmjenu bilo kojeg dijela dokumenta u realnom vremenu. DOM razine 2, izdan 2000. godine, podržava dohvaćanje elemenata dokumenta i putem njegove identifikacije funkcijom *getElementById*. Najnovija je inačica DOM razine 3, izdan je 2004. godine. Među ostalim, DOM razine 3 je donio i podršku za XML upitni jezik XPath.

Iako DOM nije obavezna komponenta web preglednika, većina prikaznih mehanizama (eng. *Layout mechanism*) popularnih web preglednika imaju sposobnost parsiranja HTML i XML dokumenata u DOM strukturu. Parsiranjem dokumenta stvara se stablasta DOM struktura pri čemu čvor (eng. *node*) stabla odgovara elementu parsiranog dokumenta. DOM struktura HTML dokumenta iz isječka 6 prikazana je na slici 5.

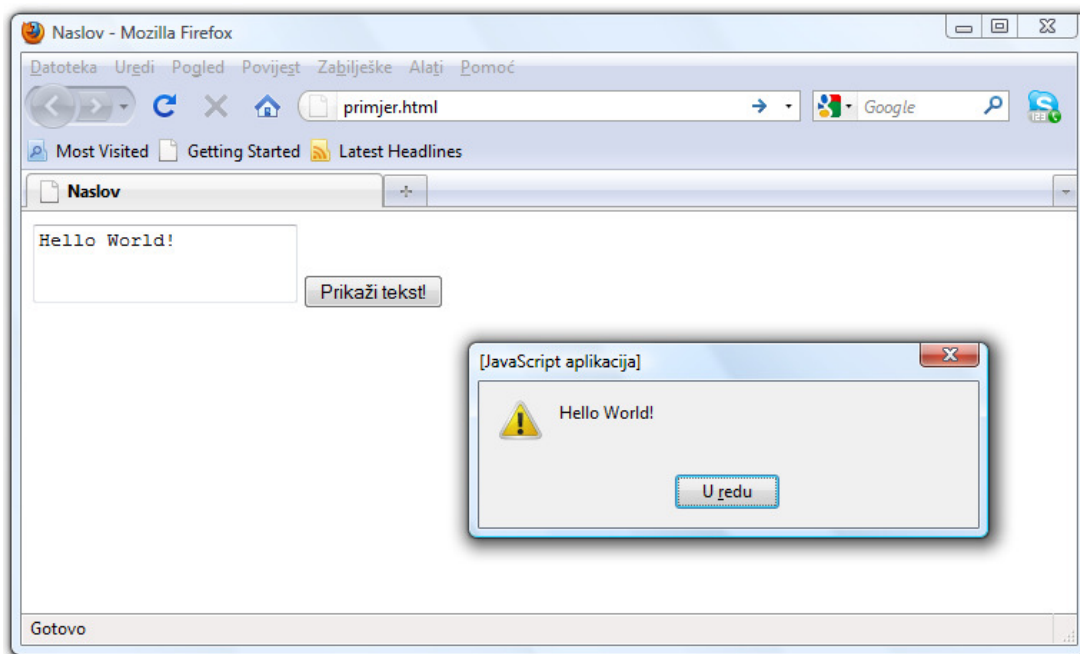


Slika 5: DOM struktura jednostavnog HTML dokumenta

Funkciju iz isječka 9 moguće je izmijeniti tako da se varijabli *tekst*, umjesto statičke vrijednosti, korištenjem DOM funkcija dinamički pridodaje vrijednost HTML elementa Textarea iz isječka 6. Izmijenjena funkcija prikazana je u isječku 10. Vrijednost HTML elementa Textarea dohvaća se pomoću DOM naredbe *getElementById* predajući joj kao parametar ime kojim je element identificiran. Na slici 6 prikazan je izgled dokumenta *primjer.html* u web pregledniku Mozilla Firefox nakon korisnikovog klika na gumb.

```
function akcija(){
    var tekst = document.getElementById('prostor').value;
    window.alert(tekst);
}
```

Isječak 10: Upotreba DOM funkcionalnosti u JavaScriptu



Slika 6: Izgled stranice *primjer.html* nakon korisnikove akcije

4 Proširenja preglednika

Proširenjem preglednika smatra se dio softvera koji proširuje funkcionalnost web preglednika. Gotovo sve popularnije preglednike današnjice moguće je nadograditi koristeći proširenja. Govoreći o naraširenijim preglednicima, Mozilla Firefox osmišljen je kao jednostavan preglednik koji kompliciranije funkcionalnosti prepušta svojim proširenjima, dok Internet Explorer podržava proširenja tek od inačice 4.0.

4.1 Preglednik Mozilla Firefox

Mozilla Firefox besplatan je internetski preglednik *open source* tipa te ga je moguće pokrenuti na većini današnjih operacijskih sustava kao što su Windows, Mac OS X, Linux itd.

Izvorni kôd Firefoxa besplatan je i dostupan za proučavanje, korištenje, proučavanje i modificiranje bez ograničenja te kopiranje bez ograničenja ili s minimalnim ograničenjima. Iste je objavljen pod trostrukom licencom koja je kombinacija GPL (GNU General Public License), LGPL (GNU Lesser General Public License) i MPL (Mozilla Public License).

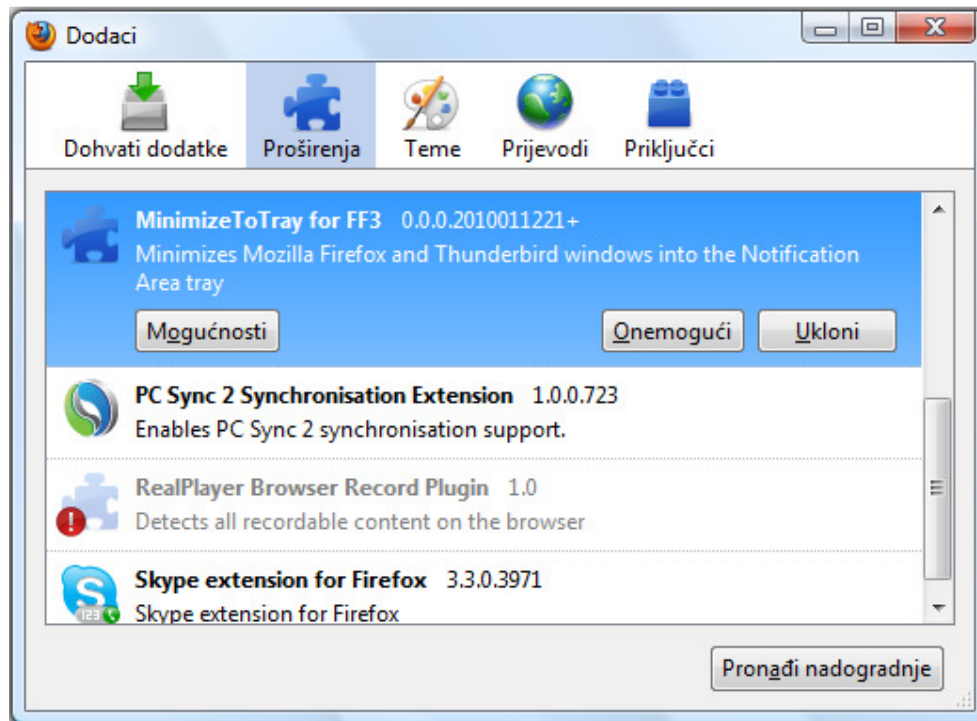
Izvorni kôd Firefoxa zadužen za ostvarivanje funkcionalnosti na razini operacijskog sustava napisan je u programskom jeziku C++, dok je funkcionalnost na razini preglednika ostvarena JavaScriptom. Korisničko sučelje definirano je jezikom XUL koji je dodatno opisan pomoću jezika CSS. Za prikazivanje web stranica (eng. *rendering*) koristi se prikazni mehanizam Gecko, neovisan od samog sučelja Firefoxa.

Sučelje Firefoxa izgrađeno je minimalistički. Firefox, radi jednostavnosti korištenja, sadrži osnovnu funkcionalnost preglednika što je izravno pridonijelo njegovoj brzini rada. Minimalistički prikaz i brži rad u odnosu na konkurenciju uvelike su zaslužni za popularizaciju Firefoxa. Svaku željenu dodatnu funkcionalnost preglednika moguće ostvariti kao Mozilla dodatak (eng: *add-on*).

4.2 Mozilla dodaci

Mozilla dodatak je programska nadogradnja nekog od Mozillinih projekata, primjerice Firefoxa ili Thunderbirda. Za proces instalacije i sigurnog uklanjanja dodataka brine se program za upravljanje dodacima *Add-on Manager* prikazan

slikom 7. Osim instalacijske podrške zadužen je i za privremeno onemogućavanje, izmjenu postavki te ažuriranje (eng. *update*) dodataka. Brine se i o kompatibilnosti dodatka s projektom za koji se instalira te pruža uvid u stranice pomoći i programske potpore.



Slika 7: Upravljanje dodacima u *Add-on Manageru*

Mozillin dodatak po svom tipu može biti proširenje, priključak, prijevod ili tema. Tipovi i kratak opis njihove funkcionalnosti prikazani su u tablici 2.

Tablica 2: Podjela mozillinih dodataka

Vrsta dodatka	Funkcionalnost
Proširenje	proširuje funkcionalnost preglednika ili dodaje funkcije i metode kojima će se promijeniti izgled ili funkcionalnost željenih web stranica
Priključak	pregledniku dodaje opciju korištenja vanjskih programa kao što su Acrobat Reader, Flash Player ili Java
Prijevod	lokalizira korisničko sučelje
Tema	izmjenjuje vizualni izgled preglednika

4.3 Mozilla proširenja

Mozilla proširenje izmijenjuju postojeće ili pridodaje sasvim novo ponašanje Mozillinom projektima za koji je namijenjeno. Svi Mozillini projekti podržavaju višejezičnost te je veoma jednostavno napisati proširenje lokalizirano na više jezika ili lokalizirati postojeće proširenje.

S obzirom na funkcionalnost, tehnologije korištene kod svih Mozillinih projekata i njihovih proširenja mogu se podijeliti na četiri razine prikazane na slici 8. Jezgrena razina sposobna za funkcionalnost na razini operacijskog sustava zove se XPCOM. Nadalje, razina specifična za preglednik zadužena za definiranje korisničkog sučelja zove se XUL. Koristeći JavaScript ostvaruje se dodatna funkcionalnost preglednika. U konačnici, koristeći CSS specificira se izgled korisničkog sučelja na razini korisnika.



Slika 8: Podjela tehnologija korištenih kod Mozillinih proširenja

4.3.1 XPCOM

XPCOM (Cross-Platform Component Object Model) je Mozillin objektni model neovisan o platformi. Za izgradnju XPCOM komponente potrebni su C++ kompajler, kojeg većina operacijskih sustava posjeduje, te perl interpreter.

Sučelja XPCOM komponenti obično su napisana u jeziku za opis sučelja IDL (Interface Description Language). Opisivanje sučelja u jeziku IDL česta je praksa kod pisanja Firefox ekstenzija pošto je IDL neovisan o platformi. Implementacija sučelja piše se odvojeno od sučelja posebno za svaki od podržanih operacijskih sustava.

Prvi korak prilikom izrade sučelja za XPCOM komponente jest generiranje univerzalnog jedinstvenog ključa (eng. *UUID*) za svoje razrede. Najjednostavniji način generiranja ključeva je korištenje nekog od postojećih alata za generiranje ključeva. Drugi je korak izrada UUID-a za svako od pretpostavljenih sučelja. Zatim se svakom od jedinstvenog ključa razreda pridružuje tekstualni ključ *contract ID*. Preporučeni format tekstualnog ključa naveden je u isječku 11 gdje su neobavezni dijelovi naznačeni uglatim zagradama. U isječku 12 naznačeni su neki od Mozillinih tekstualnih ključeva.

```
@<internetska_domena>/modul[/podmodul[...]];<verzija>[?<ime>=<vrijednost>[...]]
```

Isječak 11: Preporučeni format tekstualnog ključa

```
@mozilla.org/file/directory_service;1  
@mozilla.org/file/local;1  
@mozilla.org/file;1
```

Isječak 12: Primjeri Mozillinih tekstualnih ključeva

U idućem koraku definiraju se sučelja unutar željenog IDL sučelja koristeći prethodne tekstualne ključeve. Sintaksa jezika IDL veoma je slična obitelji programskih jezika C. U isječku 13 prikazan je primjer sučelja napisanog u jeziku IDL. Nakon opisivanja sučelja slijedi kompajliranje IDL sučelja koristeći XPIDL kompajler. Kompajler kreira XPT datoteku s pripadajućom knjižnicom tipova te šablonu za C++ komponentu.

```
[scriptable, uuid(12345678-1234-5678-9ABC-123456789ABC)]  
interface expampleInterface : nsISupports  
{  
    boolean Open (in string strDevice, out double timeout);  
    boolean Close ();  
    readonly attribute string source;  
};
```

Isječak 13: Primjer sučelja XPCOM komponente

Iduća stavka prilikom izgradnje XPCOM komponente jest implementacija tih sučelja zasebno za svaku platformu na kojoj će se izvoditi. Kako se XPCOM komponenti dodaje funkcionalnost preglednika na razini operacijskog sustava, tj. koriste se jezgrene funkcije operacijskog sustava, za pisanje komponente koristi se C++. Korištenjem raznih mostova omogućeno je kombiniranje jezgrenog C++ s

drugim jezicima, pa je naprednije programske zadatke moguće napisati u jezicima kao što su JavaScript, Java i Python. Konkretno, most prema JavaScriptu zove se XPConnect ili kraće XPT.

XPI (Cross-Platform Installer) je datoteka kojom se vrši instalacija proširenja na neki Mozillin projekt. XPI datoteka je ustvari običan skup direktorija i datoteka kompresiran ZIP tehnologijom.

4.3.2 XUL

XUL (XML User Interface Language) je Mozillin jezik, baziran na jeziku XML, koji omogućava izgradnju aplikacija bogatih funkcionalnosti s opcijom pokretanja na različitim platformama. Koristi se za definiranje korisničkog sučelja (eng: *User Interface*) kod Mozillinih proširenja. Sučelje definirano u jeziku XUL moguće je dodatno opisati pomoću CSS te lokalizirati koristeći neku od XML opisnih shema, primjerice DTD. Tablica 3 prikazuje podjelu XUL oznaka po funkcionalnosti u 13 kategorija.

Tablica 3: Kategorije XUL oznaka

Kategorija	Često korištene oznake
prozor (eng. <i>window</i>)	dialog, overlay, page, window, wizard, wizardpage...
struktura prozora (eng. <i>window structure</i>)	browser, tabbrowser, editor, iframe, titlebar, statusbar, notification...
meniji i iskočni izbornici (eng. <i>menus and popups</i>)	menubar, menu, menuitem, menuseparator, menupopup, panel, tooltip, popupset
alatna traka (eng. <i>toolbar</i>)	toolbar, toolbarbutton, toolbaritem, toolbarset, toolbox...
tabovi i grupe (eng. <i>tabs and groupings</i>)	tabbox, tab, tabpanel, groupbox, separator...
kontrolne (eng. <i>controls</i>)	button, checkbox, menulist, radio, textbox...
tekst i slike (eng. <i>text and images</i>)	description, label, image
liste (eng. <i>lists</i>)	listbox, listcell, listitem, listcol, listheader...
stabla (eng. <i>trees</i>)	tree, treechildren, treeitem...
izgled (eng. <i>layout</i>)	box, stack, grid, row, column...
predlošci (eng. <i>templates</i>)	action, binding, query, where...
skriptni kôd (eng. <i>scripting</i>)	script, command, key, keyset...
pomoćni elementi (eng. <i>helper elements</i>)	dropmaker, scrollbar, grippy...

Sintaksa jezika XUL veoma je slična sintaksi jezika XHTML. Korjenski XUL element obično se uzima iz kategorije prozora. XUL prostor imena (eng. *namespace*) naznačen je unutar korjenskog elementa kao atribut `xmlns` s vrijednosti "`http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul`".

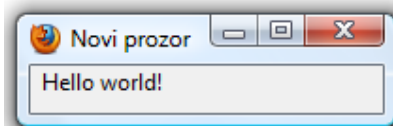
Atributom `id` unutar pripadajuće oznake identificira se element koji oznaka predstavlja. Kako su poznate identifikacije čitavog Mozillinog sučelja moguće je referencirati se na neki od njegovih elemenata te mu izmijeniti ili pridodati novi opis.

U zaglavlju XUL dokumenta nalazi se uobičajeno XML zaglavlje. Nakon zaglavlja mogu se, po potrebi, nalaziti i CSS kôd ili referenca na CSS kôd kojim je opisano sučelje definirano unutar XUL dokumenta te referenca na direktorij u kojem se nalaze lokalizirane DTD sheme. Isječak 14 prikazuje XUL dokument jednostavan vlastiti prozor s tradicionalnim Mozilla Firefox izgledom, dok je njegov konačan izgled prikazan na slici 9.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window
  id="noviProzor"
  title="Novi prozor"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <label id="labela"> Hello world! </label>
</window>
```

Isječak 14: Dokument *noviProzor.xul*



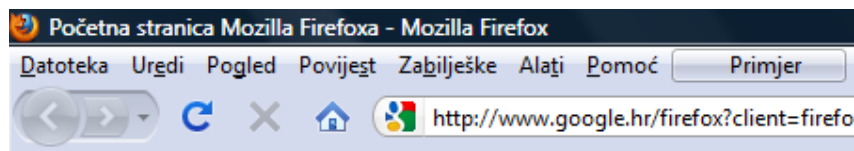
Slika 9: Izgled prozora *noviProzor*

Osim kreiranja novih prozora i elemenata moguće je i izmijeniti ili nadopuniti postojeće sučelje uobičajenog Firefox prozora. Izmjene sučelja pišu se unutar korjenskog elementa oznake *overlay* koja označava da se ugniježđeni XUL kôd pridodaje izvornom XUL kôdu preglednika i ostalih proširenja. Sučelje se nadopunjuje tako da se unutar postojećeg elementa sučelja ugnijezdi nova, vlastita, komponenta. U isječku 15 prikazano je dodavanje novog gumba *primjer* u

postojeću glavnu izborničku traku. Komponenta *primjer* identificira se vrijednošću pridodanom atributu *id*, dok atribut *label* označava tekst gumba, a atribut *oncommand* JavaScript funkciju koja će se pozvati prilikom pritiska na gumb. Komponenta *primjer* biti će pridodana ostalim elementima izborničke trake *main-menubar*. Vizualni prikaz novog gumba u okruženju izborničke trake vidljiv je na slici 10.

```
<menubar id="main-menubar">
  <menuitem
    id="primjer"
    label="Primjer"
    oncommand="prikaziPrimjer()"
  />
</menubar>
```

Isječak 15: Dodavanje komponente *primjer* u glavnu izborničku traku



Slika 10: Izgled novog gumba u glavnoj izborničkoj traci

4.3.3 Programski jezik JavaScript

Kod Mozillinih proširenja JavaScript ima funkciju mišićnog tkiva koje povezuje akcije na korisničkom sučelju s funkcionalnostima preglednika. Funkcije na razini samog preglednika neovisno o platformi na kojoj se izvodi, napisane su izravno u JavaScriptu, dok kod korištenja XPCOM komponenti služi kao most prema komponenti.

Upravljanje korisničkim sučeljem

Korištenje DOM funkcija JavaScriptu omogućuje laganu manipulaciju, te jednostavno upravljanje korisničkim sučeljem definiranim jezikom XUL. Osim toga, DOM je moguće koristiti i za izmjenu HTML kôda prilikom učitavanja.

U prijašnjim načinima dinamičke izmjene HTML dokumenata koristio se JavaScript, konkretnije svojstvo *innerHTML* nekog HTML elementa, međutim, DOM omogućava kvalitetniju manipulaciju XML dokumentima, a kako XUL jeziku nedostaje ekvivalent svojstvu *innerHTML* manipulacija XUL dokumentima bez korištenja DOM funkcija gotovo je nezamisliva.

JavaScript funkcija pozvana kao rezultat korisničke akcije na komponenti iz isječka 15 prikazana je u isječku 16.

```
function prikaziPrimjer(){
    window.open("chrome://minimizetotray/content/noviProzor.xul",
                "prozor",
                "chrome,width=200");
}
```

Isječak 16: JavaScript funkcija *prikaziPrimjer()*

DOM funkcionalnost rasprostire se na tri razine. Prethodno spomenuti mehanizam Gecko podržava gotovo sve DOM funkcije druge razine te neke funkcije treće razine.

Korištenje XPCOM komponenti

JavaScriptu je omogućeno instancirati njemu nepoznati razred pisan u drugom programskom jeziku ukoliko posjeduje referencu na sučelje razreda napisano u jeziku IDL. Dodirna točka između JavaScripta i implementacije XPCOM komponente je IDL sučelje komponente na koje se u JavaScriptu referencira koristeći tekstualni ključ *contract ID*.

4.3.4 CSS

CSS se koristi za opis izgleda HTML i XML dokumenata, a kako su sučelja Mozillinih projekata, kao i njihovih proširenja, napisana u jeziku XUL, i njihov se izgled također može opisati koristeći CSS.

Bez korištenja neke stilske stranice kod prozora iz isječka 14 Firefox ne bi znao kako prikazati *noviProzor*. Tako prozor iz isječka 14 posjeduje referencu na CSS stilsku stranicu korištenu za standardni izgled Firefox prozora. Standardna je stilska stranica smještena lokalno, a nalazi se na putanji *"chrome://global/skin/"*.

5 Sustav za pozadinsko izvođenje web primjenskih programa

U ovom poglavlju je opisano kako ostvariti Firefox proširenje koje nastavlja izvođenje primjenskih programa učitanih u web preglednik nakon njegova gašenja. Struktura tehnologija proširenja prikazana je u tablici 4.

Tablica 4: Struktura tehnologija proširenja

<i>korisnik</i>		
<i>funkcionalnost</i>	<i>tehnologije korištene za dopunu funkcionalnosti</i>	<i>tehnologije korištene za upravljanje korisničkim sučeljem</i>
na razini preglednika	JavaScript (izmjena funkcionalnosti preglednika)	
		XUL (dopuna korisničkog sučelja)
	IDL (XPCOM sučelja)	
na razini operacijskog sustava	C++ (implementacija sučelja)	
<i>operacijski sustav</i>		

5.1 Programska knjižnica *Minimize to Tray*

Kao XPCOM dio proširenja koristi se postojeći *Minimize to Tray* programsku knjižnicu. Projekt je započet početkom 2004. godine da bi sredinom 2005. originalni autor Mark Yen prestao raditi na projektu. Kako je sâm projekt otvorenog kôda danas postoji nekoliko projekata s referencom na izvornu programsku knjižnicu. Tablica 5 prikazuje programsko sučelje koje se sastoji od tri sučelja napisana u jeziku IDL.

Tablica 5: Sučelja *Minimize to Tray* programske knjižnice

ime sučelja	opis sučelja
<i>trayWindowWatcher</i>	daje JavaScriptu informaciju o događajima koji se odvijaju na prozoru preglednika s razine operacijskog sustava
<i>trayWindowHider</i>	sakriva prozore na razini operacijskog sustava u prostor za obavijesti te obavještava o događajima na ikoni (lijevi klik miša i sl.)
<i>trayWindowCallback</i>	se koristi kao povratna funkcija za događaje iz binarne komponente. Potrebno ga je implementirati u JavaScriptu.

Uz programsko sučelje, knjižnica *Minimize to Tray* sadrži i implementacije sučelja *trayWindowWatcher* i *trayWindowHider* za 32-bitne Windows operacijske sustave napisane u programskom jeziku C++. Obje implementacije jednostavan su presjek Mozillinih i Microsoftovih biblioteka gdje se Mozillin prozor tretira kao prozor unutar Microsoft operacijskog sustava nad kojim se zatim poziva jezgrena funkcija operacijskog sustava *ShowWindow()* koja ovisno o predanim parametrima sakriva ili otkriva Mozillin prozor. Sučelja implementirana na razini operacijskog sustava, tj. *trayWindowWatcher* i *trayWindowHider* u platformno neovisnom dijelu proširenja dostupna su pomoću njihovih tekstualnih ključeva (*Contract ID*) navedenih u Isječku 17.

```
@minimizetotray.mozdev.org/minimizetotray/window-watcher;1
@minimizetotray.mozdev.org/minimizetotray/window-hider;1
```

Isječak 17: Tekstualni ključevi sučelja implementiranih u API-ju

Sučelje *trayWindowsCallback* nije implementirano u programskoj knjižnici već ga je potrebno samostalno implementirati. Kako je funkcionalnost sučelja na razini preglednika, implementirati će se u JavaScriptu. Sučelje sadrži konstantne varijable prikazane u tablicama 6-9 koje predstavljaju korisnikove akcije nad preglednikom. Konstante služe kao statičke konstante te su korištene i u implementacijama drugih sučelja.

Tablice 6-9: Konstante za korisnikove akcija nad prozorom

korisnikove akcije mišem
EVENT_MOUSEDOWN_LEFT
EVENT_MOUSEDOWN_RIGHT
EVENT_MOUSEDOWN_MIDDLE
EVENT_MOUSEUP_LEFT
EVENT_MOUSEUP_RIGHT
EVENT_MOUSEUP_MIDDLE
EVENT_MOUSEDBLCLK_LEFT
EVENT_MOUSEDBLCLK_RIGHT
EVENT_MOUSEDBLCLK_MIDDLE

promjena stanja prozora
EVENT_WINDOW_ACTIVATE
EVENT_WINDOW_MINIMIZE
EVENT_WINDOW_CLOSE

akcija nad prozorom korištenjem miša
PARAM_MOUSE_MINIMIZE
PARAM_MOUSE_MAXIMIZE
PARAM_MOUSE_CLOSE

modifikatori akcija mišem
KEY_CONTROL
KEY_ALT
KEY_SHIFT

Programska knjižnica dostupna je kao DLL (*Dynamic Link Library*) biblioteka. Programsku knjižnicu moguće je nadograditi te izgraditi novu biblioteku za što potrebno kompajlirati izvorni kôd Firefoxa.

5.2 Izmjena standardnih funkcionalnosti preglednika

Za izmjena standardne funkcionalnosti preglednika, kao što je gašenje, potrebno je kreirati osluškivač za akcije nad prozorom preglednika. Taj zadatak pripada JavaScriptu kojim se, prilikom kreiranja novog prozora preglednika, instancira i inicijalizira XPCOM komponenta *windowWatcher* iz *Minimize to Tray* programske knjižnice. Postupak instanciranja i inicijalizacije *windowWatcher* komponente prikazan je u isječku 18. Svrha komponente *windowWatcher* je nadgledanje svih korisnikovih akcija nad prozorom preglednika. JavaScriptu je omogućeno instancirati njemu nepoznati razred napisan u drugom programskom jeziku ukoliko posjeduje referencu na sučelje razreda napisano u IDL-u. Tako je dodirna točka između JavaScripta i XPCOM komponente IDL sučelje komponente. Prije

instanciranja potrebno je dohvatiti razred iz liste postojećih razreda *Components.classes[]* koristeći njegov tekstualni ključ *contract ID*. Pozivom funkcije *createInstance()* s predanim imenom sučelja instancira se *windowWatcher* koji se potom inicijalizira funkcijom *init* kojoj se kao parametri predaju osnovni prozor Firefoxa te JavaScript funkcija *windowWatcherCallback* koja će se pozivati nakon korisnikove akcije nad Firefox prozorom.

```
self = window.extensions.mook.minimizetotray;
self.k_contractid_windowWatcher =
    '@minimizetotray.mozdev.org/minimizetotray/window-watcher;1';

self.m_windowWatcher =
    Components.classes[self.k_contractid_windowWatcher].
        createInstance(Components.interfaces.trayIWindowWatcher);

self.m_windowWatcher.init(
    self.getBaseWindow(window),
    self.windowWatcherCallback);
```

Isječak 18: Instanciranje i inicijalizacija komponente *windowWatcher*

Funkcija *windowsWatcherCallback* je pozvana od strane prethodno instancirane i inicijalizirane funkcije *windowWatcher* prilikom svake korisnikove akcije nad prozorom preglednika. Svrha *windowsWatcherCallback* funkcije jest ovisno o postavljenim zastavicama iz tablica 6-9 sakriti prozor preglednika. Isječak 19 prikazuje izgled funkcije *windowsWatcherCallback* koja maskira standardnu opcija zatvaranja (eng. *close*) te umjesto toga sakriva prozor preglednika.

```
window.extensions.mook.minimizetotray.windowWatcherCallback =
function( aEvent, aParam, aKeyMask, aData1, aData2) {
    var self = window.extensions.mook.minimizetotray;
    var iface = Components.interfaces.trayIWindowCallback;
    var rv = false;
    switch (aEvent) {
        case iface.EVENT_WINDOW_CLOSE:
            // zatvaranje Alt+F4 opcijom
            self.minimizeWindow();
            rv = true;
            break;
        case iface.EVENT_MOUSEDOWN_LEFT:
            if (aParam == iface.PARAM_MOUSE_CLOSE) {
                // zatvaranje klikom miša na opciju close
                self.minimizeWindow();
                rv = true;
            }
            break;
    }
    return rv;
};
```

Isječak 19: Izgled funkcije *windowWatcherCallback*

Sakrivanje prozora preglednika vrši se na način da se preglednik umjesto gašenja ili minimizacije nastavlja izvoditi kao pozadinski proces sakriven od korisnika. Funkcionalnost sakrivanja prozora implementirana je u *Minimize to Tray* programskoj knjižnici koja je JavaScriptu dostupna putem sučelja *trayWindowHider*. U isječku 20 prikazana je funkcija *minimizeWindow()* te primjer korištenja implementacije sučelja *trayWindowHider*. Funkciji *minimize()* komponente za sakrivanje predaju se parametri broja prozora koji se sakrivaju, niza prozora koji se sakrivaju, *null* za slučaj jednog prozora te ime funkcije koja će obraditi akcije nad preglednikom dok je sakriven.

```
window.extensions.mook.minimizetotray.minimizeWindow =
function() {
  this.m_windowHider =
    Components.classes[this.k_contractid_windowHider]
      .createInstance(Components.interfaces.trayIWindowHider);

  var baseWindows = [this.getBaseWindow(window)];
  this.m_isHidden = [window];

  this.m_windowHider.minimize(
    baseWindows.length,
    baseWindows,
    null,
    this.trayCallback);
};
```

Isječak 20: Izgled funkcije za sakrivanje prozora

Nakon sakrivanja prozora preglednika, preglednik se nastavlja izvoditi kao pozadinski proces. Pozadinski procesi sakriveni od korisnika obično se nalaze u prostoru za obavijesti (eng. *Notification Area*) gdje su prikazani korisniku u obliku ikone. Komponenta za sakrivanje prozora *trayWindowHider* osluškuje korisničke akcije nad ikonom, primjerice klik miša, te za svaku od njih poziva funkciju *trayCallback* prikazanu u isječku 21. Spomenuta funkcija *trayCallback* služi za obradu korisničke akcije nad ikonom. Ona implementira isto sučelje, i ima gotovo identičan način rada, kao funkcija za obradu korisničke akcije nad prozorom preglednika iz isječka 19.

Komponenta za sakrivanje prozora *windowHider* kreirana u isječku 20 nudi i metodu za njegovo ponovno prikazivanje. Funkcija *restore()*, koristeći prethodno kreiranu komponentu *windowHider*, ponovno prikazuje sakriveni prozor. Isječak 22 prikazuje funkciju *restore()*.

```

window.extensions.mook.minimizetotray.trayCallback =
function ( aEvent, aParam, aKeyMask, aMouseX, aMouseY ) {
    var self = window.extensions.mook.minimizetotray;
    var iface = Components.interfaces.trayIWindowCallback;
    var rv = false;
    switch (aEvent) {
        case iface.EVENT_MOUSEDBLCLK_LEFT:
            // dupli klik lijeve tipke miša
            self.restore();
            rv = true;
            break;
    }
    return rv;
};

```

Isječak 21: Izgled funkcije *trayCallback*

```

window.extensions.mook.minimizetotray.restore =
function ( ) {
    var wh = this.m_windowHider;
    this.m_windowHider = null;
    if (wh) {
        var windows = this.m_isHidden;
        for (var i = 0; i < windows.length; ++i) {
            var w = windows[i]; // prozor koji se ponovno prikazuje
        }
        wh.restore();
    }
}

```

Isječak 22: Izgled funkcije za ponovno prikazivanje sakrivenog prozora

Zamjensko rješenje

Osim predloženog maskiranja standardne funkcionalnosti preglednika kao što su zatvaranje ili minimizacija preglednika, *Minimize to Tray* programska knjižnica omogućava kombinaciju klasičnih opcija s vrstom korisnikove selekcije kao što su desni klik miša, dupli klik miša ili klik miša popraćen ctrl tipkom. U isječku 23 prikazan je alternativni switch-case blok naredbi za funkciju *windowsWatcherCallback* iz isječka 19 koji sakriva prozor preglednika ukoliko korisnik desnim klikom miša klikne na gumb za smanjivanje prozora.

```

switch (aEvent) {
    case iface.EVENT_MOUSEDOWN_RIGHT:
        if (aParam == iface.PARAM_MOUSE_MINIMIZE) {
            // klik desne tipke miša na opciju minimize
            self.minimizeWindow();
            rv = true;
        }
        break;
}

```

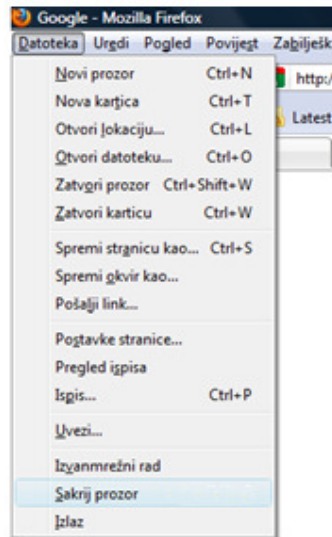
Isječak 23: Alternativni switch-case blok za funkciju *windowWatcherCallback*

5.3 Korisničko sučelje

Osim korištenja postojećih komponenti korisničkog sučelja, korisniku je moguće predstaviti novu funkcionalnost i sasvim novom komponentom korisničkog sučelja, primjerice kao opciju unutar datotečnog padajućeg izbornika (eng. *File*). Padajući izbornik u jeziku XUL definiran je oznakom *menupopup*, dok identifikacija *menu_FilePopup* kao atribut označava da se radi o datotečnom izborniku. Oznaka za pojedini član padajućeg izbornika definiran je kao *menuitem*. Pomoću atributa moguće je identificirati novostvorenu opciju, pridodati joj tekst gumba atributom *label* te slovo brzog biranja atributom *accesskey*. U konačnici, pomoću atributa *insertafter* ili *insertbefore* definira se na kojem će se mjestu u odnosu na postojeće elemente nalaziti dok se atributom *oncommand* pridružuje JavaScript funkcija sakrivanja prozora koja će se izvoditi kao posljedica odabira opcije. U isječku 24 prikazan je primjer dodavanja nove funkcionalnosti koja je vidljiva na slici 11.

```
<menupopup id="menu_FilePopup">
  <menuitem
    id="extensions.mook.minimizetotray.command.minimize"
    insertafter="goOfflineMenuitem"
    label="Sakrij prozor"
    accesskey="S"
    oncommand="window.extensions.mook.minimizetotray.minimizeWindow();"
  />
</menupopup>
```

Isječak 24: Dodavanje nove funkcionalnosti u datotečni padajući izbornik

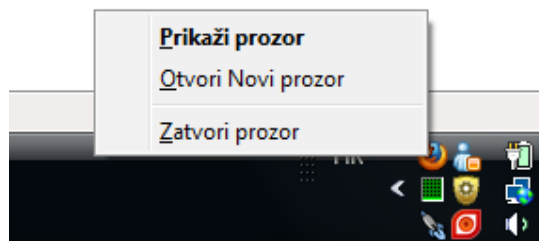


Slika 11: Izgled nove funkcionalnosti

Osim referenciranja na postojeće dijelove Firefoxa, koristeći XUL, moguće je stvoriti i novi nezavisan entitet unutar preglednika. Isječak 25 prikazuje novi iskočni izbornik (eng. *popup*) koji će se prikazati kao rezultat korisnikove interakcije s ikonom skrivenog prozora u traci za obavijesti. Kao ugniježdene sadrži elemente s oznakom *menuitem* koji mogu biti odvojeni pomoću elementom *menuSeparator*. Prvu opciju, opciju ponovnog prikazivanja prozora moguće je prikazati drugačije od ostalih korištenjem CSS deklaracija. CSS deklaracije zadaju se kao vrijednost atributa *style*. Kod akcije prikazivanja prozora koristi se funkcija *restore()* iz isječka 22, dok se za akcije otvaranja novog te gašenja trenutnog prozora koriste standardne Mozilline JavaScript funkcije *cmd_newNavigator* i *cmd_closeWindow*. Izgled iskočnog prozora u traci za obavijesti prikazan je u na slici 12. Novi iskočni izbornik, prikazan na slici 12, potrebno je prikazati unutar funkcije za obradu korisničkih akcija nad ikonom iz isječka 21.

```
<popupset id="mainPopupSet">
  <popup id="extensions.mook.minimizetotray.traypopup">
    <menuitem
      label="Prika&#382;i prozor"
      accesskey="P"
      oncommand="window.extensions.mook.minimizetotray.restore();"
      style="font-weight: 700;"
    />
    <menuitem
      label="Otvori Novi prozor"
      accesskey="O"
      command="cmd_newNavigator"
    />
    <menuseparator/>
    <menuitem
      label="Zatvori prozor"
      accesskey="Z"
      command="cmd_closeWindow"
    />
  </popup>
</popupset>
```

Isječak 25: Novi iskočni izbornik



Slika 12: Izgled novog iskočnog izbornika

6 Zaključak

Kako se web primjenski programi sve više šire na područja primjene na kojima su se uobičajeno koristili klasični oblici primjenskih programa, sve je veća zastupljenost web primjenskih programa koji se izvode bez stalnog korisničkog nadzora. Takvi primjenski programi sadrže programske funkcionalnosti koje ne zahtijevaju stalno međudjelovanje s korisnikom te funkcionalnosti pokretane vremenom ili događajima iz okoline. Gašenjem preglednika zaustavlja se izvođenje primjenskih programa, pa takvi web primjenski programi zahtijevaju neprekidan rad web preglednika.

U ovom završnom radu programski je ostvaren sustav koji nastavlja izvođenje primjenskih programa učitanih u web preglednik nakon njegova gašenja. Programski sustav ostvaren je u obliku proširenja (eng. *extension*) za preglednik Mozilla Firefox. Prilikom izgradnje proširenja korištena je programska knjižnica *Minimize to Tray* koja omogućava nastavljavanje izvođenja prozora preglednika kao pozadinskog procesa. Korištenje ostvarenog proširenja izravno bi moglo unaprijediti razinu kvalitete web primjenskih programa.

7 Literatura

- [1] Parrish, Rick: *XPCOM Part 1: An introduction to XPCOM*,
<http://www.ibm.com/developerworks/webservices/library/co-xpcom.html>, 1. veljače 2001.
- [2] Parrish, Rick: *XPCOM Part 2: XPCOM component basics*,
<http://www.ibm.com/developerworks/webservices/library/co-xpcom2.html>, 1. travnja 2001.
- [3] Parrish, Rick: *Intro to XPCOM, Part 4*,
<https://www.ibm.com/developerworks/webservices/library/co-xpcom4/index.html>, 1. svibnja 2001.
- [4] Mozilla Developer Center: *Building an Extension*,
https://developer.mozilla.org/en/Building_an_Extension, 2. siječnja 2010.
- [5] Mozilla Developer Center: *Firefox addons developer guide*,
https://developer.mozilla.org/En/Firefox_addons_developer_guide, 12. listopada 2009.
- [6] Mozilla Developer Center: *Setting up an extension development environment*,
https://developer.mozilla.org/en/Setting_up_extension_development_environment, 10. siječnja 2010.
- [7] Mozilla Developer Center: *Building an Extension*,
https://developer.mozilla.org/en/Building_an_Extension, 2. siječnja 2010.
- [8] Mozilla Wiki: *XUL:Home Page*, <https://wiki.mozilla.org/XUL>
- [9] Žagar, Mario: *Oznake*,
http://www.fer.hr/download/repository/OR_3_Oznake.pdf, 4. ožujka 2009.
- [10] Žagar, Mario: *XML Uvod*,
http://www.fer.hr/download/repository/OR_4_XML_Uvod.pdf, 18. ožujka 2009.

8 Sažetak

Primjenski programi za web sve se više šire na područja primjene na kojima su se uobičajeno koristili klasični oblici primjenskih programa. Tako je danas sve veća zastupljenost web primjenskih programa koji ne zahtijevaju stalnu interakciju s korisnikom, ili onih pokretanih vremenom ili događajima iz okoline.

Koristeći proširenje preglednika, programski je moguće ostvariti sustav koji nastavlja izvođenje takvih primjenskih programa i nakon gašenja preglednika. Proširenjem preglednika smatra se dio softvera koji proširuje funkcionalnost web preglednika.

Ključne riječi:

web primjenski programi, web preglednik, pozadinsko izvođenje, Mozilla Firefox, proširenja preglednika

9 Summary

Web applications are more and more taking the place of classic standalone applications in various areas of use. There is a significant increase of web applications which do not need continuous interaction with its users and event-driven web applications.

It is possible to create a browser extension which continues to run web applications after the browser is closed. A browser extension is a computer program which extends the functionality of a browser in some way.

Keywords:

web application, web browser, background process, Mozilla Firefox, browser extension