

Volum coordonat de

Sabin Buraga

Lenuța Alboaie • **Sînică Alboai**e • **Sergiu Dumitriu**

Marta Gîrdea • **Diana Gorea** • **Sergiu Tauciu**c

Tendințe actuale în proiectarea și dezvoltarea aplicațiilor Web



Web semantic • servicii Web • extensii Firefox • Wiki • foi de stiluri • AJAX • implementări

Volumul de lucrări ale celei de a V-a ediții
a *workshop*-ului <Web /> dedicat tehnologiilor Web

Iași, 26-27 noiembrie 2005

Sabin Buraga
(coordonator)

Lenuța Alboaiie • Sînică Alboaiie • Sergiu Dumitriu
Marta Gîrdea • Diana Gorea • Sergiu Tauciuc

Tendințe actuale în proiectarea și dezvoltarea aplicațiilor Web

Volumul de lucrări ale celei de a V-a ediții
a *workshop*-ului <Web /> dedicat tehnologiilor Web

Iași, 26-27 noiembrie 2005

Coperta: *Adrian Mironescu și Sabin-Corneliu Buraga*

Copyright © 2005 *Sabin-Corneliu Buraga*
<http://www.infoiasi.ro/~busaco/>

Ultima actualizare: 21 decembrie 2005

Permisunile de copiere, distribuire și/sau modificare ale acestui material se pot face conform termenilor stipulați de *GNU Free Documentation License*, versiunea 1.2.

Coordonatorul și autorii acestui volum nu își asumă nici o responsabilitate privind posibilele erori, omisiuni sau alte pagube ce pot rezulta din utilizarea informațiilor puse la dispoziție de acest document.

Toate numele de produse și servicii sunt mărci înregistrate ale proprietarilor respectivi.

Cuprins

Prefață	v
Capitolul 1. <i>Prin AJAX, spre Data Web și Semantic Web</i> (Sabin Buraga)	1
1. Preambul	1
2. De la Web 1.0 la Web 2.0 (<i>Data Web</i>)	2
2.1 Preliminarii. Caracteristici ale „noului” Web	
2.2 Marcaje (adnotări) definite de utilizator	
2.3 Participare, nu doar publicare a datelor	
2.4 Descentralizare radicală	
2.5 Încredere radicală. Folosirea standardelor	
2.6 Interacțiune bogată cu utilizatorul	
3. AJAX (<i>Asynchronous JavaScript And XML</i>)	9
3.1 Caracterizare	
3.2 Un prim exemplu: verificarea existenței unui nume de cont-utilizator	
3.3 Al doilea exemplu: jurnalizarea pe partea de server a excepțiilor apărute în programele JavaScript executate în cadrul <i>browser</i> -ului	
3.4 Aspecte importante în legătură cu realizarea de aplicații AJAX	
3.5 AJAX în contextul REST	
3.6 Utilizări și interfețe de programare AJAX	
4. În loc de concluzii	25
Capitolul 2. <i>Democrația pe Web: siturile Wiki</i> (Diana Gorea)	29
1. Conceptul de Wiki	29
2. Evoluție	30
2.1 Ideile care stau la baza noțiunii de Wiki	
2.2 Inteligența colectivă	
3. Principii de proiectare a siturilor Wiki	32
4. Web 2.0	33

5. Aplicații Wiki	34
5.1 MediaWiki	
5.2 TWiki	
5.3 XWiki	
5.4 Alte aplicații Wiki	
6. Concluzii	41
Capitolul 3. <i>Aplicații fără fir bazate pe Web-ul semantic</i> (Sabin Buraga)	43
1. Introducere	43
2. Prezentare succintă a Web-ului semantic	45
2.1 Preambul	
2.2 Niveluri ale Web-ului semantic	
3. Specificarea profilului unui dispozitiv <i>wireless</i>	47
3.1 Premise	
3.2 Specificarea caracteristicilor	
3.3 Avantaje	
4. Propunere de implementare	51
4.1 Arhitectură conceptuală	
4.2 Utilizări	
5. Concluzii	53
Capitolul 4. <i>Our Photos: descrierea, regăsirea și vizualizarea fotografiilor în cadrul unui album online partajat</i> (Sergiu Tauciuc)	55
1. Introducere	55
2. Suportul semantic pentru căutare	56
2.1 Fișierele de descriere RDF	
2.2 Fișierele de index XML	
3. Aspecte privind implementarea	61
4. Concluzii	70
Capitolul 5. <i>MUMSAI – un sistem de autentificare automată</i> (Lenuța Alboai, Sînică Alboai)	73
1. Introducere	73
2. Abordarea <i>MUMSAI</i>	73
3. Tehnologii folosite. Detalii de implementare	77
4. Testare	81
5. Concluzii	81

Capitolul 6. Dezvoltarea extensiilor pentru Firefox (Sergiu Dumitriu)	83
1. Introducere	83
1.1 about:mozilla	
1.2 <i>Firefox</i>	
2. Structura Firefox	85
2.1 <i>Mozilla Application Framework</i>	
2.2 <i>Chrome</i>	
2.3 Navigatorul <i>Firefox</i>	
2.4 <i>Plugin-uri</i>	
2.5 Fișierele <i>extensions.ini/ extensions.rdf</i>	
2.6 Extensiile efective	
3. Extensiile Firefox	90
3.1 Introducere	
3.2 Exemple	
3.3 Structura unei extensii	
3.4 Instalarea unei extensii	
4. Tehnologii	94
4.1 <i>XUL</i> – interfețe independente de platformă	
4.2 <i>JavaScript</i> – mic, dar puternic	
4.3 Localizare – aceeași aplicație, oricâte locații	
4.4 Stiluri, teme, aspecte	
5. Concluzii	103
Capitolul 7. <i>Valențe CSS</i> (Marta Gîrdea)	105
1. Introducere	105
2. Motivație	107
3. Impactul foilor de stiluri	109
3.1 Formă și conținut	
3.2 Static și dinamic	
3.3 Efecte speciale	
4. De la idee la punerea în practică	123
4.1 Cruda realitate	
4.2 Posibile soluții	
5. Concluzii	125

Prefață

„Fiecare este arhitectul propriului viitor.”

Appius Claudius

Incontestabil, prezentul științei calculatoarelor (*Computer Science*) este dominat de tehnologiile Internet, în general, și de cele Web, în special. Circumscris acestui domeniu, începând cu anul 2001 este organizat anual, la Iași, *workshop-ul* <Web /> dedicat variatei suite de tehnologii Web.

Inițiat și coordonat de Dr. *Sabin-Corneliu Buraga*, evenimentul are drept principale teme de interes atât proiectarea și dezvoltarea de aplicații Web sau dedicate spațiului WWW, cât și interacțiunea Web. Pe parcursul anilor, în cadrul <Web /> au fost dezbătute – de către specialiști ai mediului academic și industrial – subiecte referitoare la maniera bazată pe meta-lingvajul XML de organizare, stocare, descriere și regăsire a datelor, proiectarea și implementarea siturilor Web complexe, crearea de interfețe-utilizator ergonomice, exploatarea tehnologiilor Web avansate în diverse arii și circumstanțe etc.

Aflat la cea de a V-a ediție, *workshop-ul* a avut loc în perioada 26-27 noiembrie 2005, fiind organizat de *WebGroup* – grup de interes în domeniul tehnologiilor Web – și de *Asociația Studenților Informaticieni din Iași (ASII)*, activând în cadrul Facultății de Informatică a Universității „Alexandru Ioan Cuza” din Iași. Detalii privind programul științific și modul de desfășurare a evenimentului sunt disponibile la adresa <http://www.infoiasi.ro/~web/>.

Volumul de față cuprinde o parte dintre numeroasele contribuții expuse la ultima ediție a atelierului <Web />, reprezentând variante extinse ale prezentărilor avându-i drept autori pe *Lenuța Alboae, Sînică Alboae, Sabin-Corneliu Buraga, Sergiu Dumitriu, Marta Gârdea, Diana Gorea* și *Sergiu Tauciuc*, absolvenți sau cadre didactice ale Facultății de Informatică din Iași.

Această lucrare aduce în atenția cititorilor – în principal, specialiști sau viitori specialiști activând în domeniul Web-ului – o serie de aspecte privind Web-ul semantic, comunitățile virtuale, folosirea eficientă a stilurilor, implementările destinate dispozitivelor fără fir, extensiile programelor de navigare sau utilizarea serviciilor Web, surprinzând tendințele actuale de proiectare și dezvoltare a aplicațiilor Web.

Dr. *Sabin-Corneliu Buraga*
Iași, 17 decembrie 2005

Capitolul 1

PRIN AJAX, SPRE DATA WEB ȘI SEMANTIC WEB

Sabin-Corneliu Buraga

Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” din Iași
Str. G-ral. Berthelot, nr. 16 400783 Iași, România
busaco@infoiasi.ro – <http://www.infoiasi.ro/~busaco/>

Rezumat. Capitolul de față va realiza o prezentare generală a AJAX (*Asynchronous JavaScript And XML*), suită de tehnologii de transfer asincron de informații XML, în contextul evoluției spațiului World-Wide Web către așa-numitul stadiu de Web al datelor (*Data Web*), etapă premergătoare a Web-ului semantic. Se vor ilustra, de asemenea, o serie de exemple practice de implementare de *script*-uri AJAX pentru îmbunătățirea interacțiunii cu utilizatorul.

Cuvinte-cheie: asincronism, XML, interacțiune, Web semantic, proiectare Web, *scripting*.

1. PREAMBUL

Unul dintre cele mai importante și de succes servicii ale Internetului, **World-Wide Web**-ul – mai pe scurt **Web** sau spațiul **WWW** –, a fost instituit la CERN (*Centre Européen pour la Recherche Nucléaire* – Centrul European de Cercetări Nucleare de la Geneva) în anul 1989, grație viziunii lui Sir Tim Berners-Lee. Acesta, împreună cu Robert Caillau și o echipă de specialiști, a propus un sistem informatic distribuit, scopul principal urmărit fiind facilitarea accesului rapid la informațiile tehnice cuprinse în manualele de utilizare a calculatoarelor. În contextul apariției Web-ului, două mari direcții precursore trebuie menționate: dezvoltarea *hipertextului* (sau a procesării computerizate a documentelor electronice complexe) și dezvoltarea protoalelor Internet care au făcut posibilă comunicarea globală dintre rețelele de calculatoare. Data de 12 noiembrie 1990 este considerată ziua de naștere oficială a Web-ului.

Web-ul reprezintă un *sistem de distribuție locală sau globală a informațiilor hipermedia* (Berners-Lee, 1999). Din punct de vedere tehnic, spațiul Web pune la dispoziție un sistem global și standardizat de comunicare multimedia, informațiile fiind organizate asociativ și distribuite în funcție de cererile utilizatorilor, funcționând conform modelului client/server. Putem vedea Web-ul ca fiind un

spațiu informațional compus din elemente de interes, denumite *resurse*, desemnate de identificatori globali denumiți *URI (Uniform Resource Identifiers)*.

Scopurile originare principale ale spațiului WWW au fost acelea de a oferi o modalitate de comunicare inter-umană prin partajarea cunoștințelor și posibilitatea exploatării în manieră distribuită a puterii de calcul a computerelor conectate la Internet (Berners-Lee, 1989). Una dintre prioritățile Consorțiului Web (W3C) – organism care coordonează și contribuie la standardizarea tehnologiilor Web – este aceea de a pune la dispoziție o modalitate, bazată pe XML – *Extensible Markup Language* (Bray *et al.*, 2004), de prelucrare de către calculator a informațiilor. Soluția este prezentată în (Berners-Lee *et al.*, 2001) prin intermediul unui scenariu despre ceea ce ar trebui să fie cunoscut sub numele de *Web semantic* – a se vedea cele descrise în capitolul 3 al volumului de față.

Ideea de bază este aceea ca spațiul World-Wide Web să reprezinte o pânză consistentă de relații stabilite între obiecte identificabile (în prezent prin identificatori uniformi de resurse, iar ulterior prin nume stabilite de utilizatori), dezvoltarea viitoare a Web-ului fiind focalizată asupra mașinilor, nu numai asupra oamenilor (Buraga, 2004a).

În acest capitol, vom lua în considerație principalele caracteristici ale „noului” Web, denumit și *Data Web*, ca etapă preliminară și necesară constituirii Web-ului semantic. În cadrul sub-capitolului 2, vom trece în revistă schimbările majore aduse de noile aplicații Web, în ceea ce privește suportul pentru realizarea de marcaje (*tag-uri*) definite de utilizatori, maniera deschisă de participare la „stilul de viață” Web – ne referim în principal la *Weblog-uri* –, gradul ridicat de descentralizare a datelor și serviciilor existente, încrederea radicală și interacțiunea tot mai bogată și complexă cu utilizatorul. În acest context, următorul sub-capitol descrie suita de tehnologii AJAX (*Asynchronous JavaScript And XML*), însoțind expunerea cu diverse exemple concludente, dar și cu unele detalii privind maniera de proiectare și implementare a aplicațiilor alinate modelului AJAX. Tot în cadrul sub-capitolului 3 se realizează o comparație între AJAX și arhitectura REST.

Capitolul se încheie cu o serie de concluzii referitoare la evoluția viitoare a spațiului World-Wide Web, mai ales în ceea ce privește componenta socială a acestuia (*Social Web*).

2. DE LA WEB 1.0 LA WEB 2.0 (DATA WEB)

2.1 Preliminarii. Caracteristici ale „noului” Web

Actualmente, se constată o tranziție de la „vechiul” Web la „noul” Web, spațiul WWW fiind văzut mai mult ca o platformă software, în care utilizatorul își controlează propriile date, punându-le în mod uzual la dispoziție altora, prin intermediul unor instrumente colaborative.

Astfel, asistăm la proliferarea de *servicii specializate* disponibile grație tehnologiilor Web, atenția focalizându-se asupra lor și nu asupra pachetelor software, conglomerate de funcționalități, de cele mai multe ori puțin importante pentru marea majoritate a utilizatorilor.

De asemenea, Web-ul poate fi considerat un *mediu participativ*, cel mai pregnant aspect fiind al siturilor/aplicațiilor aliniate curentului „*social networking*” (Drummond *et al.*, 2004). Ne putem referi la comunități virtuale, *Weblog-uri*, situri de tip *Wiki*, aplicații de partajare a imaginilor, a notițelor, a „semnelor de carte” (adrese Web), situri permițând clasificarea și comunicarea globală prin intermediul așa-numitelor *folksonomies* (Mathes, 2004). Toate acestea au putut fi posibile datorită unor tehnologii standardizate bazate pe XML precum: RDF – *Resource Description Framework* (Manola & Miller, 2004), RSS – *Really Simple Syndication* (RSS) sau FOAF – *Friend Of A Friend* (Dodds, 2004; FOAF).

O altă caracteristică a Web-ului este cea a *scalabilității*, serviciile oferite putând fi invocate într-o manieră transparentă, de către oricine, în mod ubicuu. Asistăm la crearea și consolidarea unei arhitecturi orientate spre servicii (SOA – *Service-Oriented Architecture*), concept privind maniera de dezvoltare de aplicații distribuite slab conectate (*loose coupling*) care pot fi exploatate de utilizatori (umani sau nu) prin intermediul unor tehnologii deschise precum serviciile Web construite cu SOAP (*Simple Object Access Protocol*) ori REST (*REpresentational State Transfer*) – detalii în lucrările (He, 2003) și (Erl, 2005). De asemenea, putem considera că aplicațiile software pot fi rulate de oriunde, independente de platformă și de dispozitiv.

Grație unor tehnologii și aplicații Web deschise, datele pot fi transformate în orice format dorit de utilizator, însuși utilizatorul contribuind la editarea acestora, prin intermediul unor instrumente precum *blog-urile* sau *Wiki-urile* (Buraga, 2004b; Buraga, 2005). Mediatizarea schimbărilor survenite pe un sit se poate realiza ușor via conținuturi RSS ori Atom.

Nu putem să nu menționăm faptul că, în prezent, Web-ul încurajează constituirea așa-numitei *inteligențe colective* (O’Reilly, 2005). Prin stabilirea unei adrese Web pentru un fragment de cunoștințe conținând informații și/sau programe, și apoi prin partajarea acelei adrese cu alții, autorii pun în practică procesul democratic următor: orice persoană aparținând comunității Web poate arăta că anumite informații sunt interesante („bune”) prin crearea unei legături către resursa al cărei conținut include respectivele informații, pentru o eventuală utilizare ulterioară. Utilizatorii nu copiază documentul inițial, pentru că ar fi prea costisitor; cel mai ieftin și eficient mod de a accesa și propaga ideile conținute de respectivul document este prin intermediul adresei Web, realizându-se o legătură hipertext către acel fragment de informație. Aceasta alimentează ciclul selecției naturale în reprezentarea cunoștințelor: utilizarea determină comunitatea, care rafinează la rândul ei, permanent, ontologia comună (Buraga, 2004a). Succesul actualelor motoare de căutare, a marilor magazine virtuale și a e-comunităților reprezentate în principal de *blog-uri* și *Wiki-uri* se datorează tocmai acestui aspect.

2.2 Marcaje (adnotări) definite de utilizator

Ceea ce apare pregnant în cazul Web-ului datelor (*Data Web*) este faptul că utilizatorii pot folosi propriile lor marcaje (*tag-uri*) astfel încât să adnoteze informațiile pe care le pun sau au la dispoziție, specificând astfel *meta-date*.

În Web-ul „vechi”, *meta-datele* sunt create de profesioniști specializați pe un anumit domeniu, astfel apărând vocabulare privitoare la creațiile intelectuale – e.g., MARC (*MAchine Readable Cataloging*) sau OPAC (*Online Public Access Catalogs*), având drept precursori schemele de catalogare și clasificare folosite de biblioteci – de exemplu, sistemul zecimal Dewey.

O alternativă la acestea o reprezintă modalitatea ca însuși autorul să specifice *meta-datele* dorite, una dintre maniere fiind cea oferită de DCMI (*Dublin Core Metadata Initiative*), cu un anumit succes în unele arii de cunoaștere, mai ales în ceea ce privește descrierea resurselor (electronice sau nu) – detalii la (DCMI). Ca și în cazul precedent, mulțimea de marcatori permiși este una fixată, pentru orice noi construcții trebuind găsite alte vocabulare publice, de multe ori inexistente.

O a treia soluție este aceea de a permite utilizatorilor să-și definească proprii marcatori, într-o manieră explicită, pentru a putea fi ulterior folosiți de aplicații. O metodă – considerată deja „venerabilă” – este cea oferită de *Weblog-uri*, dar există și alte maniere precum marcarea conținutului (*tagging content*) via instrumente ca *Del.icio.us* ori *Flickr.com*, veritabile aplicații aliniate direcției „*social bookmarking tools*” (Hammond *et al.*, 2005).

Se instituie astfel așa-numita arhitectură a participării (*architecture of participation*), conform (O’Reilly, 2003), care pune la dispoziție aplicațiile necesare definirii de *tag-uri* proprii, într-o manieră nestructurată ori liber structurată de clasificare a conținutului resurselor Web. Utilizatorii sunt capabili să eticheteze, conform propriilor criterii, adrese de situri Web ori reprezentări de resurse, activitate referită prin termeni precum „*folksonomy*” (*folk + taxonomy*), „*folk classification*”, „*ethno-classification*”, „*distributed classification*” sau „*social classification*” (Hammond *et al.*, 2005; Mathes, 2004).

Printre aplicațiile „noului” Web se numără *Connotea*, *del.icio.us*, *Flickr.com*, *Frassle*, *Furl*, *Spurl.net* și *unalog*, cu implementări în limbaje ca Perl, Python, PHP ori Java – detalii în (Hammond *et al.*, 2005), (Lund *et al.*, 2005) și (Mathes, 2004).

2.3 Participare, nu doar publicare a datelor

În mod pregnant, aici o cvasi-supremație o dețin *Weblog-urile*. Un *Weblog* (sau *blog*, pe scurt) reprezintă o pagină Web care conține diverse fragmente de informație puse la dispoziție de către o persoană posibililor vizitatori (Buraga, 2005). Termenul provine de la cuvintele „*Web*” (WWW) și „*log*” (jurnal) și a fost introdus în premieră de Jorn Barger în decembrie 1997.

Un *blog* poate fi considerat un mijloc de comunicare inter-personală, natura sa putând varia de la o formă de jurnal *on-line* până la depozit de informații privitoare la un domeniu (tehnic sau nu). Un astfel de *blog* este redactat (întreținut) fie de o singură persoană, fie de un grup. Luând în considerație tematica abordată, un *blog* poate fi încadrat în următoarele categorii principale:

- personal,
- comunitar,
- oferind noutăți,
- specializat,
- colaborativ,
- politic,
- corporatist.

Se consideră că fenomenul *Weblog* a început să ia amploare începând cu anul 2000 (în era „vechiului” Web), unii analiști estimând că actualmente pe Web apar zilnic peste 80 de mii de noi *blog-uri*, întreținute atât de anonimi, cât și de diverse personalități provenind dintr-o sumedenie de domenii (nu neapărat cele vizând informatica). În ianuarie 2005, revista Fortune listează un număr de 8 *blogger-i* (persoane care își redactează propriile *blog-uri*) cărora oamenii de afaceri ar trebui să nu le ignore comentariile publicate pe Web. Din aceasta se poate deduce faptul că *blog-ul* reprezintă o componentă-cheie a „noului” Web.

După Greg Hard, „*blogging-ul* nu reprezintă doar un *blog*. Este o manieră de partajare la nivelul întregii lumi a opiniilor și gândurilor personale. Dacă punem la dispoziție informații de interes, atunci trebuie să partajăm hiper-legături și să facem disponibile comentariile din cadrul altor *blog-uri*. Legăturile hiper-text reprezintă punctul forte al Web-ului, iar în cazul *Weblogging-ului* ele sunt elementul cheie.” Din punctul de vedere al strategiei realizării afacerilor electronice sau a instituirii unei comunități *on-line*, „*blogging-ul* este una dintre cele mai rapide căi de actualizare a sitului propriu” (Eric Dolecki).

De asemenea, un alt context de utilizare a *blog-urilor* este cel reprezentat de intranetul întreprinderilor virtuale, un *blog* putând deveni un instrument indispensabil de inter-comunicare la nivel de echipe de lucru în cadrul unui proiect. Din această perspectivă, *blog-ul* poate fi o componentă importantă a aplicațiilor de tip *groupware* (*teamware*).

Conținutul unui *blog* nu trebuie să fie exclusiv textual, existând *blog-uri* care oferă prezentări multimedia (audio – de succes sunt MP3 *blog-urile*, fotografiile, video – denumirea acestei categorii fiind *videoblog* sau, pe scurt, *vlog*). Alte detalii pot fi consultate în lucrările (Buraga, 2004b), (Buraga, 2005) și (Doctorow *et al.*, 2002).

La succesul fenomenului *blogging* au contribuit:

- tehnologia RSS/Atom folosită pentru mediatizarea (*syndication*) conținutului siturilor Web; formatele RSS și Atom permit calculatoarelor să interschimbe informații privitoare la conținutul siturilor Web (în acest caz, al *blog*-urilor). Fiind bazat pe meta-limbajul XML, un document RSS/Atom este ușor de procesat sau de creat, existând diverse instrumente – numite *agregatori de conținut* – care extrag, sumarizează și stochează datele RSS/Atom;
- instituirea de relații între *blog*-uri grație legăturilor permanente (*permanent links*); astfel, un mesaj poate fi accesat fie direct, fie parcurgând lista mesajelor (numite și *post*-uri) existente;
- crearea așa-numitei *blogosfere*, mulțimea de *blog*-uri interconectate punând astfel bazele unei rețele sociale (*social network*) în stilul *peer-to-peer* pentru facilitarea realizării de discuții și interschimb de cunoștințe în manieră sincronă sau asincronă – a se vizita situri precum vezi *Friendster*, *LinkedIn*, *Orkut* etc.

Conform (O'Reilly, 2005), utilizatorii adaugă valoare resurselor Web („*users add value*”). Companiile alinate „noului” Web oferă sau vor trebui să ofere instrumente privind agregarea datelor-utilizator și de a crește valoarea acestora ca efect colateral utilizării aplicațiilor puse la dispoziție. Participarea utilizatorilor va reprezenta o parte fundamentală a arhitecturii aplicațiilor Web viitoare.

2.4 Descentralizare radicală

Web-ul datelor implică o descentralizare completă a datelor, serviciilor și aplicațiilor. Fiecare aplicație Web importantă a zilelor noastre se bazează pe o bază de date specializată – de exemplu, Google pe indexul resurselor găsite de roboții Web, Amazon pe baza de date privitoare la produsele disponibile, MapQuest pe baze de date referitoare la hărți geografice. În „vechiul” Web informațiile din cadrul acestor baze de date erau și sunt în continuare protejate de *copyright*.

Încurajând utilizatorii să participe cu propriile date (a se vedea și secțiunile 2.2 și 2.3), aplicațiile „noului” Web vor fi capabile să ofere noi moduri de (re-)utilizare și de (re-)agregare a informațiilor, eventual contribuind la îmbogățirea semantică a acestora.

Mai mult, un serviciu devine automat mai bun cu cât mai mulți oameni îl utilizează, principiu de bază al Web-ului 2.0, conform (O'Reilly, 2005). Un exemplu este cel al aplicației BitTorrent, în care fiecare utilizator își oferă resursele proprii în crearea unei arhitecturi Internet descentralizate de transfer *peer-to-peer* al fișierelor. Fișierele partajate sunt divizate în fragmente care pot fi preluate din diverse locații și asamblate la destinatar. Cu cât un fișier devine mai popular, cu atât va fi mai rapid transferat, din moment ce va fi deținut de un număr mai mare de utilizatori și, deci, va putea fi preluat în paralel din surse multiple.

2.5 Încredere radicală. Folosirea standardelor

Surprinzător sau poate nu, una dintre categoriile de aplicații ale Web-ului date-lor este cea a siturilor de tip *wiki* care permit utilizatorilor să adauge diverse tipuri de conținuturi, dând în plus posibilitatea vizitatorilor să editeze acel conținut. Termenul *wiki* se referă și la software-ul colaborativ utilizat pentru crearea acestor tipuri de situri (Buraga, 2005) – a se consulta, pentru detalii, capitolul 2.

Un exemplu important este *Wikipedia.com*, fiecare intrare a acestei enciclopedii deschise putând fi editată de orice utilizator de pe glob, reprezentând astfel un experiment interesant de încredere completă. Se aplică, astfel, dictonul lui Eric Raymond, în contextul original al scrierii de software *open source*: „având suficienți ochi, orice greșeală este relevată” („*with enough eyeballs, all bugs are shallow*”). Putem considera că Wikipedia constituie o schimbare profundă în ceea ce privește dinamica și calitatea creării de conținut.

Desigur, nu trebuie să ignorăm faptul că protecția proprietății intelectuale limitează re-utilizarea informațiilor și previne experimentările. Astfel, conform (O'Reilly, 2005), atunci când beneficiile provin din adoptări colective de informații și nu din restricții private, trebuie creat cadrul unei cât mai facile utilizări a acelor informații, prin urmarea standardelor (deschise) existente și prin recurgerea la licențe de utilizare cât mai puțin restrictive – „*some rights reserved*” în loc de „*all rights reserved*”.

Mai mult, aplicațiile trebuie sau vor trebui proiectate având în vedere interoperabilitatea, tratarea utilizatorilor de zi cu zi ca fiind co-dezvoltatori ai software-ului și adoptarea unor modele de programare ușoare (*lightweight*). Un exemplu este cel al folosirii unor conglomerate de tehnologii precum AJAX, RSS sau REST, în detrimentul unora mai sofisticate.

2.6 Interacțiune bogată cu utilizatorul

Poate mai mult decât în cazul interfețelor clasice, utilizatorii doresc flexibilitate, internaționalizare și calitate de la interfețele expuse de siturile/aplicațiile Web actuale. Spațiul WWW a fost conceput cu scopul declarat de a fi utilizat de oameni, deci interfețele Web trebuie să ofere aceleași facilități pe care le regăsim la interfețele tradiționale (Buraga, 2003a):

- *funcționalitatea* – se referă la ce anume o interfață realizează; acest scop este îndeplinit pe Web prin adoptarea unor tehnici de programare la *nivel de client* (navigator) – prin intermediul programelor *script* (i.e., JavaScript în contextul HTML-ului dinamic), a *applet*-urilor Java, a prezentărilor Flash sau a controalelor ActiveX – sau la *nivel de server* Web – folosindu-se *script*-uri CGI (*Common Gateway Interface*), concepute în diverse limbaje de programare (C, C++, Perl, Python etc.), servere de aplicații Web (JSP, PHP, Zope etc.), *servlet*-uri Java sau componente .NET – vezi (Buraga, 2001; Buraga, 2003b; Buraga, 2004b); o altă abordare este cea uti-

lizând agenți software sau componente intermediare – *middleware* (CORBA, DCOM, servicii Web); interfețele Web trebuie să fie modulare – elementele de interfață, disponibile local sau la distanță, pot fi „legate” în manieră dinamică, grație hipertextului;

- *utilizabilitatea* – această proprietate specifică modul *cum* o interfață satisface funcționalitatea sa; există o serie de factori, majoritatea calitativi, pentru determinarea acestui aspect: timpul de învățare, ușurința în utilizare, performanța în utilizare, ergonomia; interfețele Web moștenesc utilizabilitatea aplicațiilor de navigare pe Web și a documentelor (X)HTML care pot fi vizitate, dar este încă dificil de manipulat informațiile unui sit Web bogat în elemente multimedia (în special dacă include *applet*-uri Java, prezentări Flash sau componente ActiveX); cerințele ar fi ca uneltele de implementare (editoarele de cod HTML, generatoarele de *script*-uri etc.) să includă instrumente pentru măsurarea și optimizarea utilizabilității;
- *izolarea interfeței de aplicația propriu-zisă* – în cele mai multe cazuri, izolarea prezentării datelor de logica aplicației este aproape imposibilă pe Web: funcționalitatea (programele rulate pe partea de server sau de client) alternează cu interfața (elementele HTML), aceasta conducând la dificultăți majore în structurarea, implementarea și menținerea unui sit Web de dimensiuni medii sau mari (conținând peste 100 de documente); imperios necesar este să se poată reutiliza anumite fragmente de interfață pentru aplicații diferite, operându-se eventual doar modificări minore – actualmente, acest lucru este aproape imposibil, deși există un anumit suport pentru împlinirea acestui deziderat (*e.g.*, foile de stiluri CSS (*Cascading Style Sheets*) externe, directivele SSI (*Server-Side Includes*), componentele ASP.NET sau *JavaBeans*, folosirea unor servere de aplicații încurajând adoptarea șabloanelor de proiectare Web);
- *adaptabilitatea* – interfața trebuie să ofere suport pentru interacțiuni multimodale, acest aspect devenind foarte important în contextul interfețelor Web (Candell & Raggett, 2002), prin proliferarea diferitelor dispozitive mobile care pot prezenta mijloace diverse de intrare/ieșire; instrumentele de implementare ale interfețelor Web vor trebui să fie capabile să pună la dispoziție facilități de interacționare multimodală cu utilizatorii (via voce, gesturi, scris de mână etc.);
- *consistența* – în prezent este dificil să se asigure consistența unui sit Web (Buraga, 2005), în special în cazul adoptării unor tehnici multiple de proiectare și din cauza lipsei de instrumente adecvate care să instruiască designerii Web să utilizeze exclusiv elemente consistente (asigurându-se consistența conținutului, a structurilor navigaționale, a dispunerii spațiale etc.).

Interfețele „noului” Web trebuie să ia în calcul avantajele oferite de mediul WWW în ceea ce privește ubicuitatea, disponibilitatea distribuirii datelor via

hipertext, posibilitățile de căutare etc. De asemenea, vor trebui asigurate – așa cum am menționat mai sus – o interacțiune sofisticată, similară celei oferite de interfețele-utilizator convenționale, și o cât mai bună utilizabilitate.

Drept aplicații pionere care pun la dispoziție o interacțiune Web avansată cu utilizatorul pot fi menționate *Gmail*, *Google Maps* și *Writely*, iar următorii ani vor aduce cu siguranță în atenția publicului larg alte tipuri de aplicații aliniate noilor direcții de interacțiune.

Una dintre componentele-cheie ale „noului” Web în ceea ce privește interacțiunea cu utilizatorul este suita de tehnologii deschise cunoscută sub numele de AJAX (*Asynchronous JavaScript And XML*), prezentată pe larg în următorul subcapitol al volumului de față.

3. AJAX (ASYNCHRONOUS JAVASCRIPT AND XML)

3.1 Caracterizare

Termenul AJAX aparține lui Jesse James Garrett și reprezintă o suită de tehnologii deschise, încorporând (Eernisse, 2005):

- limbaje standardizate de prezentare a datelor, precum XHTML – *Extensible HTML* și CSS – *Cascading Style Sheets* (W3C);
- redare și interacțiune via standardul DOM – *Document Object Model* (Le Hors *et al.*, 2000; Buraga, 2001; W3C);
- interschimb și manipulare de date prin XML și/sau XSLT – *Extensible Stylesheet Language Transformations* (Clark, 1999);
- transfer asincron de date via obiectul *XMLHttpRequest* (McLelland, 2005);
- procesare prin limbajul ECMAScript/JavaScript (ECMA).

Componenta de bază este obiectul *XMLHttpRequest* pus la dispoziție de către *browser*-ul Web. Acest obiect permite realizarea de cereri HTTP (*e.g.*, GET și POST) dintr-un program rulând la nivel de client (*browser*) spre o aplicație de pe server, într-un mod *asincron*. Astfel, conținutul paginilor Web nu mai trebuie reîncărcat în întregime. În mod uzual, datele vehiculate între programele client și server sunt marcate în XML. Există cazuri în care se poate folosi o altă convenție de marcare a datelor – un exemplu este prezentat în (Eernisse, 2005).

Modul de implementare a obiectului *XMLHttpRequest* depinde de navigator: Mozilla având versiunea mai mare sau egală cu 1.0 și toate versiunile de Firefox îl oferă ca obiect nativ, iar în Internet Explorer 5.0 sau ulterior poate fi instanțiat drept obiect ActiveX. De asemenea, navigatorul Safari disponibil pe platformele Mac OS îl implementează începând cu versiunea 1.2. La fel, cele mai recente versiuni ale programului Opera oferă suport pentru *XMLHttpRequest*.

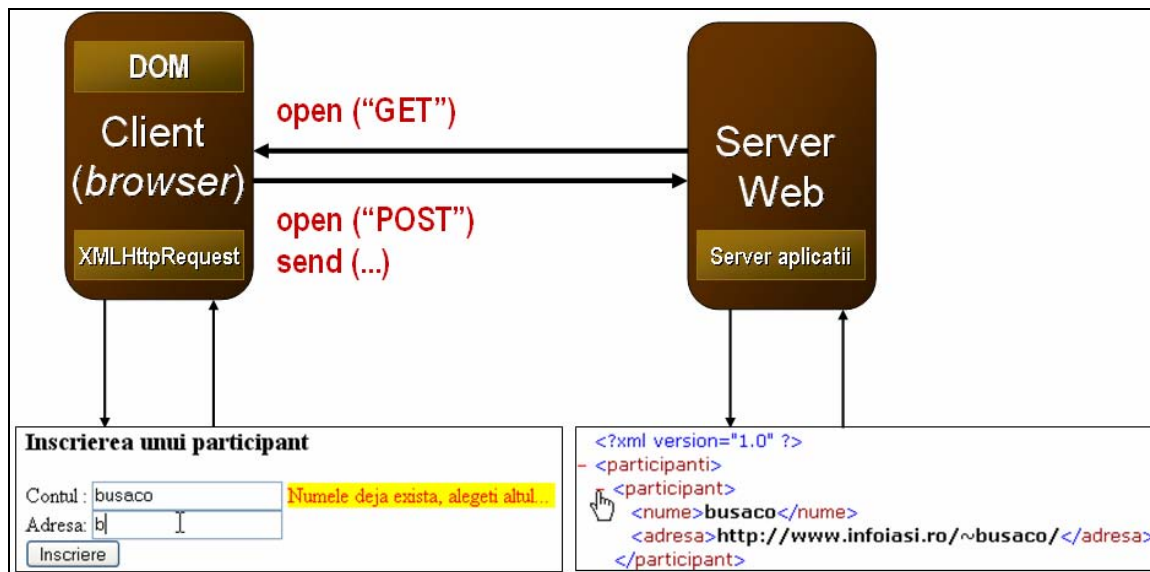


Figura 1. Transferul datelor XML între client și server (fereastra navigatorului, în stânga, și documentul XML stocat la nivel de server, în dreapta)

Dialogul dintre aplicațiile client și server este ilustrat în figura 1.

Principalele metode care vor fi folosite efectiv de un dezvoltator de aplicații AJAX sunt următoarele:

- *open()* – inițiază (deschide) o conexiune HTTP cu serverul, trimițând o cerere (uzual, GET sau POST) către acesta; desigur, aplicația de pe server cu care va avea loc schimbul de date va fi desemnată de un URI;
- *send()* – transmite date, în manieră asincronă, către aplicația rulând pe server;
- *abort()* – abandonează transferul curent;
- *setRequestHeader()* – specifică diverse câmpuri ale antetului HTTP (de exemplu, se poate stabili o anumită valoare pentru câmpul *Content-Type*).

Proprietățile de bază ale obiectului sunt detaliate în continuare:

- *readyState* – conține codul de stare a transferului (e.g., valoarea 4 specifică faptul că transferul datelor s-a efectuat complet);
- *status* – reprezintă codul de stare HTTP întors de serverul Web; de exemplu, 200 (*Ok*), 404 (*Not Found*), 500 (*Server Error*);
- *statusText* – desemnează șirul de caracter explicând în limbaj natural codul de stare obținut;
- *onreadystatechange* – desemnează funcția care va fi invocată la modificările privind schimbarea stării de transfer a datelor (reamintim faptul că datele sunt transmise în manieră asincronă și deci trebuie să fim notificați asupra schimbării evenimentelor ce pot surveni pe parcursul transferului).

3.2 Un prim exemplu: verificarea existenței unui nume de cont-utilizator

În cadrul acestui prim exemplu, vom considera următorul scenariu: un vizitator al sitului dedicat *workshop*-ului <Web /> dorește să se înscrie ca participant, pentru aceasta trebuind să completeze un formular care îi va solicita un nume de cont și o adresă de *e-mail*. Numele de cont trebuie să fie unic, desemnând fără ambiguități acel utilizator.

O primă soluție ar fi să verificăm pe server existența numelui de cont, prin consultarea unei baze de date și generarea unui document XHTML nou care va fi transmis clientului în cazul în care numele de cont este deja ales de altcineva ori atunci când dorim să semnalăm faptul că informațiile au fost stocate la nivel de server. Aceasta conduce la reîncărcarea întregului conținut al paginii Web, prin efectuarea unui transfer suplimentar. De asemenea, utilizatorul nu va putea cunoaște dacă numele de cont ales este incorect decât după completarea tuturor câmpurilor formularului Web și acționarea butonului de tip *submit*.

A doua soluție, firesc, va face apel la maniera AJAX de transfer asincron al datelor și este inspirată de (McLellan, 2005). La apariția evenimentului de pierdere a *focus*-ului asupra câmpului de introducere a numelui de cont, prin intermediul unui program JavaScript se va realiza un transfer asincron către aplicația rulând pe server, aplicație care va întoarce răspunsul privind existența numelui de cont. În acest caz, utilizatorul va primi imediat *feedback* pozitiv de la aplicație, trebuind să specifice un alt nume. Desigur, verificarea existenței contului respectiv va trebui realizată la nivel de server și după apăsarea butonului *submit*, deoarece navigatorul ar putea să nu aibă implementat obiectul *XMLHttpRequest* sau vizitatorul are dezactivat suportul pentru JavaScript ori nu introduce un nume de cont valid (inexistent).

Formularul electronic de interacțiune cu utilizatorul ar putea avea structura următoare:

```
<form action="adauga.php" method="get">
  Contul : <input type="text" name="nume"
    title="Introduceți numele de cont dorit"
    onblur="javascript:verificaNume(this.value, '')" />
  <span class="ascuns" id="eroareNume">
    Numele deja exista, alegeti altul...
  </span>
  <br />
  Adresa: <input type="text" name="adresa"
    title="Introduceți adresa d-voastra (e-mail sau URI)" />
  <br />
  <input type="submit" value="Inscriere"
    title="Apasati aici pentru a va inscrie" />
</form>
```

Se remarcă faptul că apare un element ** care va conține mesajul de eroare afișat utilizatorului în cazul în care numele de cont deja există memorat pe server. Inițial, acest element va fi ascuns și va fi afișat prin modificarea proprietății de stil *display* oferită de CSS.

Clasele de proprietăți CSS sunt specificate în antetul documentului XHTML astfel:

```
<style type="text/css">
input[type="text"]:focus {
  background: #EEE;
  border-color: black
}
.eroare { /* stiluri pentru mesajul de eroare */
  display: inline;
  color: red;
  background-color: yellow;
}
.ascuns { /* stiluri pentru ascunderea mesajului */
  display: none;
}
</style>
```

La apariția evenimentului *onblur* va fi apelată funcția JavaScript *verificaNume()* care va avea în responsabilitate efectuarea transferului asincron către o aplicație PHP rulând pe server ce va apela o metodă de verificare a existenței numelui de cont. Atât cererea, cât și rezultatul obținut vor fi marcate în XML.

Codul-sursă al funcției *verificaNume()* este următorul:

```
// functia de verificare a existentei unui nume de persoana
function verificaNume (nume, raspuns) {
  // avem un raspuns?
  if (raspuns != '') {
    // preluam via metodele DOM elementul cu id='eroareNume'
    // pentru a afisa mesajul de eroare
    mesaj = document.getElementById ('eroareNume');
    // schimbam stilul de afisare (in caz de eroare vor fi aplicate
    // proprietatile de stil din clasa 'eroare',
    // altfel textul va fi ascuns
    mesaj.className = raspuns == 1 ? 'eroare' : 'ascuns';
  } else {
    // nu e nici un raspuns setat, vom verifica
    // trimittind o cerere asincrona catre server
    incarcaXML ('http://www.sit.org/verifica.php?nume=' + nume);
  }
}
```

Dacă răspuns a fost returnat, prin intermediul DOM vom prelua conținutul elementului *<eroareNume>* (al documentului XML transmis de aplicația de pe server) care va conduce, eventual, la modificarea proprietăților de stil asociate elementului ** descris mai sus.

Cererea asincronă către programul PHP rulând pe server va fi realizată de funcția *incarcaXML()* al cărei cod este dat mai jos:

```
var cerere; // cererea catre serverul Web

// incarca un document XML desemnat de 'url'
function incarcaXML (url) {
  // verificam existenta obiectului XMLHttpRequest
  if (window.XMLHttpRequest) {
    // exista suport nativ (Mozilla, Firefox)
    cerere = new XMLHttpRequest ();
```

```

    // stabileste functia de tratare a starii incarcarii
    cerere.onreadystatechange = trateazaCererea;
    // preluam documentul prin metoda GET
    cerere.open ("GET", url, true);
    cerere.send (null);
}
else if (window.ActiveXObject) {
    cerere = new ActiveXObject ("Microsoft.XMLHTTP");
    if (cerere) { // se poate folosi obiectul ActiveX in MSIE
        // preluam documentul prin metoda GET
        cerere.onreadystatechange = trateazaCererea;
        cerere.open ("GET", url, true);
        cerere.send ();
    }
} // final de if
}

```

După cum se poate remarca, se verifică disponibilitatea obiectului *XMLHttpRequest* în cadrul navigatorului Web. Modul de transfer se va face prin metoda GET, iar tratarea evenimentelor de transmisie asincronă a datelor are loc în funcția *trateazaCererea()*:

```

// functia de tratare a schimbarii de stare a cererii
function trateazaCererea () {
    // verificam daca incarcarea s-a terminat cu succes
    if (cerere.readyState == 4) {
        // verificam daca am obtinut codul de stare '200 Ok'
        if (cerere.status == 200) {
            // procesam datele receptionate
            // (preluam elementul radacina al documentului XML)
            raspuns = cerere.responseXML.documentElement;
            metoda =
                raspuns.getElementsByTagName('metoda')[0].firstChild.data;
            rezultat =
                raspuns.getElementsByTagName('rezultat')[0].firstChild.data;
            // evaluam metoda
            eval ( metoda + '(\'\', rezultat)');
        }
        // eventual, se pot trata si alte coduri de stare (404, 500 etc.)
        else { // aceasta alertare nu foloseste prea mult...
            alert ("A aparut o problema la transferul datelor XML:\n" +
                cerere.statusText);
        }
    } // final de if
}

```

Datele XML care vor fi recepționate reprezintă un nume de metodă – în cazul nostru, *verificaNume()* – și rezultatul întors de aceasta. Desigur, aceste date vor fi preluate doar în caz de succes (codul de stare va fi 200).

Pe partea de server, va trebui scris un program PHP, numit *verifica.php*, care va testa existența numelui de cont-utilizator. Vom stoca aceste informații într-un document XML cu structura:

```

<?xml version="1.0"?>
<participanti>
  <participant>
    <nume>busaco</nume>
    <adresa>http://www.infoiasi.ro/~busaco/</adresa>
  </participant>

```

```

<participant>
  <nume>laur</nume>
  <adresa>laur@grapefruit.ro</adresa>
</participant>
</participanti>

```

Codul-sursă al programului este furnizat în continuare:

```

<?php
/* Program PHP care verifica existenta unui nume de cont
   prin parcurgerea unui document XML memorind informatii
   despre participantii la un eveniment (se foloseste modelul DOM)
*/

// locul unde e stocat documentul XML
define ("PATH", '');
# De exemplu, pentru Windows cu Apache2Triad instalat in 'apache2':
# define ('PATH', 'c:\\apache2\\htdocs\\php-a\\');
define ('DOCXML', 'participanti.xml');

// trimitem tipul continutului
header ('Content-type: text/xml');

// functie care verifica daca un nume exista deja
// returneaza 1 daca numele exista, 0 in caz contrar
function verifica ($un_nume) {
  // incarcam fisierul XML cu datele despre participantii
  if (!$dom = domxml_open_file (PATH . DOCXML)) {
    return 0;
  }
  $radacina = $dom->document_element();
  $participanti = $radacina->get_elements_by_tagname('participant');
  // parcurgem toti participantii gasiti...
  foreach ($participanti as $participant) {
    $nume = $participant->get_elements_by_tagname('nume');
    // comparam, ignorind minusculele de majuscule
    if (!strcasecmp($un_nume, $nume[0]->get_content())) {
      return 1;
    }
  }
  return 0;
}

// generam continutul XML
echo '<?xml version="1.0" ?>';
?>
<raspuns>
  <metoda>verificaNume</metoda>
  <rezultat><?php echo verifica ($_REQUEST['nume']); ?></rezultat>
</raspuns>

```

Se va returna un conținut XML, așteptat pe partea de client de către programul JavaScript. Folosind modelul DOM (Le Hors *et al.*, 2000), se verifică existența numelui de cont furnizat *script*-ului PHP de către funcția JavaScript care l-a apelat în manieră asincronă.

Desigur, mai trebuie scris programul PHP care realizează adăugarea informațiilor solicitate, apelat prin metoda GET la acționarea butonului *submit* din cadrul formularului Web. O posibilă soluție este următoarea:

```

<?php
/* Program PHP care insereaza informatii privitoare la un participant
   intr-un document XML dat (se foloseste modelul DOM)
*/

// locul unde e stocat documentul XML
define ("PATH", '');
# De exemplu, pentru Windows cu Apache2Triad instalat in 'apache2':
# define ('PATH', 'c:\\apache2\\htdocs\\php-a\\');
define ('DOCXML', 'participanti.xml');

// preluam valorile parametrilor
$nume = $_REQUEST['nume'];
$adresa = $_REQUEST['adresa'];

if (!isset ($nume) || !isset ($adresa) || !$nume || !$adresa) {
    echo "<p>Trebuie specificati parametrii de intrare 'nume' si
        'adresa'.</p>";
    exit;
}

// incarcam fisierul XML cu date despre participanti
if (!(($dom = domxml_open_file (PATH . DOCXML))) {
    echo '<p>Eroare la deschiderea documentului XML.</p>';
    exit;
}
// preluam radacina documentului
$radacina = $dom->document_element();

// cream un element <participant>
$participant = $dom->create_element ('participant');
// cream un element <nume>
$nume_participant = $dom->create_element ('nume');
// cream un nod text, cu valoarea numelui
$valoare_nume_participant = $dom->create_text_node ($nume);
// ...si-l inseram la <nume>
$valoare_nume_participant =
    $nume_participant->append_child ($valoare_nume_participant);
// adaugam <nume> la <participant>
$nume_participant = $participant->append_child ($nume_participant);
// cream un element <adresa>
$adresa_participant = $dom->create_element ('adresa');
// cream un nod text, cu valoarea adresei
$valoare_adresa_participant = $dom->create_text_node ($adresa);
// ...si-l inseram la <adresa>
$valoare_adresa_participant =
    $adresa_participant->append_child ($valoare_adresa_participant);
// adaugam <adresa> la <participant>
$adresa_participant =
    $participant->append_child ($adresa_participant);
// la final, adaugam elementul <participant> la arbore
$radacina = $radacina->append_child ($participant);

// salvam arborele ca fisier XML
$dom->dump_file (PATH . DOCXML);

echo "<p>Au fost adaugate urmatoarele date:
    '$nume' si '$adresa'.</p>";
?>

```

Figura 2 reprezintă o captură-ecran oferind mesajul de eroare obținut de utilizator în cazul în care introduce un nume de cont deja existent.

Figura 2. Mesajul obținut în urma introducerii unui nume de cont existent

Apelând *script*-ul PHP în mod explicit, obținem ceea ce era de așteptat: un document XML ilustrat de figura 3.

Figura 3. Documentul XML returnat de programul PHP de verificare a existenței unui nume de cont

3.3 Al doilea exemplu: jurnalizarea pe partea de server a excepțiilor apărute în programele JavaScript executate în cadrul *browser*-ului

Ca și alte limbaje de programare moderne, JavaScript oferă dezvoltatorilor posibilitatea de a capta și trata excepțiile survenite în cadrul programelor. Cea mai uzuală cale este cea de recurgere la construcțiile *try ... catch* (ECMA). Singurul impediment este acela că mesajele de eroare și informațiile privind mediul de execuție (*e.g.*, versiunea navigatorului, platforma pe care rulează, componentele instalate etc.) sunt disponibile la nivelul clientului (vizitatorului) și nu la cel al serverului (personalul de programare și de testare a aplicațiilor) Web. Astfel, posibilele situații de excepție și/sau de eroare nu sunt semnalate în mod corespunzător celor abilitați să le rezolve.

Prin intermediul AJAX, această problemă poate fi rezolvată. Pentru început, vom scrie un program JavaScript care va încapsula într-o clasă *Jurnal* întreg procesul de jurnalizare a excepțiilor apărute. Spre deosebire de exemplul precedent, în acest caz prin metoda POST vom expedia aplicației de pe server informațiile privitoare la mesajele de eroare, informații ce vor fi stocate într-un fișier-jurnal. Vehicularea datelor se va produce în mod uzual într-un singur sens, de la programul-client JavaScript către cel invocat pe server (scris, după cum vom vedea, în două variante – una în Perl și cealaltă în PHP).

Clasa *Jurnal* este definită în *script*-ul *jurnal.js* (ea va putea fi folosită în orice program JavaScript căruia dorim să-i jurnalizăm mesajele de excepție survenite la rulare):

```
// definim o clasa folosita pentru jurnalizarea erorilor survenite
// in programele JavaScript
function Jurnal() {
    // date-membru
    this.cerere = null; // obiectul ce va face cererea POST spre server
    // metode
    this.serializeaza = serializeaza; // serializeaza eroarea
    this.jurnalizeaza = jurnalizeaza; // jurnalizeaza eroarea
}

// metoda de stocare a datelor despre o eroare intr-un document XML
// (serializeaza un obiect Error in XML)
function serializeaza (eroare) {
    var xml = '<eroare><nume>' + eroare.name + '</nume>\n' +
        '<mesaj>' + eroare.message + '</mesaj>\n';
    if (eroare.location) { // exista setata localizarea erorii
        xml += '<loc>' + eroare.location + '</loc>\n';
    }
    if (eroare.navigator) { // exista informatii despre navigator
        xml += '<client>' + eroare.navigator + '</client>\n';
    }
    xml += '</eroare>';
    return xml;
}

// metoda de jurnalizare
// (trimite spre serverul Web eroarea serializata ca document XML)
function jurnalizeaza (eroare) {
    // pregatim cererea
    if (window.XMLHttpRequest) {
        this.cerere = new XMLHttpRequest ();
        this.cerere.overrideMimeType ('text/xml');
    }
    else if (window.ActiveXObject) {
        this.cerere = new ActiveXObject ("Microsoft.XMLHTTP");
    } else {
        return; // nu e suportat AJAX
    }

    // stabilim metoda de transfer...
    // si URI-ul programului care va prelua datele transmise spre server
    this.cerere.open ('POST', 'jurnal.pl.cgi', true);
    // stabilim cimpurile-antet HTTP
    this.cerere.setRequestHeader ('Referer', location.href);
    // trimitem un continut XML
    this.cerere.setRequestHeader ('Content-type', 'text/xml');
    // stabilim functia de tratare a schimbarii starii
    this.cerere.onreadystatechange = trateazaJurnal;
    // transmitem datele XML catre scriptul de pe server
    this.cerere.send (this.serializeaza (eroare));
    // stabilim un timp de asteptare a transferului
    // (daca cererea nu e realizata in maxim 10 secunde, abandonam)
    this.timeout = window.setTimeout ('anuleazaJurnalizare()', 10000);
}

// instantiem obiectul de jurnalizare
var jurnal = new Jurnal();
```

```

// functie de abandonare a jurnalizarii
function anuleazaJurnalizare() {
    jurnal.cerere.abort ();
    alert ('Jurnalizarea a esuat.');
```

```

}

// functie de tratare a schimbarii de stare
function trateazaJurnal () {
    if (jurnal.cerere.readyState != 4) {
        return; // transferul inca are loc
    }

    // daca transferul s-a efectuat, stergem timeout-ul
    window.clearTimeout (jurnal.timeout);
    // transferul s-a efectuat complet (cerere finalizata)
    if (jurnal.cerere.status >= 400) { // semnalam erorile...
        alert ('A intervenit o eroare la jurnalizare: ' +
            jurnal.cerere.status);
    }
}

```

Vom transmite către aplicația de pe server un conținut XML și vom specifica și adresa de la care provine cererea POST prin intermediul câmpului-antet *Referer*. Obiectul încapsulând informațiile referitoare la eroare – codul și mesajul de eroare, locația, *browser*-ul etc. – va fi serializat via metoda *serializeaza()* ca document XML ce va fi prelucrat la nivelul serverului. Funcția *jurnalizeaza()* va efectua transferul efectiv al datelor XML spre programul de pe server. Dacă transmisia nu s-a realizat cu succes, vom semnala acest aspect utilizatorului – a se vedea metoda *trateazaJurnal()* –, iar dacă transferul nu a avut loc în maximum 10 secunde, atunci el va fi abandonat – consultați funcția *anuleazaJurnalizare()*.

Pe server, va fi invocat un *script* CGI conceput în limbajul Perl. Grație modului *CGI::Carp*, datele recepționate de la client vor fi elegant incluse în fișierul standard de jurnalizare al serverului Web – în cazul nostru, în fișierul *error.log* generat de Apache. Un fragment din acest fișier, privind erorile semnalate via AJAX, este redat în continuare (se observă mesajele diferite obținute în urma execuției unui cod JavaScript rulat pe două navigatoare):

```

[Fri Dec 16 15:41:29 2005] [error] [client 193.230.10.1]
[Fri Dec 16 15:41:29 2005] jurnal.pl.cgi: Eroare client:
ReferenceError, mesaj: codNecunoscut is not defined, locatie:
http://localhost/php-a/test_jurnal.html, functia: incorecta(),
client: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8)
Gecko/20051111 Firefox/1.5 at C:/apache2/htdocs/php-a/jurnal.pl.cgi
line 32, referer: http://localhost/php-a/test_jurnal.html

```

```

[Fri Dec 16 15:42:54 2005] [error] [client 193.231.33.1]
[Fri Dec 16 15:42:54 2005] jurnal.pl.cgi: Eroare client:
TypeError, mesaj: Object expected, locatie:
http://localhost/php-a/test_jurnal.html, functia: incorecta(),
client: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET
CLR 1.1.4322; .NET CLR 2.0.50727) at
C:/apache2/htdocs/php-a/jurnal.pl.cgi line 32, referer:
http://localhost/php-a/test_jurnal.html

```

Programul Perl are următorul *listing*:

```
#!/usr/bin/perl
# Pentru Windows: c:\apache2\perl\bin\perl

# Script Perl care primeste prin POST un continut XML referitor
# la erorile survenite pe partea client (clientul foloseste AJAX
# pentru a trimite datele spre scriere in fisierele-jurnal
# ale serverului Web)
use CGI::Carp;
use XML::Simple;

# preluam datele XML de la client
read (STDIN, $date, $ENV{'CONTENT_LENGTH'});

my $metoda = $ENV{'REQUEST_METHOD'};

print "Content-type: text/xml\n\n";

if ($metoda eq 'POST') { # am receptionat o cerere via POST
  eval {
    my $continut = $ENV{'CONTENT_TYPE'};
    if ($continut ne 'text/xml') { # nu e continutul asteptat
      print "Status: 415 Unsupported Media Type\n\n";
    }
    # procesam datele XML primite de la client
    my $doc = XML::Simple::XMLin ($date);
    # preluam numele, mesajul si locatia erorii raportate
    my ($nume, $mesaj, $loc, $client) =
      ($doc->{'nume'}, $doc->{'mesaj'}, $doc->{'loc'},
       $doc->{'client'});
    # ...si includem in fisierul de jurnalizare al serverului Web
    carp "Eroare client: $nume, mesaj: $mesaj,
        locatie: $loc, client: $client";
  };
  # verificam daca au aparut erori la nivel de server
  if ($?) {
    print "Status: 500 Internal server error\n\n";
    croak "Eroare in timpul jurnalizarii: $?";
  } else {
    # succes, trimitem codul de stare 204,
    # semnaland clientului sa nu astepte nici un continut
    print "Status: 204 No content\n\n";
  }
} else {
  # a fost folosita o metoda diferita de POST...
  print "Status: 405 Method not supported\n\n";
  croak "Metoda incorecta: $metoda";
}
```

Se remarcă utilizarea modului *XML::Simple* pentru procesarea facilă a conținutului XML recepționat de la client – mai multe detalii în lucrarea (Buraga *et al.*, 2002). De asemenea, se verifică dacă datele au fost recepționate prin metoda POST și dacă aparțin tipului MIME *text/xml*. În caz de eroare, se semnalează programului de la client codurile de stare corespunzătoare: 405, respectiv 415. Mesajele de eroare transmise sunt scrise în fișierul de jurnalizare al serverului Web prin intermediul metodelor *carp()* și *croak()* oferite de modulul *CGI::Carp*.

Programul nu va returna nici un conținut *script*-ului JavaScript care l-a invocat, aspect semnalat via codul de stare 204.

O altă rezolvare complementară este următoarea, apelând de data aceasta la o implementare PHP. În acest caz, informațiile trimise de client sunt stocate într-un fișier de jurnalizare propriu, numit *jurnal.log* și generat de programul PHP. În mod obișnuit, un *script* PHP nu are acces la fișierele-jurnal ale serverului Web, deci a fost adoptată o metodă mai simplă. Codul-sursă al programului este furnizat în continuare:

```
<?php
/* Script PHP care primește prin POST un conținut XML referitor
   la erorile survenite pe partea client (clientul folosește AJAX
   pentru a trimite datele spre scriere într-un fișier-jurnal)
*/

// locul unde e stocat documentul XML
define ("PATH", '');
# De exemplu, pentru Windows cu Apache2Triad instalat în 'apache2':
# define ('PATH', 'c:\\apache2\\htdocs\\php-a\\');
define ('JURNAL', 'jurnal.log');

$metoda = $_SERVER['REQUEST_METHOD']; // determinăm metoda HTTP

echo "Content-type: text/xml\n\n"; // vom genera conținut XML

// am recepționat o cerere via POST
if (!strcmp($metoda, 'POST')) {
    $continut = $_SERVER['CONTENT_TYPE'];
    if (strcmp($continut, 'text/xml')) {
        // nu e conținutul așteptat
        echo "Status: 415 Unsupported Media Type\n\n";
        exit;
    }
    // deschidem fișierul de jurnalizare
    $fis = @fopen (PATH . JURNAL, 'a');
    if (!$fis) {
        // eroare de deschidere a fișierului
        echo "Status: 500 Internal server error\n\n";
        exit;
    }

    // preluăm datele transmise de client via POST
    $date = $HTTP_RAW_POST_DATA;
    // procesăm prin DOM informațiile
    if (!(($dom = domxml_open_mem($date))) {
        // eroare de procesare
        echo "Status: 500 Internal server error\n\n";
        exit;
    }
    $radacina = $dom->document_element ();
    $nume = $radacina->get_elements_by_tagname ('nume');
    $mesaj = $radacina->get_elements_by_tagname ('mesaj');
    // scriem în fișier informațiile despre eroare
    @fwrite ($fis, 'Nume: ' . $nume[0]->get_content() . ', mesaj: ' .
        $mesaj[0]->get_content() . "\n");
    fclose ($fis);
    // gata cu succes (trimitem codul de stare 204,
    // semnalînd clientului să nu aștepte nici un conținut
```

```

    print "Status: 204 No content\n\n";
} else {
    // a fost folosita o metoda diferita de POST...
    echo "Status: 405 Method not supported\n\n";
}
?>

```

Preluarea datelor transmise de client prin metoda POST se realizează via variabila predefinită `$HTTP_RAW_POST_DATA`. Aceste date sunt de fapt marcate în XML și grație metodei `get_elements_by_tagname()` oferită de DOM sunt inserate în fișierul-jurnal. Ca și în situația anterioară, posibilele erori sunt semnalate prin intermediul codurilor de stare HTTP.

La finalul acestei secțiuni, furnizăm codul-sursă al *script*-ului JavaScript care testează maniera de stocare pe server a excepțiilor survenite la nivelul navigatorului. Acest cod este încapsulat în documentul XHTML următor:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="ro" xml:lang="ro">
<head>
<title>Testarea jurnalizarii</title>
<script type="text/javascript" src="jurnal.js"></script>
<!-- Un exemplu de cod JavaScript care va cauza erori -->
<script type="text/javascript">
// functie de tratare a erorilor survenite
// (va apela la jurnal pentru a consemna erorile si pe server)
function trateazaErrori (mesaj, uri, linie) {
    var eroare = new Error (mesaj);
    eroare.location = uri + ', linia: ' + linie;
    jurnal.jurnalizeaza (eroare);
    // avertizam si utilizatorul
    avertizeazaUtilizatorul (eroare);
    return true;
}
// orice eroare va cauza executia functiei de tratare a erorii
window.onerror = trateazaErrori;

function incorecta () {
    try { // incercam sa executam codul...
        codNecunoscut ();
    }
    catch (eroare) { // prindem eroarea...
        // modificam proprietate 'location', adaugind date noi
        eroare.location = location.href + ', functia: incorecta()';
        eroare.navigator = navigator.userAgent;
        jurnal.jurnalizeaza (eroare);
        avertizeazaUtilizatorul (eroare);
    }
}

// functie de avertizare (enervare?) a utilizatorului
function avertizeazaUtilizatorul (eroare) {
    document.write ('A intervenit o eroare!
        Vom semnala aceasta eroare programatorilor: ' + eroare.location);
    // redirectam navigatorul spre o pagina fara erori ; )
    //location.href = '/';
}
</script>

```

```

</head>
<body>
<p>Incercam sa cauzam erori JavaScript...</p>
<!-- executam functia JavaScript incorecta() -->
<script type="text/javascript">incorecta()</script>
</body>
</html>

```

Maniera ilustrată de exemplul de față poate fi extinsă la raportarea unor informații adiționale (e.g., rezoluția calculatorului-client, preferințele lingvistice ale vizitatorului etc.) cu scopul de monitorizare/ameliorare a interacțiunii cu utilizatorul.

Listing-ul complet al programelor prezentate mai sus este disponibil la adresa <http://www.infoiasi.ro/~web/ajax-demo.tgz>.

3.4 Aspecte importante în legătură cu realizarea de aplicații AJAX

Printre cele mai importante aspecte de care trebuie să se țină cont se enumeră următoarele (Baekdal, 2005):

- evitarea încărcării întregii pagini – prin intermediul obiectului *XMLHttpRequest* și a modelului DOM se pot modifica doar fragmente de document, fără ca întreg documentul să fie necesar a fi (re)încărcat de pe server; un aspect negativ este acela că în unele situații utilizatorii nu vor putea realiza „semne de carte” ale paginii (*bookmarking*-ul poate fi compromis);
- dezvoltatorii trebuie să facă distincție clară între aplicație Web și sit Web – o *aplicație Web* reprezintă o colecție interconectată de pagini Web cu conținut dinamic menită a oferi o *funcționalitate specifică* utilizatorilor (Buraga, 2005); natural, interacțiunea dintre aplicație și utilizatori are loc prin intermediul unei interfețe Web, așteptările (*expectations*) vizitatorilor trebuind să primeze; interfața aplicației Web trebuie să respecte reglementările și standardele în vigoare în ceea ce privește proiectarea interfeței cu utilizatorul (Buraga, 2003a);
- oferirea de alternative la AJAX, când suportul pentru el nu există implementat – trebuie ținut seama de faptul că nu orice *browser* oferă suport pentru AJAX; de asemenea, utilizatorul ar putea avea dezactivat opțiunea de rulare a programelor JavaScript;
- eliminarea paginilor de confirmare – folosind AJAX, numărul paginilor de confirmare sau de semnalare a unor erori scade, ceea ce conduce la o interacțiune mai bună cu utilizatorul.

În ceea ce privește principiile de proiectare a aplicațiilor AJAX (aliniată așadar „noului” Web), se pot menționa (Mahemoff, 2005):

- minimizarea traficului dintre navigator și server – aplicația Web trebuie să răspundă în timp util solicitărilor utilizatorului, volumul de date transmis în manieră asincronă fiind cât mai posibil redus;

- stabilirea unui mod de interacțiune clar – metaforele de interacțiune trebuie să fie predictibile, utilizatorul știind la ce trebuie să se aștepte din partea interfeței (interacțiune HTML *versus* AJAX *versus* aplicație convențională);
- evitarea confuziilor, prin adoptarea convențiilor de interacțiune Web/clasică – utilizatorul trebuie să fie capabil să învețe rapid maniera de manipulare a aplicației Web;
- eliminarea distragerii utilizatorului – *e.g.*, folosirea de animații gratuite;
- adoptarea tehnologiilor AJAX pentru creșterea utilizabilității, nu doar de dragul tehnologiei.

Dintre șabloanele arhitecturale existente, le enumerăm pe cele de bază – detalii pe situl *AJAXPatterns.org*:

- tratarea evenimentelor la nivel local – pe cât posibil, evenimentele survenite trebuie rezolvate la nivelul clientului, fără a se iniția cereri (și, implicit, transferuri) către aplicații-server;
- reîmprospătarea periodică a conținutului – mai ales în ceea ce privește aplicațiile Web cu un grad ridicat de interacțiune (de exemplu, oferind informații care se perimează rapid, precum cele financiare), navigatorul va trebui să se sincronizeze cu serverul; informațiile pot să se schimbe pe partea de server după ce utilizatorul a efectuat ultima sa modificare, datorită folosirii în manieră concurrentă a aplicației;
- anticiparea *download*-urilor (pre-încărcarea datelor ce vor fi solicitate) – aplicația Web trebuie să ia în calcul posibilele acțiuni viitoare pe care le va realiza vizitatorul și să se comporte în consecință;
- transmiterea explicită a datelor spre server – utilizatorul trebuie să aibă posibilitatea de-a solicita explicit transferul informațiilor și nu numai când survin diverse evenimente ori la fiecare nou *input* al vizitatorului;
- oferirea de posibilități de *bookmarking* – utilizatorii trebuie să fie capabili să realizeze și să partajeze „semne de carte” privind aplicația; unele rezultate ale interacțiunii dintre client și server via obiectul *XMLHttpRequest* ar putea să nu reflecte modificări în cadrul URI-ului aplicației.

De asemenea, pot fi formulate o serie de principii de afișare a datelor (Mahemoff, 2005):

- folosirea proprietăților CSS – pe cât posibil, trebuie utilizate proprietățile de stil puse la dispoziție de CSS pentru a face distincția clară între structură, datele propriu-zise și maniera de afișare (aceasta din urmă ar putea fi dependentă de dispozitivul de redare);
- adoptarea principiilor de utilizabilitate – ca și în cazul aplicațiilor convenționale, disponibile pe platformele consacrate, aplicațiile Web vor

trebui concepute avându-se în vedere utilizabilitatea, în general, și accesibilitatea, în special – detalii în lucrarea (Buraga, 2005);

- indicarea „vârstei” informației afișate – pentru ca utilizatorii să aibă deplină încredere în informațiile puse la dispoziție, trebuie specificată data afișării datelor importante (e.g., prețuri, acțiuni, parametri de stare/control etc.);
- oferirea de indicii privind ce date au fost deja transmise serverului și ce date se află în așteptare (*pending*) pentru a fi transmise – utilizatorul trebuie să cunoască starea în care se află aplicația Web și dacă informațiile au fost transmise cu succes sau nu la server.

Desigur, se pot adopta diverse șabloane de interacțiune, unele dintre ele similare celor folosite în cazul aplicațiilor convenționale: *drag & drop*, *popup data input*, *popup information*, *highlighting*, *auto-completion* etc.

3.5 AJAX în contextul REST

Putem considera că – via AJAX – se pot implementa servicii Web asincrone, în stilul REST (*REpresentational State Transfer*), arhitectură de dezvoltare a aplicațiilor Web. Conform filosofiei REST, ale cărei baze au fost puse în (Fielding, 2000), rezultatul unei procesări conduce la returnarea unei *reprezentări* a unei resurse Web. Orice accesare a unei reprezentări plasează aplicația-client într-o stare care va fi schimbată în urma unui *transfer* de date (accesarea altei reprezentări, pe baza traversării unei hiper-legături – desemnate de un URI – incluse în reprezentarea resursei inițiale). Transferul se realizează prin HTTP, reprezentarea este marcată în XML, iar adresabilitatea se rezolvă via URI, spațiul World-Wide Web putând fi considerat ca fiind un sistem REST.

Serviciile Web actuale se pot dezvolta în concordanță cu arhitectura REST. Componentele care invocă funcționalități vor consuma reprezentări de resurse (în stilul *pull*) conform clasicii arhitecturi client/server. Fiecare cerere este considerată independentă, fără a se lua în considerație contextul – conexiuni *stateless*. Resursele Web pot fi accesate printr-o interfață generică pusă la dispoziție de metodele protocolului HTTP: GET, POST, PUT și DELETE. Actualmente, sunt utilizate preponderent GET și POST. A se vedea și (Erl, 2005) și (He, 2004).

Aplicațiile folosind AJAX pot fi, astfel, considerate servicii Web implementate conform REST, deoarece inter-schimbul de date se desfășoară marcat în XML prin intermediul protocolului HTTP, respectându-se paradigma client/server. Orice resurse vehiculate sunt adresate via identificatori uniformi de resurse.

3.6 Utilizări și interfețe de programare AJAX

AJAX reprezintă una dintre componentele-cheie ale „noului” val de aplicații aliniat problematicilor *Data Web*-ului (a se vedea și cele expuse în subcapitolul 2). O serie dintre siturile majore înglobează AJAX în vederea oferirii

vizitatorilor a unei interacțiuni complexe și mulțumitoare. Drept exemple notabile, putem enumera:

- *24SevenOffice* – serviciu pentru managementul relațiilor cu clienții (CRM – *Customer Relationship Management*);
- *A9.com* – sit *e-business* subsidiar Amazon;
- *EpiphanyRadio* – sit de știri radio emise via Web;
- *Flickr.com* – sit de organizare (clasificare) și de partajare de imagini, adnotate liber de utilizatori;
- *Google Maps* – serviciu Google oferind informații referitoare la hărți și obiective socio-geografice;
- *Google Suggest* – serviciu Google punând la dispoziția vizitatorilor a unei liste de sugestii de termeni privind căutarea pe Web, în funcție de datele specificate de utilizatori;
- *Gmail* – serviciu Google destinat managementului de mesaje de poștă electronică.

Deja există diverse cadre de lucru (*frameworks*) care oferă modalități de programare facilă a aplicațiilor AJAX. Astfel, pot fi folosite biblioteci de funcții JavaScript precum *Google AJAXSLT*, *Prototype*, *Rico* sau *Sarissa*. De asemenea, se poate recurge la funcționalități implementate la nivel de server: *AjaxAnywhere*, *Direct Web Remoting (DWR)*, *JavaScript Remote Scripting (JSRS)* ori *SAJAX* – detalii în (Zametti *et al.*, 2005).

4. ÎN LOC DE CONCLUZII

În cadrul acestui capitol, am detaliat o serie dintre direcțiile de viitor în dezvoltarea de aplicații destinate „noului” Web, cu un accent mai pronunțat asupra suitei de tehnologii AJAX.

În viitorul apropiat, AJAX poate fi văzut ca fiind unul dintre elementele de construcție a Web-ului social (*Social Web*), etapă premergătoare constituirii Web-ului semantic. Aplicațiile AJAX și nu numai ele vor oferi premisele realizării de hiper-legături între persoane și organizații, nu doar între mașini și documente, prin intermediul așa-numiților identificatori extensibili de resurse (XRI – *Extensible Resource Identifiers*), persistenți la schimbări și rezolvând problemele legate de intimitate personală (*privacy*) și încredere (*trust*) – a se consulta lucrările (Drummond & Strongin, 2004) și (Drummond *et al.*, 2004).

În concordanță cu acest Web social, *Data Web*-ul poate fi considerat drept soluție simplificatoare pentru inter-schimb de date, bazată pe principiile arhitecturale ale Web-ului și pe conceptele de bază ale serviciilor Web și Web-ului semantic. Conform (Drummond & Strongin, 2004), datele și ofertanții de date vor fi identificate via XRI, reprezentarea și „legarea” datelor se vor realiza

printr-o schemă XDI (*XRI Data Interchange*), iar inter-schimbul de date va avea loc grație serviciilor XDI (extensii ale serviciilor Web actuale).

Categoriile de aplicații ale Web-ului social și cel de date, care vor beneficia și de tehnologiile care compun AJAX, se prefigurează a fi următoarele:

- porți (portaluri) de acces la contacte personale;
- filtre de încredere (*trust filters*);
- aplicații de management inteligent al *e-mail*-ului;
- calendare de evenimente și semne de carte generate automat;
- posibilitatea auto-înregistrării, auto-conectării și auto-personalizării utilizatorilor;
- protecția furtului identității digitale;
- aplicații pentru efectuarea de căutări cu specific social (*social search*), precum (re)găsirea prietenilor, grupurilor de interes, comunităților axate pe un anumit domeniu etc.;
- rețele de reputație (*reputation networks*).

În încheiere, putem concluziona cu aserțiunea că actualele și viitoarele aplicații, aliniate „nouului” Web, vor trebui să integreze servicii oferite de dispozitive mobile, calculatoare personale, servere etc. De asemenea, trebuie remarcat faptul că atunci când dispozitivele și programele sunt conectate la Internet, aplicațiile nu mai constituie artefacte software, ci devin servicii în permanentă dezvoltare (inclusiv cu aportul utilizatorilor finali, în concordanță cu paradigma *extreme programming*) – apare astfel fenomenul numit *the perpetual beta* (O’Reilly, 2005). „Useful software written above the level of the single device will command high margins for a long time to come.” (Dave Stutz)

Referințe

- Baekdal, T., *XMLHttpRequest Usability Guidelines*, Baekdal.com, 2005
- Berners-Lee, T., *Information Management: A Proposal*, CERN, 1989:
<http://www.w3.org/History/1989/proposal.html>
- Berners-Lee, T., *Weaving the Web*, Orion, Cambridge, 1999
- Berners-Lee, T., Hendler, J., Lassila O., „The Semantic Web”, *Scientific American*, 5, 2001
- Bray, T. et al. (eds.), *Extensible Markup Language 1.0 (Third Edition)*, W3C Recommendation, Boston, 2004: <http://www.w3.org/TR/REC-xml>
- Buraga, S., *Tehnologii Web*, Matrix Rom, București, 2001:
<http://www.infoiasi.ro/~busaco/books/web.html>
- Buraga, S. et al., *Programare Web în bash și Perl*, Polirom, Iași, 2002:
<http://www.infoiasi.ro/~cgi/>
- Buraga S., „Suportul pentru implementare”, în Pribeanu, C. (ed.), *Introducere în interacțiunea om-calculator*, Matrix Rom, București, 2003

- Buraga, S. (coord.), *Aplicații Web la cheie*, Polirom, Iași, 2003:
<http://www.infoiasi.ro/~phpapps/>
- Buraga, S., *Semantic Web. Fundamente și aplicații*, Matrix Rom, București, 2004:
<http://www.infoiasi.ro/~sweb/>
- Buraga, S. (coord.), *Situri Web la cheie. Soluții profesionale de implementare*, Polirom, Iași, 2004:
<http://www.infoiasi.ro/~busaco/books/webapps/>
- Buraga, S., *Proiectarea siturilor Web* (ediția a doua), Polirom, Iași, 2005:
<http://www.infoiasi.ro/~design/>
- Candell, E., Raggett, D., *Multimodal Interaction Use Cases*, W3C Note, Boston, December 2002:
<http://www.w3.org/TR/mmi-use-cases/>
- Clark, J. (ed.), *XSL Transformations (XSLT) – Version 1.0*, W3C Recommendation, Boston, 1999:
<http://www.w3.org/TR/xslt>
- Davies, J., Fensel, D., van Harmelen, F. (eds.), *Towards the Semantic Web*, John Wiley & Sons, 2003
- Doctorow, C. et al., *Essential Blogging*, O'Reilly & Associates, 2002
- Dodds, L., *An Introduction to FOAF*, XML.com, 2004:
<http://www.xml.com/pub/a/2004/02/04/foaf.html>
- Drummond, R., Strongin, G., „The Dataweb: An Introduction to XDI”, *White Paper*, OASIS XDI Technical Committee, 2004
- Drummond, R. et al., „The Social Web: Creating An Open Social Network with XDI”, *Planetwork Journal*, July 2004: <http://journal.planetwork.net/article.php?lab=reed0704&page=1>
- Eernisse, M., *A Simpler Ajax Path*, OnLamp.com, 2005:
<http://www.onlamp.com/pub/a/onlamp/2005/05/19/xmlhttprequest.html>
- Erl, T., *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR, 2005
- Fielding, R., „Architectural Styles and the Design of Network-based Software Architectures”, *Ph.D. Dissertation*, 2000: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- Hammond, T. et al., „Social Bookmarking Tools (I). A General Review”, *D-Lib Magazine*, vol. 11, no. 4, April 2005: <http://www.dlib.org/dlib/april05/hammond/04hammond.html>
- He, H., *What is Service-Oriented Architecture?*, XML.com, 2003:
<http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>
- He, H., *Implementing REST Web Services: Best Practices and Guidelines*, XML.com, 2004:
<http://www.xml.com/pub/a/2004/08/11/rest.html>
- Le Hors et al. (eds.), *Document Object Model (DOM) Level 2 Core Specification – Version 1.0*, W3C Recommendation, Boston, 2000: <http://www.w3.org/TR/DOM-Level-2-Core>
- Lund, B. et al., „Social Bookmarking Tools (II). A Case Study – Connota”, *D-Lib Magazine*, vol. 11, no. 4, April 2005: <http://www.dlib.org/dlib/april05/lund/04lund.html>
- Mahemoff, M., „AJAX Patterns: Design Patterns for AJAX Usability”, *Software As She's Developed Weblog*, AJAXPatterns.org, 2005
- Manola, F., Miller, E. (eds.), *RDF (Resource Description Framework) Primer*, W3C Recommendation, Boston, 2004: <http://www.w3.org/TR/rdf-primer/>
- Mathes, A., *Folksonomies – Cooperative Classification and Communication Through Shared Metadata*, University of Illinois Urbana-Champaign, 2004: <http://www.adammathes.com/>
- McLellan, D., *Very Dynamic Web Interfaces*, XML.com, 2005:
<http://www.xml.com/pub/a/2005/02/09/xml-http-request.html>
- O'Reilly, T., „The Architecture of Participation”, *O'Reilly Developer Weblogs*, O'Reilly.com, April 2003: <http://www.oreillynet.com/pub/wlg/3017>

- O'Reilly, T., *What is Web 2.0. Design Patterns and Business Models for the Next Generation of Software*, O'Reilly.com, 2005:
<http://www.oreilynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>
- Zametti, F. *et al.*, *Survey of AJAX/JavaScript Libraries*, OSAF.com, 2005:
<http://wiki.osafoundation.org/bin/view/Projects/AjaxLibraries>
- ***, DCMI (*Dublin Core Metadata Initiative*): <http://www.dublincore.org/>
- ***, ECMA (*European Computer Manufacturers Association*): <http://www.ecma.ch/>
- ***, FOAF (*Friend Of A Friend*) Specification: <http://rdfweb.org/>
- ***, RSS (*Really Simple Syndication*) 2.0 Specification: <http://blogs.law.harvard.edu/tech/rss>
- ***, World-Wide Web Consortium, Boston, 2005: <http://www.w3.org/>

Capitolul 2

DEMOCRAȚIA PE WEB: SITURILE WIKI

Diana Gorea

Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” Iași
Str. G-ral Berthelot, nr. 16, Iași, România
dgorea@infoiasi.ro – <http://www.infoiasi.ro/~dgorea/>

Rezumat. Lucrarea investighează diverse aspecte tehnologice și sociale legate de una dintre cele mai importante tendințe în evoluția actuală a Web-ului, și anume siturile colaborative de tip Wiki. În construcția unui astfel de sit sunt angrenate sisteme de baze de date și tehnologii software complexe. De asemenea, sunt prezentate cele mai reprezentative aplicații Wiki care sunt utilizate la ora actuală de către diverse organizații în intranet sau în mod public pe Internet.

Cuvinte-cheie: Wiki, colaborare, aplicații Web.

1. CONCEPTUL DE WIKI

În (Leuf & Cunningham, 2001) un Wiki este definit ca o colecție liber expandabilă de pagini Web interconectate, un sistem hipertext de stocare și modificare a informațiilor (o bază de date) în care fiecare pagină este editabilă într-o manieră facilă de către orice utilizator care dispune de un navigator Web având suport pentru formulare.

Siturile Wiki pun la dispoziția utilizatorilor o sintaxă simplă bazată pe text pentru crearea *ad-hoc* de noi pagini sau de hiper-legături interne (între paginile aceluiași Wiki) sau externe.

Paginile Wiki sunt controlate (create, modificate, șterse, mutate, redenumite, corelate) printr-un limbaj de programare sau de *scripting* și păstrate fie ca text într-un sistem de fișiere, fie într-o bază de date relațională externă (MySQL, Oracle, PostgreSQL). Paginile Wiki sunt redactate ca reprezentări HTML de către serverul Wiki.

Dintre cele mai importante utilizări ale siturilor Wiki amintim:

- gestiunea colaborativă a cunoștințelor: adăugare și modificare de conținut, clasificarea conținutului;
- organizarea de evenimente: conferințe, concursuri, expoziții etc.;

- *brainstorming* individual și colectiv: se pot posta idei personale sau publice, se pot interconecta aceste idei cu ale altora prin hiper-legături și crea astfel un sit Web;
- managementul de informații personale (*Personal Information Manager*);
- gestionarea documentațiilor pentru aplicații software, a manualelor de procedură, a conținutului academic;
- creație artistică colectivă (carte, film, piesă de teatru);
- unealtă de predare;
- proiecte corporatiste (intranet, extranet, fluxuri de producție);
- crearea și gestionarea listelor FAQ (*Frequently Asked Questions*);
- un *weblog* mai flexibil.

Încă de la începutul apariției lor, siturile Wiki au avut un succes enorm și au fost adoptate în cadrul comunităților tehnice pentru a inter-schimba informații folosind doar navigatoarele Web standard. Succesul se datorează versatilității și a numeroaselor posibilități de utilizare. Provocarea actuală constă în a extinde acest succes și asupra utilizatorilor obișnuiți.

2. EVOLUȚIE

Termenul de Wiki a fost introdus de Ward Cunningham, pornind de la termenul hawaian Wiki-Wiki, care înseamnă „foarte rapid” (*very quick*). Deci, un sit Wiki înseamnă un sit Web rapid, deoarece simplifică procesul de creare a siturilor Web. Primul sit Wiki creat se numea WikiWikiWeb și a fost inițiat de Ward Cunningham în 1995 în scopul facilitării inter-schimbului de idei între programatori. Aplicația era scrisă în Perl și însoțea *Portland Pattern Repository*, bază de cunoștințe în domeniul *design pattern*-urilor de programare, în special *extreme programming*. Ideea lui Cunningham era ca paginile aplicației WikiWikiWeb să fie editabile de către utilizatori, care, ținând cont de specificul sitului, erau în cea mai mare parte programatori. După cum era de așteptat, ideea a avut succes în rândul comunității programatorilor, care puteau astfel să schimbe idei de programare într-un format comod și ușor de înțeles.

Spre finalul anilor '90, s-a preconizat ideea utilizării tehnologiei Wiki pentru gestiunea bazelor de cunoștințe publice și private. Astfel, fondatorii enciclopediei electronice libere scrisă de experți Nupedia, au angajat tehnologia Wiki în construcția unui proiect complementar Nupediei. Acesta este Wikipedia și a fost lansat în ianuarie 2001, cu scopul păstrării articolelor înainte de a fi evaluate și incluse în Nupedia. Datorită contribuțiilor libere ale utilizatorilor, Wikipedia a cunoscut o dezvoltare expansivă, independentă de Nupedia, du-

când în septembrie 2003 la încetarea activității Nupediei, puținul conținut rămas al acesteia fiind asimilat de Wikipedia.

La începutul anilor 2000, Wiki-urile au pătruns și în cadrul organizațiilor sub formă de software colaborativ pentru comunicare intranet în cadrul proiectelor, destinat inițial comunităților tehnice. În decembrie 2002, compania Socialtext a lansat prima soluție Wiki comercială, urmată de o serie de alte soluții *open source* cum ar fi MediaWiki, TWiki, XWiki. Astăzi există foarte multe companii care folosesc tehnologia Wiki ca unicul software colaborativ intranet. De fapt, acest tip de utilizare a Wiki-urilor este, de departe, mai răspândit decât utilizarea publică în Internet.

2.1 Ideile care stau la baza noțiunii de Wiki

Geneza conceptului de Wiki, precum și evoluția sa poate fi explicată prin câteva teorii și idei precum: inteligența colectivă, teoria haosului, viața artificială, comportament emergent.

Teoria haosului se referă la unele sisteme dinamice neliniare care prezintă fenomenul cunoscut sub numele de haos, caracterizat printr-o sensibilitate la condițiile inițiale. Ca rezultat, comportamentul sistemelor pare a fi aleator, deși modelul sistemului este determinist. Exemple de astfel de sisteme sunt atmosfera terestră, sistemul solar, plăcile tectonice, creșterea populației, diverse fenomene economice etc.

Comportamentul emergent reprezintă procesul de formare de *pattern*-uri complexe pornind de la reguli simple. Un exemplu este interacțiunea dintre un număr mare de neuroni care dă naștere gândirii umane (neuronul în sine neavând această calitate). În general, fenomenul nu este predictibil pornind de la o descriere a nivelului inferior.

Viața artificială reprezintă studiul vieții prin intermediul unor sisteme identice create de om, studiu care utilizează algoritmi evolutivi, modele bazate pe agenți, automate celulare etc.

În continuare, vom descrie pe scurt inteligența colectivă, considerată a avea rolul cel mai important în funcționarea siturilor Wiki.

2.1.1 Inteligența colectivă

Inteligența colectivă, așa cum a fost definită de George Por în *Blog of Collective Intelligence*, reprezintă capacitatea comunității umane de a evolua către o gândire de complexitate superioară, rezolvare de probleme și integrare, prin colaborare și inovare. Conceptul s-a născut pornind de la următoarea idee: cunoașterea și capacitatea de rezolvare a unei probleme a unui grup este mult mai mare decât cea a unui singur individ. De asemenea, în acest context poate fi amintită și legea lui Metcalfe, care spune că valoarea unei rețele este $O(n^2)$, unde n reprezintă numărul de utilizatori din sistem.

Inteligența colaborativă distribuită este activitatea unui grup care colaborează într-o sferă de interacțiune virtuală. Membrii grupului pot interacționa în timp real sau asincron chiar dacă nu sunt localizați în același spațiu fizic. Tehnologiile pentru facilitarea dezvoltării inteligenței colaborative distribuite sunt cunoscute sub numele de software colaborativ (sau *groupware*). Din această categorie fac parte programele de tip *instant messenger*, *chat rooms*, conferințe Web, calendarele electronice, sistemele de gestiune a cunoștințelor, sistemele de management a proiectelor, poșta electronică, forumurile, *weblog*-urile și, bineînțeles, sistemele de tip Wiki.

Factorii care contribuie decisiv la obținerea unui grad ridicat de inteligență colectivă sunt:

- moderarea și încurajarea grupului;
- aderarea la un număr minimal de reguli privind interacțiunea între utilizatori;
- promovarea gândirii creative;
- un *feedback* susținut și de calitate al grupului;
- ideile trebuie încurajate, dar soluțiile trebuie confirmate după recenzia de către grup;
- construirea unei baze de cunoștințe bine documentate, care constituie memoria partajată a grupului.

Un proiect deosebit de interesant (bineînțeles, de tip Wiki), dedicat meta-colaborării (colaborare în privința colaborării) este cel de la adresa <http://collaboration.wikicities.com>. Scopul acestuia este explorarea similarităților și diferențelor dintre natura, metodele și motivațiile colaborărilor din diverse domenii. Se dorește crearea unui depozit de cunoștințe despre colaborare în continuă dezvoltare, crearea unei comunități de cercetători interesați de înțelegerea colaborării și dezvoltarea unei teorii generale a colaborării.

3. PRINCIPII DE PROIECTARE A SITURILOR WIKI

Atunci când a inventat conceptul, Ward Cunningham a avut în vedere câteva principii de bază, principii care se aplică și astăzi în proiectarea unui sit Wiki. Astfel, un sit Wiki trebuie să aibă următoarele caracteristici:

- deschis – dacă un utilizator găsește o pagină incompletă, cu un conținut prost organizat, este liber să opereze modificări asupra acesteia;
- incremental – paginile pot cita alte pagini, chiar pagini care nu au fost încă scrise;
- organic – structura și conținutul text evoluează în permanență;

- monden – există un număr redus de convenții care furnizează accesul la marcasele de pagină cele mai uzuale;
- universal – mecanismele de scriere sunt aceleași cu cele de organizare și editare, astfel încât un utilizator care scrie, editează și organizează în același timp;
- clar – ieșirea formatată va sugera intrarea necesară pentru a o produce;
- unificat – paginile vor fi extrase dintr-un spațiu plat, astfel încât, pentru a le interpreta nu este nevoie de vreun context adițional;
- precis – denumirea paginilor va fi făcută într-un mod precis, sugestiv, astfel încât să nu aibă loc coliziuni de nume;
- tolerant – comportamentul interpretabil este preferat returnării mesajelor de eroare;
- observabil – activitatea pe un sit Wiki poate fi vizualizată și verificată de către orice vizitator;
- convergent – redundanța trebuie evitată prin găsirea și citarea conținutului similar;
- încrederea – această calitate este esențială pentru funcționarea unui sit Wiki. „*Trust the people, trust the process, enable trust-building.*” (Ward Cunningham). Wiki se bazează pe presupunerea că utilizatorii sunt bine intenționați. În orice caz, nu se exclude posibilitatea exploatării malițioase a siturilor Wiki. Cu toate acestea, este mai ușor să se corecteze greșelile, nu să se prevină. Orice modificare este reversibilă prin intermediul jurnalelor. De asemenea, utilizatorii direct interesați de calitatea anumitor pagini, le pot monitoriza, fiind notificați la apariția unei modificări. Cu toată această filosofie deschisă și deci expusă la vandalisme, siturile Wiki funcționează cu succes. Conform unui studiu al IBM, orice vandalism asupra Wikipediei este retractat în aproximativ 5 minute.

4. WEB 2.0

Deși nu există o definiție unanim acceptată a ceea ce înseamnă Web 2.0, termenul se referă la o generație de situri și servicii care folosesc Web-ul ca platformă (de exemplu, RSS *Syndication*, *blogging*, *Wiki*, *Flickr*, *del.icio.us*, *connotea*, *BitTorrents*, *AJAX*), majoritatea implicând contribuții de la utilizatorii de rând. Este un concept deschis, în spirit *open source*. În plus, are în centru ideea de utilizatori uman, întrucât are potențialul de a crea o țesătură socială strânsă între indivizi și comunități online. „*Web 2.0 is collaboration among people in a way that scales*” (Ward Cunningham).

Termenul de Web 2.0 este folosit și pentru a desemna noțiunea de Web semantic. Cele două semnificații sunt cumva conexe și complementare, conjuncția lor ducând la apariția noțiunii de Web semantic emergent. Astfel, rețelele sociale și *folksonomiile* (clasificări *ad-hoc* de către utilizatori a obiectelor cu care aceștia interacționează *online*), creează o bază pentru un mediu semantic prin crearea de metadata interpretabile de către mașină. Web-ul nu este un mediu prielnic numai mașinilor, cât și clienților umani. Majoritatea aplicațiilor Web 2.0 se bazează pe micro-conținut provenind din surse diferite, distribuite, care sunt procesate de către mașină, reutilizate, plasate în alt context. Astfel, utilizatorii pot obține micro-conținutul de care sunt interesați într-un mod agregat, asamblat, structurat și personalizat, dintr-o perspectivă *ad-hoc*.

5. APLICAȚII WIKI

O aplicație Wiki este un tip de software colaborativ care permite dezvoltarea unui sit Wiki. De regulă este implementată ca un script pe partea de server, care rulează pe unul sau mai multe servere Web. Unele aplicații Wiki au propriul server Web sau sunt „împachetate” împreună cu acesta. De aceea, noțiunea de aplicație Wiki poate fi interpretată ca totalitatea tehnologiilor software care permit rularea unui sit Wiki. Pentru memorarea conținutului se poate folosi fie un sistem de baze de date relaționale, fie un sistem de fișiere.

Sistemele Wiki pot fi folosite și ca sisteme de gestiune a conținutului (*Content Management Systems*). Cea mai importantă diferență între sistemele Wiki și sistemele de gestiune a conținutului este modalitatea de editare a conținutului. În plus, în cazul sistemelor Wiki accentul cade în primul rând pe conținut în detrimentul prezentării.

Dintre cele mai importante aplicații Wiki amintim WikiWikiWeb, MediaWiki, MoinMoin, TWiki, XWiki, KWiki etc. Majoritatea aplicațiilor Wiki se află sub licență GPL, iar cele mai complexe (TWiki, MediaWiki, XWiki) sunt dezvoltate colaborativ sau modular, punând la dispoziție API-uri (*Application Programming Interface*) care permit dezvoltarea de noi extensii.

Câteva dintre acestea sunt descrise pe scurt în secțiunile următoare.

5.1 MediaWiki

MediaWiki este un software de tip Wiki sub licență GPL, scris în PHP și care folosește baze de date MySQL. A fost utilizat inițial de Wikipedia, apoi și de alte proiecte ale fundației Wikimedia. Software-ul este actualmente la versiunea 1.5.2, poate fi portat pe mai multe platforme și poate fi descărcat de la <http://sourceforge.net/projects/wikipedia>.

Fundația Wikimedia se află în spatele a câtorva dintre cele mai importante proiecte construite în mod colaborativ. Printre acestea se numără:

- enciclopedia *online* Wikipedia, unul din cele mai vizitate 50 situri din lume,
- Wiktionary, un dicționar multilingv,
- Wikibooks, o colecție de cărți cu conținut deschis,
- Wikiquote, o colecție de citate celebre,
- Wikinews, o sursă liberă de știri,
- Wikisource, o colecție de texte originale publice,
- Wikimedia Commons, un depozit de date multimedia.

Implementarea oferă un set de facilități de bază, printre care controlul versiunilor, spații de nume, posibilitatea urmării paginilor, securitate, *template*-uri și *skin*-uri, liste de discuții.

În baza de date se află conținutul paginilor sub formă de wikitext, precum și alte informații auxiliare despre pagini, utilizatori etc. De asemenea, în baza de date se memorează versiunile anterioare ale paginilor, având propriul sistem de control al versiunilor. Baza de date a Wikipedia este în continuă creștere. Astfel, în februarie 2005, tabelul cu versiunile curente ale paginilor (vorbit de Wikipedia în engleză) avea 3 GB (având indexul de 500 MB), iar arhiva 80 GB (cu indexul 3GB). Baza de date este disponibilă pentru *download* sub formă de *dump*.

În figura 1 este descrisă arhitectura generală a unei aplicații MediaWiki.

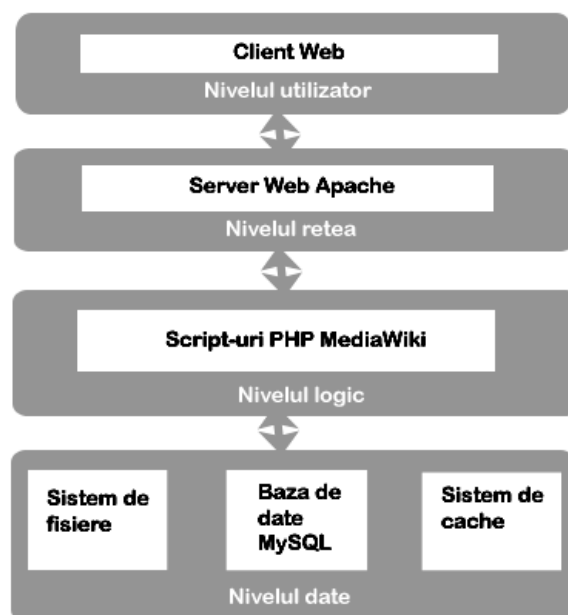


Figura 1. Arhitectura generală a aplicației MediaWiki

După cum spuneam la începutul acestei secțiuni, baza de date păstrează conținutul sub formă de wikitext. Wikitextul este un document scris într-un limbaj

de marcare Wiki, fiind un amestec de conținut, marcaje și metadate. Când se vizualizează o pagină, wikitext-ul este convertit în XHTML (sau conținutul XHTML este preluat din *cache*), care este redat de navigatorul Web al utilizatorului. Codul XHTML generat depinde de mai mulți factori, printre care:

- modul de lucru (vizualizare sau editare),
- valorile variabilelor,
- *skin*-ul utilizat,
- drepturile utilizatorului,
- spațiul de nume activ, dacă pagina este urmărită de utilizatorul respectiv.

Dacă utilizatorul alege să modifice o pagină (apăsând pe butonul *Edit*), acesta primește chiar wikitextul (întregii pagini sau doar al secțiunii selectate). După eventualele modificări, la apăsarea butonului *Preview*, wikitextul este trimis serverului, care îl convertește în XHTML, fiind redat apoi de navigator. După apăsarea butonului *Save*, modificările sunt permanentizate în wikitextul din baza de date.

Pentru mai multe informații despre aplicația MediaWiki se poate consulta situl Wikimedia Metawiki, care conține informații despre toate proiectele fundației Wikimedia.

5.2 TWiki

TWiki este o aplicație Wiki structurată, scrisă în limbajul Perl și aflată sub licență GPL, care se dorește a fi o platformă *enterprise* colaborativă. O aplicație Wiki structurată încearcă să combine avantajele unei aplicații Wiki (conținut interconectat, organic și liber, încrederea „controlată”) cu cele ale unei aplicații de baze de date (date foarte structurate, raportare facilă, controlul accesului). Această combinație produce o serie de avantaje, cum ar fi flexibilitatea adăugării de conținut liber celui structurat și, invers, utilizarea TWiki în construcția de noi aplicații Wiki.

TWiki a fost adoptat în spatele *firewall*-urilor câtorva companii importante, cum ar fi Motorola, Yahoo!, SAP, Disney Corporation, Inktomi, datorită interfeței prietenoase și a flexibilității. Aplicația este utilizată în principal ca unealtă pentru gestionarea dezvoltării proiectelor și a documentațiilor.

Principalele particularități ale soluției TWiki sunt:

- gruparea paginilor înrudite în colecții (TWiki *webs*) pentru a separa grupurile colaborative;
- căutare cu sau fără expresii regulate;
- notificare via e-mail în cazul apariției unei modificări într-o colecție;

- posibilitatea clasificării paginilor Web nestructurate și crearea de fluxuri de afaceri;
- posibilitatea atașării de fișiere paginilor Web prin *upload*;
- evidența modificărilor (ce, cine, când);
- utilizarea de variabile pentru a compune pagini Web dinamice;
- posibilitatea extinderii și personalizării funcționalităților prin intermediul *plugin*-urilor (e.g., CalendarPlugin, ChartPlugin, DatabasePlugin, SlideShowPlugin, XPTrackerPlugin etc.);
- posibilitatea utilizării platformei TWiki pentru crearea de aplicații Web specifice unor domenii de activitate;
- separarea prezentării de conținut via *template*-uri și *skin*-uri;
- autentificarea utilizatorilor;
- vizualizarea de noutăți din cadrul colecțiilor (export RSS);
- crearea de statistici asupra colecțiilor și utilizatorilor;
- utilizarea a trei nivele de preferințe: la nivel de sit, la nivel de colecție și la nivel de utilizator;
- avertizarea utilizatorilor atunci când o pagină este în curs de modificare, pentru a evita modificări simultane;
- suport pentru *backlink*-uri (găsirea referențelor pentru o pagină).

5.3 XWiki

XWiki (*eXtended Wiki*) este o soluție Wiki cu o arhitectură robustă și scalabilă bazată pe platforma J2EE, fiind considerat atât un Wiki profesional, datorită facilităților de tip *enterprise*, cât și un Wiki de aplicații, permițând scrierea de scripturi cu date structurate chiar din cadrul interfeței Wiki.

Aplicația poate fi folosită în două moduri: *online*, prin crearea de Wiki-uri proprii, sau pe serverul Web propriu, prin dezvoltarea de aplicații Wiki, fiind o alternativă de cost redus la soluțiile intranet și extranet.

Soluția XWiki prezintă următoarele facilități:

- administrare: documente, membri, albume foto, calendare de evenimente, *blog*-uri, prezentări *online*, utilizatori etc.;
- controlul versiunilor documentelor;
- posibilitatea atașării (prin *upload*) de fișiere la documente;
- căutare text în cadrul Wiki-ului sau căutare SQL în metadata;

- posibilitatea integrării de *feed-uri* RSS;
- posibilitatea editării în stil WYSIWYG (*What You See Is What You Get*) a paginilor din Wiki;
- posibilitatea exportului în format PDF a documentelor;
- schimbarea *look-and-feel*-ului prin crearea de *skin-uri*;
- suport pentru internaționalizare;
- suport puternic pentru programare în cadrul documentelor;
- posibilitatea găzduirii de Wiki-uri virtuale, având adrese Web diferite (și baze de date diferite).

Proiectul XWiki este *open source* și se bazează pe alte câteva proiecte *open source*:

- *Hibernate*, pentru abstractizarea accesului la baza de date și gestionarea atât a documentelor, a meta-datelor, cât și a informațiilor din formulare.
- *DBCP (Database Connection Pool)*, pentru gestionarea eficientă a conexiunilor la baza de date.
- *SVN (Subversion)* pentru gestionarea versiunilor documentelor și a fișierelor atașate.
- *Struts*, pentru gestionarea schemelor URL.
- *Velocity*, pentru gestionarea vizualizării documentelor și pentru generarea de conținut dinamic în documentelor Wiki, folosind elemente de programare.
- *Radeox*, pentru motorul de redare XWiki.
- *OSCache*, pentru păstrarea în *cache* a documentelor.

În comunitatea Wiki, utilizabilitatea este o provocare cheie, ea permițând tuturor utilizatorilor să înțeleagă conținutul și să aducă contribuții la acesta. Din acest motiv, a fost elaborat un prototip destinat proiectelor intranet și organizaționale, care conține ecrane XWiki și funcționalități, furnizând astfel o serie de recomandări ergonomice.

Recomandările au ca scop facilitarea interacțiunii utilizatorilor cu sistemul și se referă la:

- interfața XWiki (prezentarea conceptelor XWiki și translatarea acestora în spații pe ecran),
- prototipul XWiki (introducerea diverselor tipuri de utilizări: navigare, administrare, editare la nivel text sau programatic),
- componentele grafice (descrierea *look-and-feel*-ului componentelor grafice).

O facilitate interesantă asupra căreia vom insista în următoarele o reprezintă posibilitatea utilizatorilor de a programa doar prin intermediul interfeței XWiki, fără a scrie o singură linie de cod. După cum am menționat și la începutul secțiunii, XWiki este un tip avansat de Wiki, având propriul API. Această interfață de programare are mai multe nivele de funcționalitate, în funcție de tipul de conținut cu care interacționează utilizatorul. Unul dintre aceste moduri îl reprezintă editorul de clase (a se vedea figura 2), altele sunt introduse prin intermediul *wizard*-urilor de programare – crearea de obiecte, introducerea de obiecte într-un document etc.

Un alt aspect important și cu un oarecare grad de noutate este legat de unele colaborative puse la dispoziție de sistemul XWiki. Astfel, trebuie să existe funcționalități pentru interacțiunea între membrii grupului, structurarea bazei de cunoștințe a organizației și facilitarea oricăror operațiuni ale acesteia.

Deoarece aceste operațiuni depind de tipul organizației, sistemul nu trebuie totuși încărcat cu o multitudine de unelte, ci menținut la un grad rezonabil de simplitate pentru toți utilizatorii. În acest caz sistemul trebuie să fie destul de flexibil pentru a permite administratorului instalarea exact a uneltelor necesare unui anumit context. Cele mai uzuale unelte dintr-un sistem colaborativ sunt: indexul, membrii din spațiul de lucru, gestionarea sarcinilor.

Community Identity
Slogan, motto, logo...

Workspace 0 Workspace 1 Workspace n Workspace n+1

Xwiki editor class Switch

Switch between any available editor's mode (text, html, xml, rdf, Xwiki's coding, Object's command line...)

ClassName .class

* Mandatory: space or special characters not allowed !
Explicit ClassName can help editors and contributor of this class.

XWiki's programming wizard
» Step 1/3: Rename the ClassName field
- add a property or
- edit existing properties
The class must be saved before going to step 2

Add a property

Name

Type

Apply

Class switcher

Class

switch

Properties

Select the property you want to edit below.

Title
Content
Category
extract
Client

Name * title
Information about the field

Pretty Name * Title

Modifiable no

Number 1

Size 80

Save Save and go to step 2 >

Version 1.3 last modified by : User Name on dd/mm/yy at hh:mn
XWiki Editor :: class : ClassName

Figura 2. Interfață Web pentru editarea de clase în XWiki

5.4 Alte aplicații Wiki

UseModWiki este precursorul MediaWiki și a fost folosit din 2001 până în 2002 pentru a rula toate versiunile Wikipediei. În ciuda faptului că oferă destul de puține funcționalități, UseModWiki este una din cele mai populare aplicații Wiki, ca număr de utilizări. Aceasta se datorează instalării facile și faptului că necesită puțină putere de calcul. Aplicația este scrisă în Perl și nu folosește deloc baze de date, conținutul fiind păstrat în fișiere text. Este disponibilă sub licență GPL.

KWiki este o aplicație Wiki scrisă în limbajul Perl de către Brian Ingerson. Particularitatea acestei implementări Wiki constă în modularizare, ceea ce duce la ușurința în instalare, menținere și extindere, precum și la eliminarea redundanței. Astfel, fiecare funcționalitate este implementată ca un *plugin*, multe dintre *plugin*-uri fiind disponibile la CPAN (*Comprehensive Perl Archive Network*). În plus față de funcționalitățile standard ale unui Wiki, KWiki oferă suport pentru prezentări *online*, *blog*-uri, securitate, replicarea paginilor.

MoinMoin este o aplicație Wiki sub licență GPL, scrisă în Python, care păstrează conținutul Wiki în fișiere text. Dintre funcționalitățile notabile amintim prevenirea *spam*-urilor prin filtrarea conținutului cu ajutorul expresiilor regulate, notificare la *email*-uri, personalizarea interfeței prin CSS și XSLT, extensibilitate prin *plugin*-uri, existența unei ediții *desktop*.

SWiki (*Squeak Wiki*) este o implementare în limbajul Squeak a aplicației originale WikiWikiWeb a lui Ward Cunningham. Aplicația este utilizată în principal drept grup colaborativ de pagini Web în câteva universități și a fost dezvoltată de Grupul de Software Colaborativ de la Georgia Institute of Technology. Aplicația are propriul server Web (numit Comanche), care poate coexista cu alt server Web, atât timp cât rulează la alt port. Un SWiki este format dintr-o mașină virtuală (*squeak.exe*), o imagine (*squeak.image*), un set de *template*-uri și Wiki-urile virtuale. O aplicație SWiki poate rula pe orice platformă atât timp cât există instalată mașina virtuală și permite crearea de Wiki-uri virtuale prin intermediul unei interfețe Web de administrare. Cu excepția mașinii virtuale și a imaginii, care sunt disponibile în format binar, toate celelalte fișiere (*template*-urile și paginile Wiki) sunt fișiere XML.

ZWiki este o aplicație Web dezvoltată de Joyful Systems disponibilă sub licență GPL care rulează sub serverul Web Zope. Principalele funcționalități ale ZWiki sunt ierarhiile de pagini, *backlink*-urile, istoricul modificărilor, controlul accesului, suportul pentru preferințele-utilizator, *upload*-ul de fișiere, *skin*-urile, posibilitățile de căutare.

6. CONCLUZII

Siturile Wiki fac parte dintre noile tendințe în evoluția Web-ului, fiind utilizate în principal ca aplicații colaborative intranet în cadrul organizațiilor, precum și în mod public pe internet. Succesul înregistrat de acestea se datorează faptului că reprezintă o soluție simplă și accesibilă care favorizează comunicarea și inter-schimbul de cunoștințe în cadrul grupurilor. În consecință, utilizând acest tip de unelte, timpul de realizare a unui proiect este redus considerabil.

Lucrarea a descris principalele aspecte legate de evoluția și impactul soluțiilor Wiki în peisajul actual al Web-ului, precum și cele mai importante astfel de soluții existente pe piață. Există implementări Wiki în majoritatea limbajelor de programare (de exemplu, Java, Perl, PHP, Python etc.), fiecare din implementări oferind un set mai mare sau mai mic de funcționalități. Majoritatea sunt extensibile prin intermediul *plugin*-urilor, în acest mod putându-se adapta la necesitățile fiecărei organizații în parte.

Ca orice alt concept simplu, conceptul de editare liberă are câteva implicații profunde. Faptul că utilizatorii obișnuiți pot edita în mod instantaneu orice pagină dintr-un sit Wiki promovează crearea de conținut și încurajează utilizarea democratică a Web-ului.

Referințe

Leuf, B., Cunningham, B., *The Wiki Way: Collaboration and Sharing on the Internet*, Addison Wesley, 2001

***, *Apache Struts Project*, <http://struts.apache.org/>

***, *Blog of Collective Intelligence*, <http://www.community-intelligence.com/blogs/public>

***, *Commons DBCP*, <http://jakarta.apache.org/commons/dbcp/>

***, *GNU General Public License*, <http://www.gnu.org/licenses/gpl.html>

***, *GNU Free Documentation License*, <http://www.gnu.org/copyleft/fdl.html>

***, *Hibernate*, <http://www.hibernate.org/>

***, *OSCache*, <http://www.opensymphony.com/oscache/>

***, *Portland Pattern Repository*, <http://c2.com/ppr/>

***, *Radeox Wiki Render Engine*, <http://radeox.org>

***, *Squeak*, <http://www.squeak.org/>

***, *Subversion Project Home*, <http://subversion.tigris.org/>

***, *SWiki*, <http://minnow.cc.gatech.edu/swiki>

***, *TWiki – Enterprise Collaboration Platform*, <http://twiki.org/>

***, *Velocity*, <http://jakarta.apache.org/velocity/>

***, *Wikibooks*, http://wikibooks.org/wiki/Wikibooks_portal

***, *Wikiquote*, http://wikiquote.org/wiki/Main_Page

***, *Wikimedia Metawiki*, <http://meta.wikimedia.org>

***, *Wikimedia Commons*, http://commons.wikimedia.org/wiki/Main_Page

***, *Wikipedia*, <http://www.wikipedia.org/>

***, *Wikisource*, http://sources.wikipedia.org/wiki/Main_Page

***, *Wiktionary*, http://www.wiktionary.org/wiki/Main_Page

***, *WikiWikiWeb Frontpage*, <http://c2.com/cgi/wiki>

***, *XWiki*, <http://xwiki.org/>

***, *Zwiki FrontPage*, <http://zwiki.org/FrontPage>

Capitolul 3

APLICAȚII FĂRĂ FIR BAZATE PE WEB-UL SEMANTIC

Sabin-Corneliu Buraga

Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” din Iași
Str. G-ral. Berthelot, nr. 16 400783 Iași, România
busaco@infoiasi.ro – <http://www.infoiasi.ro/~busaco/>

Rezumat. Capitolul va lua în discuție diversele aspecte referitoare la dezvoltarea de servicii (aplicații) fără fir bazate pe tehnologiile Web-ului semantic. O cerință stringentă a utilizatorilor este aceea de a putea exploata, cu ușurință, dispozitivele fără fir, într-o manieră independentă de o anumită configurație hardware/software sau de o anumită funcționalitate. Lucrarea va investiga o serie dintre oportunitățile pe care le aduc componentele Web-ului semantic (limbajele bazate pe XML, metadatele etc.) în vederea facilitării managementului resurselor rețelelor *wireless* (mobile) constituite *ad-hoc* și compuse din dispozitive eterogene. De asemenea, se va discuta și despre realizarea unei interacțiuni facile dintre utilizatori și calculatoarele fără fir.

Cuvinte-cheie: Web semantic, servicii *wireless*, XML, interacțiune.

1. INTRODUCERE

Proliferarea dispozitivelor de calcul mobile – telefoane mobile, telefoane inteligente (*smart phones*), PDA-uri, *palmtop*-uri, *notebook*-uri, *laptop*-uri, dispozitive de tip *Tablet PC* etc. –, în contextul unei tot mai accentuate nevoi de comunicare inter-umană și de creare a unor medii computaționale ubicue, conduce la necesitatea proiectării și implementării unor aplicații (servicii) independente de platformă, care trebuie descrise într-o formă universală și extensibilă.

Avem aici în vedere mai ales dispozitivele fără fir (*wireless*), având o mobilitate mai ridicată față de alte echipamente de calcul. Caracteristicile tehnice ale dispozitivelor mobile sunt, de cele mai multe ori, mai modeste în comparație cu cele ale calculatoare convenționale de tip *desktop*. Exceptând *Tablet PC*-urile și *laptop*-urile, performanțele resurselor *hardware* sunt scăzute, deoarece dispozitivele sunt echipate cu un procesor mai puțin puternic, având o memorie de capacitate mai mică sau un ecran la rezoluție redusă (Pratt, 2004). Unul dintre avantajele majore este cel al suportului pentru o conectivitate bogată cu alte

dispozitive sau calculatoare via porturi USB (*Universal Serial Bus*) și cu infraroșu ori interfețe de rețea, folosind diversele standarde, protocoale și tehnologii Internet actuale (Kammann, Strang & Wendlandt, 2001; Schiller, 2003).

Cu toate că interacțiunea cu utilizatorul prin intermediul tastaturii este, în majoritatea cazurilor, limitată, aceste dispozitive oferă premisele unei interacțiuni multimodale prin suportul acordat folosirii altor dispozitive de intrare, precum ecranul senzitiv (*touch screen*) – exploatat în conjuncție cu *stylus*-ul (care poate asigura și o interacțiune prin intermediul gesturilor) –, creionul electronic (*pen*) sau microfonul, aceasta din urmă facilitând interacțiunea vocală prin recunoașterea vorbirii (*speech recognition*). Se poate considera faptul că noile dispozitive favorizează o interacțiune mai complexă – apropiată de cea naturală (*natural interaction*) – cu utilizatorul, implicând o paletă senzorială mai largă. De asemenea, unele dispozitive de calcul (precum *Tablet PC*-urile) prezintă afișaje orientate atât orizontal (*landscape*), similare monitoarelor obișnuite, cât și vertical (*portrait*). Unele ecrane pot avea formă pătrată (*square*). Din acest motiv, interfața cu utilizatorul ale aplicațiilor rulate pe acel dispozitiv trebuie proiectată diferit, în funcție de modul de dispunere a ecranului (Buraga, 2005b).

Aceste dispozitive creează rețele *wireless* constituite *ad-hoc*, caracterizate prin atribute ca mobilitatea ridicată a codului, a dispozitivului însuși și a utilizatorilor, eterogenitate la nivel de hardware și/sau software, diferențe majore în ceea ce privește lățimea de bandă și latența comunicației, caracterul relativ precar al legăturilor de comunicație (Grigoraș, 2005; Schiller, 2003). Una dintre problemele importante și de interes care apare este cea a descoperirii (regăsirii) resurselor și serviciilor în cadrul unei rețele *wireless ad-hoc*. Acest aspect nu poate fi deloc neglijat în contextul existenței unor aplicații-utilizator facilitând, de exemplu, constituirea de rețele sociale ubicue (*ubiquitous social networks*) și, deci, oferind suport pentru interacțiuni umane complexe.

Un alt domeniu de interes – și de viitor – este cel al afacerilor electronice (*e-business*-ul), beneficiind tot mai mult de facilitățile acordate utilizatorilor în ceea ce privește accesarea și folosirea în manieră *on-line* a diverselor servicii puse la dispoziție de situri de comerț electronic, bănci virtuale, case de licitații electronice, motoare de căutare etc. (Buraga, 2005a).

Scopul acestui capitol este cel de a descrie o metodă originală de specificare în manieră independentă de platformă a profilului unui dispozitiv fără fir, în vederea constituirii unei infrastructuri computaționale flexibile, aliniată tendințelor actuale de dezvoltare a Web-ului semantic, menită a facilita managementul resurselor disponibile într-o rețea fără fir constituită *ad-hoc*, cu implicații în ceea ce privește o mai bună interacțiune cu utilizatorul.

2. PREZENTARE SUCCINTĂ A WEB-ULUI SEMANTIC

2.1 Preambul

Scopurile originare principale ale spațiului World-Wide Web au fost acelea de a oferi o modalitate de comunicare inter-umană prin partajarea cunoștințelor și posibilitatea exploatarei în manieră distribuită a puterii de calcul a computerelor conectate la Internet.

Spațiul WWW actualmente este compus din pagini (documente) marcate în limbaje precum HTML (*HyperText Markup Language*), WML (*Wireless Markup Language*), SVG (*Scalable Vector Graphics*) – detalii în (Buraga, 2003b; Buraga, 2004b; Geroimenko, 2004; W3C) – conținând informații textuale, grafice și/sau multimedia destinate a fi citite și înțelese esențialmente de către consumatorii umani. Principala activitate a calculatoarelor este cea de a reda aceste conținuturi și nu să le interpreteze într-o manieră automată și autonomă. Informațiile trebuie regăsite într-un mod inteligent și trebuie să poată fi procesate de către mașină. Conform creatorului spațiului WWW, Sir Tim Berners-Lee, *Web-ul semantic* reprezintă o pânză consistentă și logică a tuturor resurselor de pe Web, accentul punându-se pe interpretarea datelor de către mașină și nu pe reprezentarea lor. În cadrul unui scenariu vizionar, prezentat în (Berners-Lee, Hendler & Lassila, 2001), se prefigurează modul cum dispozitive inteligente partajează cunoștințe privitoare la propriile funcționalități și la contextul în care își desfășoară activitatea, utilizând reguli de inferență și meta-date pentru a (re)găsi informațiile solicitate de utilizatorii umani.

Printre dezideratele Web-ului semantic se pot enumera – conform (Berners-Lee, Hendler & Lassila, 2001) și (Davies, Fensel & van Harmelen, 2003):

- asocierea de semantici legăturilor dintre resurse, cu posibilitatea extinderii acestor semantici;
- resursele Web să poată fi extinse și clasificate, pentru aceasta trebuind să fie adoptate specificații conceptuale;
- la nivel programatic, să existe entități capabile să proceseze în manieră inteligentă informațiile și să poată raționa, oferind mașinilor și oamenilor servicii complexe (*e.g.*, căutarea datelor, regăsirea unor tipuri de resurse fizice/logice, monitorizarea activității aplicațiilor, filtrarea informațiilor etc.);
- partajarea de către utilizatori a cunoștințelor, indiferent de modul de stocare și de reprezentare a acestora.

2.2 Niveluri ale Web-ului semantic

Arhitectura Web-ului semantic este una funcțională, deoarece modul de constituire a acesteia se bazează pe specificarea incrementală a unor limbaje, pornind

de la nivelul inferior (i.e. nivelul meta-datelor) și ajungând la nivelurile superioare (e.g., nivelul logic) – a se vedea și figura 1. Limbajele disponibile pe fiecare nivel pot satisface cerințe impuse de diferite tipuri (sau niveluri) de aplicații (Buraga, 2004a; Davies, Fensel & van Harmelen, 2003):

1. *nivelul meta-datelor* pune la dispoziție cadrul general pentru exprimarea unor aserțiuni semantice simple. Modelul oferit conține concepte precum resursă și proprietate, utilizate să exprime meta-informații. Modelul se specifică via limbajul RDF (*Resource Description Framework*) și diverse vocabulare de meta-date precum DCMI (*Dublin Core Metadata Initiative*), RSS (*Rich/RDF Site Summary*), FOAF (*Friend Of A Friend*) etc. – a se vedea (Geroimenko, 2004) și (FOAF);
2. *nivelul schemelor* oferă posibilitatea specificării de ontologii simple (ca, de exemplu, taxonomii) pentru a se putea defini o descriere ierarhică a conceptelor și proprietăților;
3. *nivelul logic* introduce limbaje ontologice mai complexe, capabile a modela ontologii sofisticate. Se dorește astfel constituirea unor servicii (*reasoning services*) pentru Web-ul semantic, de interes în ceea ce privește aplicațiile destinate unor domenii precum *e-business* (e.g., servicii de luare sau asistare în luarea deciziilor, servicii de monitorizare a vânzărilor *on-line* etc.).

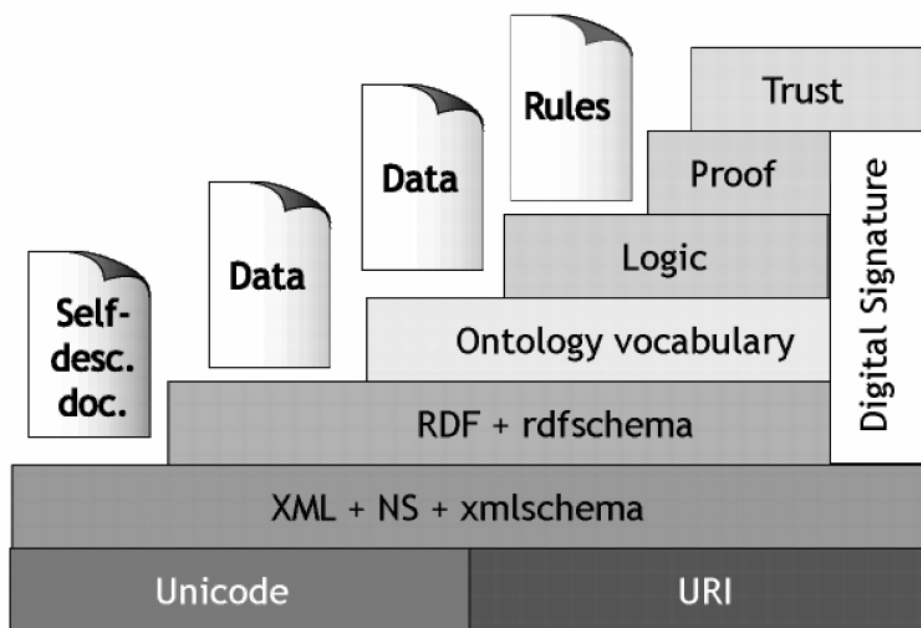


Figura 1. Nivelurile de specificare a Web-ului semantic

La baza acestor niveluri, se află două dintre componentele principale ale Web-ului actual: identificatorii uniformi de resurse URI (*Uniform Resource Identifiers*) (W3C) și meta-limbajul XML (*Extensible Markup Language*) – a se vedea (Bray *et al.*, 2004). Identificatorii uniformi de resurse sunt folosiți pentru localizarea resurselor Web, identificându-le printr-o reprezentare a mecanismului de accesare (via adresa IP sau numele de domeniu Internet, de exemplu) ori referindu-le printr-un nume persistent și unic – detalii în (Buraga, 2004a). Meta-

limbajul XML reflectă o manieră uniformă sintactică, independentă de platformă și de limbajul de programare, de structurare a datelor. Actualmente, se utilizează un număr însemnat de limbaje bazate pe XML pentru marcarea diferitelor informații – a se consulta și (Buraga, 2004a), (Geroimenko, 2004) și (W3C).

Pentru fiecare nivel al arhitecturii stratificate a Web-ului semantic, Consorțiul Web (W3C) a standardizat sau urmează să standardizeze diferite limbaje având drept temelie meta-limbajul XML.

3. SPECIFICAREA PROFILULUI UNUI DISPOZITIV WIRELESS

3.1 Premise

Pentru a asigura o interacțiune corespunzătoare cu aplicațiile rulând pe dispozitive mobile, trebuie luate în primul rând în considerație caracteristicile tehnice, enumerate în cadrul secțiunii 1, ale dispozitivelor fără fir, în vederea definirii unui *profil* al dispozitivelor utilizate (Buraga, 2003a).

În cadrul profilului unui dispozitiv se vor putea specifica, printre altele:

- metodele de acces la Internet – cu fir (*wired*) sau fără fir (*fixed wireless*, *wireless LAN*, *Public Land Mobile Networks*), conform celor prezentate în (Kammann, Strang & Wendlandt, 2001);
- stiva de protocoale folosită (TCP/IP sau X.25);
- tipul conexiunii (*e.g.*, punct-la-punct ori punct-la-multipunct);
- standardele industriale adoptate (*i.e.*, Bluetooth sau IrDA).

Din punctul de vedere al interacțiunii dintre utilizator și dispozitiv, acest profil va putea asigura:

- realizarea unei proiectări independente de dispozitiv a interfeței-utilizator,
- efectuarea unor teste de evaluare a componentelor sau a prototipului de interfață-utilizator a aplicațiilor destinate a fi rulate pe dispozitive (reale ori virtuale),
- existența unui suport „inteligent” oferit de mediile de dezvoltare a interfețelor pentru realizarea rapidă de prototipuri de dispozitive, pentru proiectarea și implementarea interfeței-utilizator destinate unor tipuri (artefacte) de aplicații exploatabile pe dispozitive mobile sau pentru efectuarea unor teste de utilizabilitate (la nivel formal sau informal).

În vederea specificării într-o manieră independentă de platformă (configurație *hardware*, sistem de operare, mediu și limbaj de programare etc.) a profilului dispozitivului, soluția propusă se bazează pe meta-limbajul XML (Bray *et al.*, 2004), în general, și pe cadrul de descriere a resurselor RDF (Buraga, 2004a; Geroimenko, 2004; W3C), în special.

3.2 Specificarea caracteristicilor

Fiecărui dispozitiv, privit ca o resursă Web care poate fi identificată printr-un identificator uniform de resursă, îi vom asocia via construcții RDF diverse meta-date reprezentând caracteristicile (proprietățile) acestuia. Suplimentar, se vor putea specifica detalii referitoare la platforma/platformele de calcul folosite: sistem(e) de operare, limbaj(e) de programare suportate, medii și biblioteci de dezvoltare a aplicațiilor etc. Toate aceste caracteristici vor fi descrise prin intermediul marcajelor XML.

Identificând dispozitivul printr-un URI, avem posibilitatea să-i asociem fie un URN (*Uniform Resource Name*) – a cărui valoare va putea desemna, de exemplu, un identificator unic stabilit de producător –, fie un URL (*Uniform Resource Locator*) – care va reprezenta localizarea aceluși dispozitiv în cadrul spațiului World-Wide Web sau adresa unui serviciu Web de localizare a resurselor pe baza unor criterii specificate (Grigoraș, 2005). Această flexibilitate este utilă în contextul unui mediu ubicuu compus din dispozitive inteligente eterogene ori în cel al unui mediu computațional distribuit pe scară largă de tip Grid.

Pentru anumite tipuri de dispozitive (*e.g.*, *Tablet PC*), vor trebui precizate anumite detalii referitoare la mijloacele multiple de interacțiune (tastatură, mouse, *touch screen*, *stylus*) și la caracteristicile de afișare: rezoluție, ecran versatil cu orientare *landscape/portrait*, sensibilitate tactilă etc.

Un alt aspect esențial este cel privind autonomia dispozitivului. Vor trebui reținute detalii referitoare la calitatea serviciilor (*QoS - Quality of Service*) vizând:

- energia consumată (de exemplu, starea de încărcare a bateriilor proprii și diversele restricții de utilizare în contextul folosirii unor aplicații necesitând o energie electrică substanțială – *e.g.*, rularea unui *player* multimedia),
- conectivitatea la Internet (de exemplu, capacitatea de transfer via o legătură Internet stabilită prin portul infraroșu sau intervalul temporal al stabilirii unei conexiuni optime la Internet într-o rețea Bluetooth),
- interacțiunea cu utilizatorul (pot fi luate în considerație diverse deprinderi ale utilizatorului, precum gradul de folosire a tastaturii în detrimentul mouse-ului sau *stylus*-ului, acuitatea vizuală, abilitățile loco-motorii, preferințele lingvistice etc.).

De asemenea, acest profil va putea include informații privitoare la profilul Bluetooth de acces la rețea, în cazul dispozitivelor care au suport pentru această tehnologie. Astfel, se vor putea memora date privitoare la profilurile de acces prin port serial (*serial port profile*), prin modem (*dial-up networking profile*) sau via rețeaua locală (*local area network access profile*) – a se vedea (Kammann, Strang & Wendlandt, 2001).

Folosind aserțiuni RDF, se oferă suport pentru nivelul de meta-date și cel ontologic în vederea clasificării dispozitivelor și a exprimării relațiilor dintre dispozitivele mobile. Acest aspect devine foarte important în contextul utilizării acestor specificații în cadrul Web-ului semantic.

Un prim exemplu

Vom considera, drept exemplu prezentat în (Buraga, 2005), un profil pentru un dispozitiv *Tablet PC* din categoria dispozitivelor prezentând caracteristici tehnice avansate. Acest profil, exprimat în XML/RDF, va putea avea următoarea structură:

```
<?xml version="1.0" ?>
<rdf:Description
  rdf:about="urn:busaco:TabletPC">
  <!-- Informatii generale -->
  <c:device type="TabletPC" wid="...">
    <c:name>...</c:name>
    <c:producer>...</c:producer>
    <c:owner>...</c:owner>
    ...
  </c:device>
  <!-- Caracteristicile tehnice -->
  <c:hardware>
    <c:processor>...</c:processor>
    <c:memory>...</c:memory>
    <c:devices>
      <rdf:Bag>
        <rdf:li>
          <c:connection port="IrDA">...</c:connection>
          <c:screen>
            <c:resolution type="1024x768" />
            <c:format>
              <rdf:Bag>
                <rdf:li>landscape</rdf:li>
                <rdf:li>portrait</rdf:li>
              </rdf:Bag>
            </c:format>
          </c:screen>
        </rdf:li>
        ...
      </rdf:Bag>
    </c:devices>
    <c:power>...</c:power>
  </c:hardware>
  <!-- Platforma -->
  <c:platform>
    <c:os version="...">...</c:os>
    <c:framework>...</c:framework>
  </c:platform>
  <!-- Preferintele utilizatorilor -->
  <c:preferences>
    <rdf:Bag>
      <rdf:li>
        <rdf:Description rdf:about="#User">
          <c:input>...</c:input>
        </rdf:Description>
      </rdf:li>
    </rdf:Bag>
  </c:preferences>
</rdf:Description>
```

```

        <c:output>...</c:output>
    </rdf:Description>
</rdf:li>
...
</rdf:Bag>
</c:preferences>
</rdf:Description>

```

Spațiile de nume *rdf* și *c* desemnează construcțiile RDF, respectiv meta-datele asociate dispozitivului.

Un profil de dispozitiv va trebui să includă detalii privitoare la caracteristicile *hardware*, la platforma/platformele utilizate și la diverse preferințe specifice (fie specificând parametri *QoS*, fie preferințe generale și/sau implicite ale utilizatorului), conform fiecărui tip de dispozitiv în parte.

De asemenea, considerând că fiecare dispozitiv fără fir implică folosirea unui anumit număr și tipuri de resurse, va trebui utilizat un limbaj de modelare a meta-datelor privitoare la aceste resurse. Pentru aceasta se poate recurge la limbajul *XFiles* (Buraga, 2002). Fiecare proprietate a unui resurse (considerată drept fișier) va putea fi specificată prin intermediul unor construcții sintactice *XFiles* (elemente și atribute).

Un al doilea exemplu

Exemplul de mai jos recurge la aserțiuni RDF care includ elemente *XFiles* pentru specificarea de alternative pentru execuția de la distanță a unui program de citire a știrilor RSS (*Rich Site Summary*).

```

<rdf:RDF>
  <!-- o descriere RDF a unui fisier RSS -->
  <rdf:Description rdf:about="http://rowd.org/news.rss">
    <xf:Properties>
      <!-- diverse metadata asociate -->
      <xf:Type xf:mime="application/xml+rss">ordinary</xf:Type>
      <xf:Owner>busaco</xf:Owner>
      <!-- aplicatia care va prelucra continutul
            documentului RSS -->
      <xf:Parser>
        <rdf:Alt>
          <!-- vor exista mai multe posibilitati
                (utilizarea unor programe,
                 disponibile pe platforme diferite) -->
          <rdf:li>
            <rdf:Description
              rdf:about="file:///usr/local/Firefox/firefox">
              <xf:Type xf:mime="application/executable">
                Ordinary
              </xf:Type>
              <xf:Location xf:dns="localhost">
                127.0.0.1
              </xf:Location>
            </rdf:Description>
          </rdf:li>
          ...

```

```
        </rdf:Alt>
      </xf:Parser>
    </xf:Properties>
  </rdf:Description>
</rdf:RDF>
```

Spațiul de nume *xf* desemnează construcțiile puse la dispoziție de limbajul *XFiles*.

3.3 Avantaje

Descrierile de meta-date vor putea fi folosite pentru a exprima identitatea unui anumit dispozitiv sau a serviciilor pe care acesta le pune la dispoziție, în manieră publică ori privată, altor dispozitive aflate în aceeași rețea constituită *ad-hoc* sau în alte rețele. Folosind aceste meta-date, anumite aplicații vor putea realiza un management „inteligent” al resurselor pe care le posedă un dispozitiv mobil sau grup de dispozitive și, suplimentar, vor putea regăsi anumite (tipuri de) resurse în scopul de a le solicita în vederea alocării sau prelucrării, fie la nivel de sistem, fie la nivel de aplicație-utilizator.

Având asociate meta-date precum cele descrise în exemplele anterioare, un dispozitiv (ori un anumit serviciu pe care îl pune la dispoziție) poate fi accesat de la distanță, de alte dispozitive similare și/sau de calculatoare convenționale, conform unor criterii privitoare la performanță, accesibilitate, securitate, conectivitate etc., într-o manieră similară celei descrise în lucrările (Grigoraș, 2005), (Kammann, Strang & Wendlandt, 2001) și (Lai *et al.*, 2004).

Astfel, serviciile de numire și regăsire a serviciilor din cadrul unei rețele fără fir constituite *ad-hoc* vor putea avea asociate descrieri semantice, facilitând – printre altele:

- interogarea (*querying*),
- descoperirea (*discovering*),
- rutarea datelor (*routing*),
- găsirea compatibilității (*matchmaking*) dintre servicii, dispozitive și/sau utilizatori.

4. PROPUNERE DE IMPLEMENTARE

Pentru managementul unificat, independent de platformă, al profilurilor dispozitivelor formând o rețea fără fir constituită *ad-hoc*, propunem folosirea unui sistem de multi-agenți urmărind liniile expuse în (Buraga, 2004c). Agenții software sunt sisteme informaționale care se comportă asemeni unor entități într-o manieră autonomă, execută diverse acțiuni având un anumit nivel de reacție și etalează atribute precum învățarea, cooperarea și mobilitatea, asistând utilizatorii în activitățile acestora (Buraga, 2003a; Buraga, 2004a).

4.1 Arhitectură conceptuală

Sistemul de agenți va fi compus din *agenți mediatori* (agenți de achiziție de informații) care vor facilita interacțiunea dintre utilizator și aplicațiile executate la distanță via servicii Web. Pentru fiecare aplicație disponibilă pe un anumit sistem (e.g., calculatorul de la birou, cel de acasă, calculatorul portabil, *smartphone* etc.), va putea fi dezvoltat un *serviciu Web* rulat pe acel calculator care va prelua de la agenți *input*-ul utilizatorului și va trimite agenților *output*-ul aplicației prin intermediul unui limbaj bazat pe XML. De asemenea, mediul va include și diverși *asistenți digitali personali* care vor recomanda utilizatorului folosirea unor servicii Web corespunzătoare tipurilor aplicațiilor dorite a fi rulate, în funcție de anumite criterii (e.g., viteză de execuție, viteză de transfer a datelor prin Internet, preferințe etc.).

Agenții compunând mediul multi-agent propus trebuie să ofere următoarele servicii minimale (Buraga, 2004c):

- generarea profilului al utilizatorului (se au în vedere reprezentarea modelului mental și a celui cognitiv, plus modelarea cunoștințelor utilizatorului) – soluția adoptată trebuie să recurgă la limbaje de reprezentare bazate pe XML pentru a facilita interacțiunea între agenți sau cea dintre agenți și serviciile Web corespunzătoare aplicațiilor rulate. De asemenea, pentru stocarea modelului utilizatorului se poate recurge la diverse limbaje de reprezentare a cunoștințelor, limbaje care vor putea fi folosite și drept limbaje de comunicare între agenți (*agent communication languages*). Soluția propusă – detaliată în (Buraga & Alboai, 2003) și (Buraga & Alboai, 2005) – va recurge la reprezentări bazate pe XML, meta-datele referitoare la utilizator putând fi exprimate prin construcții RDF, iar conceptele vehiculate via OWL (*Web Ontology Language*) (Geroimenko, 2004; W3C).
- constituirea profilului dispozitivului – acest aspect devine foarte important dacă se intenționează realizarea de interfețe independente de periferic (Buraga, 2005b).
- transformarea datelor XML în alte tipuri de *output*-uri via foi de stiluri XSL (*Extensible Stylesheet Language*) și convertirea *input*-ului utilizatorului într-un format XML (acest serviciu prezintă o importanță deosebită mai ales în contextul unei interacțiuni multimodale).
- posibilitatea migrării odată cu utilizatorul pe diverse alte dispozitive sau calculatoare pe care acesta le poate utiliza, în vederea constituirii unui mediu de operare (interacțiune) consistent și familiar.

4.2 Utilizări

Domeniile de utilizare ale unui astfel de sistem sunt multiple. Modelul de specificare independentă de platformă, de nivel înalt, a profilului unui dispozitiv, poate servi ca bază pentru crearea unei infrastructuri computaționale compuse din agenți oferind servicii de bază pentru aplicațiile destinate a fi rulate în cadrul Internetului fără fir. Proiectanții și dezvoltatorii vor putea beneficia de

avantajele tehnologiilor Web-ului semantic pentru a descrie, clasifica, regăsi și media resurse (data și servicii), în manieră inteligentă.

Una dintre utilizările pe termen scurt ar putea fi în domeniul comunicării inter-personale între utilizatori, constituindu-se rețele sociale digitale fără fir. Relațiile dintre utilizatori pot fi specificate la nivel de aplicație grație unor vocabulare precum FOAF (*Friend Of A Friend*), iar cele între resurse și dispozitive prin intermediul limbajelor prezentate în secțiunea 3 a lucrării de față. O posibilă soluție de implementare este propusă în (Serea, 2005).

O altă utilizare practică, beneficiind de avantajele oferite de sistemele multi-agent, este cea în domeniul *e-learning*, mediul educațional putând fi accesat via dispozitive fără fir – pentru o serie de detalii, se pot consulta lucrări precum (Aroyo & Dicheva, 2004) și (Lai *et al.*, 2004). Similar, se pot imagina aplicații utile și în domeniul *e-business*.

5. CONCLUZII

Capitolul a detaliat o serie de considerații privitoare la specificarea în mod independent de platformă a profilului unui dispozitiv fără fir în vederea constituirii unei infrastructuri computaționale bazată pe tehnologiile Web-ului semantic. O astfel de abordare permite aplicațiilor – ca de exemplu, agenților inteligenți (Buraga, 2003a; Buraga, 2004a) – să efectueze căutări de (clase de) dispozitive pe baza meta-datelor asociate (vizând capacități computaționale, calități ale serviciilor, preferințe etc.). Modelul propus poate fi utilizat în conjuncție cu specificațiile CC/PP (*Composite Capability/Preference Profiles*) (W3C) – descriind proprietăților dispozitivelor și preferințelor utilizatorilor – și P3P (*Privacy Preferences Project*) (W3C) – referindu-se la maniera de declarare a regulilor de acces privat la resursele Web. Cercetările ulterioare se vor concentra, beneficiind de un posibil sprijin al industriei de profil, asupra specificării la nivel ontologic a profilului dispozitivelor mobile și a relațiilor stabilite între diverse instanțe ale acestora.

Referințe

- Aroyo, L., Dicheva D., „The New Challenges for E-learning: The Educational Semantic Web”, *Educational Technology and Society*, 7 (4), 2004
- Berners-Lee, T. , Hendler, J. , Lassila O., „The Semantic Web”, *Scientific American*, 5, 2001
- Bray, T. *et al.* (eds.), *Extensible Markup Language 1.0 (Third Edition)*, W3C Recommendation, Boston, 2004: <http://www.w3.org/TR/REC-xml>
- Buraga S., „A Model for Accessing Resources of the Distributed File Systems”, în Grigoraș, D. *et al.* (eds.), *Advanced Environments, Tools and Applications for Cluster Computing, Lecture Notes in Computer Science – LNCS 2326*, Springer Verlag, 2002
- Buraga S., „Suportul pentru implementare”, în Pribeanu, C. (ed.), *Introducere în interacțiunea om-calculator*, Matrix Rom, București, 2003

- Buraga, S. (coord.), *Aplicații Web la cheie*, Polirom, Iași, 2003:
<http://www.infoiasi.ro/~phpapps/>
- Buraga S., Alboai, L., Alboai, S., „The Use of XML Technologies for Exchanging Information within a Multi-Agent System”, *International Scientific Journal of Computing*, 2 (3), 2003
- Buraga, S., *Semantic Web. Fundamente și aplicații*, Matrix Rom, București, 2004:
<http://www.infoiasi.ro/~sweb/>
- Buraga, S. (coord.), *Situri Web la cheie. Soluții profesionale de implementare*, Polirom, Iași, 2004:
<http://www.infoiasi.ro/~busaco/books/webapps/>
- Buraga, S., „Asigurarea interacțiunii om-calculator prin intermediul instrumentelor Web”, în Trăușan-Matu, Ș., Pribeanu, C. (eds.), *Interacțiune om-calculator – Volumul de lucrări ale primei Conferințe Naționale de Interacțiune Om-Calculator (RoCHI 2004)*, Editura Printech, București, 2004
- Buraga, S., *Proiectarea siturilor Web* (ediția a doua), Polirom, Iași, 2005:
<http://www.infoiasi.ro/~design/>
- Buraga, S., „Aspecte ale proiectării și implementării interfețelor-utilizator destinate dispozitivelor mobile în contextul Web-ului semantic”, în Pitariu, H., Buraga, S. (eds.), *Interacțiune om-calculator – Volumul de lucrări ale celei de-a doua Conferințe Naționale de Interacțiune Om-Calculator (RoCHI 2005)*, Editura ASCR, Cluj-Napoca, 2005
- Buraga, S., Alboai, L., Alboai, S., „An XML/RDF-based Proposal to Exchange Information within a Multi-Agent System”, în Grigoraș, D., Nicolau, A. (eds.), *Concurrent Information Processing and Computing, NATO Science Series*, vol. 195, IOS Press, 2005
- Davies, J., Fensel, D., van Harmelen, F. (eds.), *Towards the Semantic Web*, John Wiley & Sons, 2003
- Geroimenko, V., *Dictionary of XML Technologies and the Semantic Web*, Springer Verlag, 2004
- Grigoraș, D., „Service-Oriented Naming Scheme for Wireless Ad Hoc Networks”, în Grigoraș, D., Nicolau, A. (eds.), *Concurrent Information Processing and Computing, NATO Science Series*, vol. 195, IOS Press, 2005
- Kammann, J., Strang, T., Wendlandt, K., „Mobile Services over Short Range Communication”, *Workshop on Commercial Radio Sensors and Communication Techniques*, Technical University of Linz, August 2001
- Lai, A. et al., „Improving Web Browsing on Wireless PDAs Using Thin-Client Computing”, *Proceedings of WWW 2004*, ACM Press, 2004
- Pratt, J., „Developing Screen Orientation-Aware Applications for Windows Mobile-Based Pocket PCs”, *Proceedings of TechEd 2004*, Microsoft Press, Redmond, 2004
- Schiller, J., *Mobile Communications*, Addison-Wesley, 2003
- Serea, M., „MobiNET. Rețele sociale digitale – o nouă abordare din perspectiva interacțiunii om-calculator”, în Pitariu, H., Buraga, S. (eds.), *Interacțiune om-calculator – Volumul de lucrări ale celei de-a doua Conferințe Naționale de Interacțiune Om-Calculator (RoCHI 2005)*, Editura ASCR, Cluj-Napoca, 2005
- ***, FOAF (Friend Of A Friend) Specification: <http://rdfweb.org/>
- ***, World-Wide Web Consortium, Boston, 2005: <http://www.w3.org/>

Capitolul 4

OUR PHOTOS: DESCRIEREA, REGĂSIREA ȘI VIZUALIZAREA FOTOGRAFIILOR ÎN CADRUL UNUI ALBUM ONLINE PARTAJAT – O ABORDARE BAZATĂ PE RDF ȘI ASP.NET

Sergiu Sebastian Tauciu

Absolvent al Facultății de Informatică, Universitatea Alexandru Ioan Cuza, Iași
stauciu@yahoo.com, <http://sebicu.lx.ro/>

Rezumat. În acest capitol vom prezenta aspecte legate de conceperea și implementarea unei aplicații Web pentru gestionarea albumelor partajate de fotografii. Vom începe cu prezentarea motivelor care au stat la baza acestei abordări inovatoare și vom continua cu prezentarea rolului pe care tehnologiile XML și XML/RDF le-au jucat în implementarea aplicației.

Cuvinte-cheie: fotografie, XML, RDF, adnotare semantică, motor de căutare.

1. INTRODUCERE

Deși și-a demonstrat deja utilitatea într-o arie largă de domenii, Internetul mai are încă un imens potențial de dezvoltare care așteaptă să fie valorificat. De aceea, direcțiile posibile de cercetare sunt, practic, nenumărate, iar utilizările viitoare și nivelul de dezvoltare al tehnologiilor Web sunt greu de intuit.

Unul dintre domeniile în care Internetul a avut impactul cel mai puternic, în special din punct de vedere comercial (bucurându-se, din acest motiv, de o atenție deosebită și în privința investițiilor în cercetare și dezvoltare), este piața serviciilor. Începând cu primul serviciu de poștă electronică și continuând cu o serie de alte tipuri de servicii de comunicare (în timp real, audio, video), de divertisment, de știri, publicitare și multe altele, Internetul a revoluționat acest domeniu, găsind noi și noi posibilități de dezvoltare.

O categorie de servicii de actualitate, a cărei apariții a fost făcută posibilă de implementarea noilor tehnologii Web, este cea a comunităților virtuale. Acest

tip de serviciu permite inter-relaționarea grupurilor de persoane pe baza unor interese sau caracteristici comune. Plecând de la această nevoie de relaționare *online* sesizată deja în societatea actuală, și analizând un alt serviciu de comunicare (de data aceasta vizuală), și anume cel al albumelor de fotografii *online*, am ajuns la ideea prezentului proiect: crearea și menținerea de albume de fotografii pentru comunități (reale sau virtuale) de persoane.

Aplicație Web pentru gestionarea albumelor comune de fotografii, *Our Photos* renunță la varianta clasică de abordare, în care fiecare utilizator înregistrat își poate crea un număr de albume proprii, și adoptă o abordare orientată-album (sau orientată-comunitate): un număr de utilizatori împart un album comun, în care pot posta fotografii și impresii legate de tematica/ocazia creării albumului. Pentru realizarea unei personalizări reale, *Our Photos* oferă utilizatorilor posibilitatea adnotării semantice a fotografiilor încărcate, iar motorul avansat de căutare bazat pe tehnologii XML/RDF permite regăsirea mult mai ușoară a fotografiilor dorite.

2. SUPORTUL SEMANTIC PENTRU CĂUTARE

2.1 Fișierele de descriere RDF

Tehnologia RDF

Pentru descrierea detaliilor fotografiilor am folosit tehnologia RDF (*Resource Description Framework* – descrisă de McBride, Manola și Miller în *RDF Primer*), limbajul standard de reprezentare a resurselor în spațiul World Wide Web.

Cu ajutorul acestei tehnologii, putem reprezenta orice colecție de resurse sub forma unui graf în care nodurile reprezintă resurse și proprietăți ale acestora, iar muchiile reprezintă relații între aceste noduri (predicate).

Proiectarea fișierului de descriere

Pentru a putea reprezenta într-un mod cât mai eficient toate proprietățile considerate pentru o fotografie, le-am grupat pe trei mari categorii:

- *Context* – proprietăți legate de contextul în care a fost realizată fotografia: anotimp, moment al zilei, vremea, ocazia realizării fotografiei.
- *Content* – proprietăți legate de conținutul și semnificația fotografiei: titlul, tipul, cuvinte cheie, persoane în fotografie, autor etc.
- *Technical* – caracteristici tehnice ale fotografiei, extrase din meta-datele fișierului de stocare; am considerat acele proprietăți mai relevante pentru căutarea de tipuri de fotografii: data realizării, camera folosită, timpul de expunere, diafragma, viteza ISO, *blitz*-ul și rezoluția.

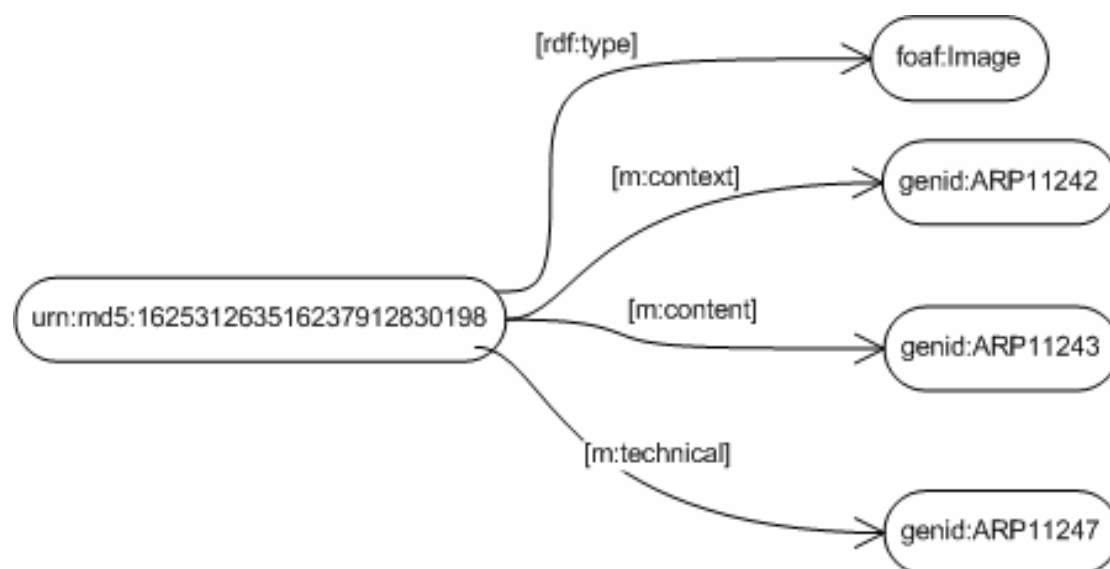


Figura 1 Reprezentarea proprietăților „context”, „content” și „technical” pentru o imagine

Cele trei categorii au determinat trei „containere” (recipiente) în structura RDF/XML, fiecare container conținând proprietățile specifice. Aceste containere sunt construite ca niște resurse (noduri) anonime, legate la fotografia descrisă prin predicatele corespunzătoare: „m:content”, „m:context”, respectiv „m:technical”. Proprietățile conținute de fiecare container vor fi reprezentate astfel: predicatul (denumirea proprietății) va fi o muchie având predicatul ca etichetă, iar valoarea luată de acea proprietate va genera un nod frunză. Muchia va lega nodul la containerul corespunzător.

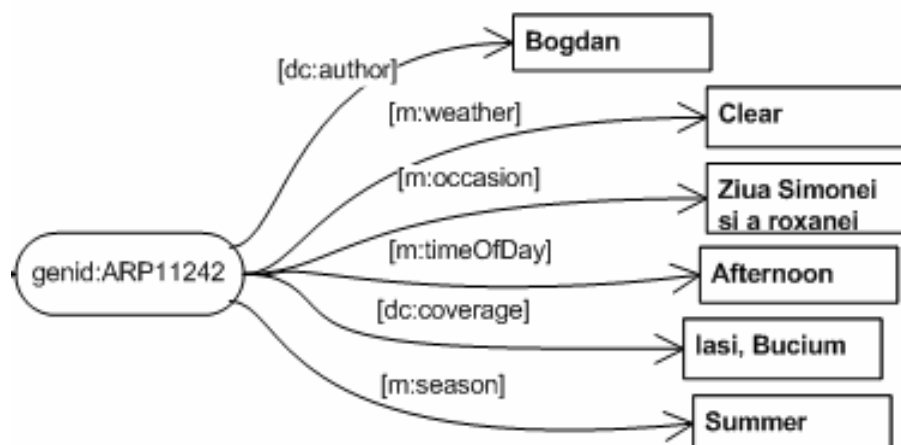


Figura 2. Reprezentarea proprietăților de context

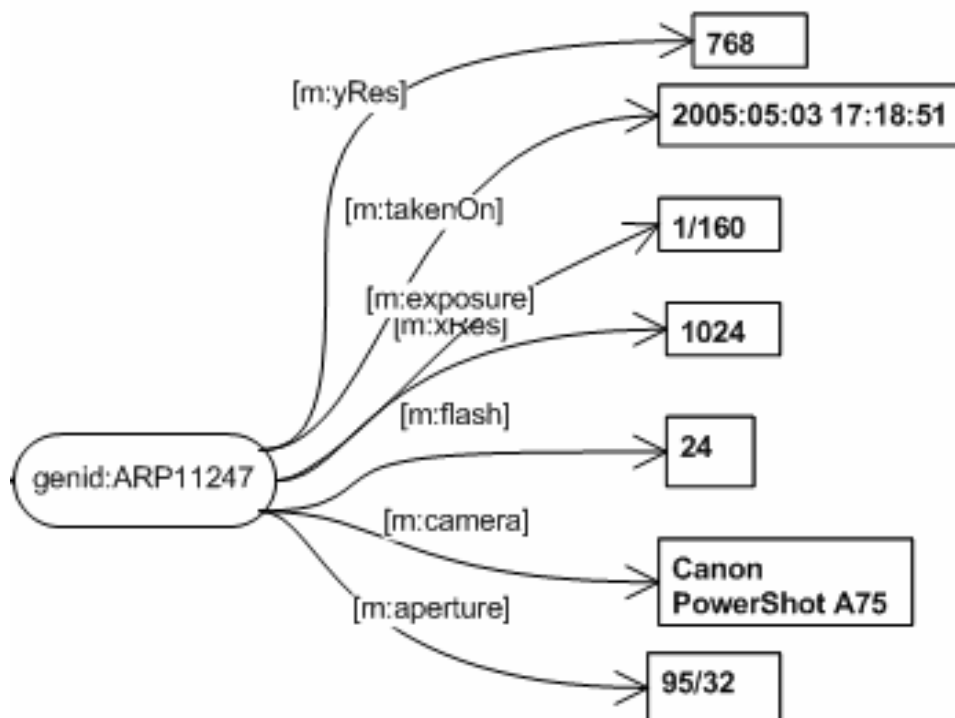


Figura 3. Reprezentarea proprietăților „technical”

Excepție la acest model de construcție sunt cuvintele cheie și persoanele prezente în fotografie. Putând apărea în număr mai mare de 1 pentru fiecare fotografie, aceste elemente au fost organizate în alte două containere în cadrul structurii *Content*, respectiv *Keywords* și *Depicts*.

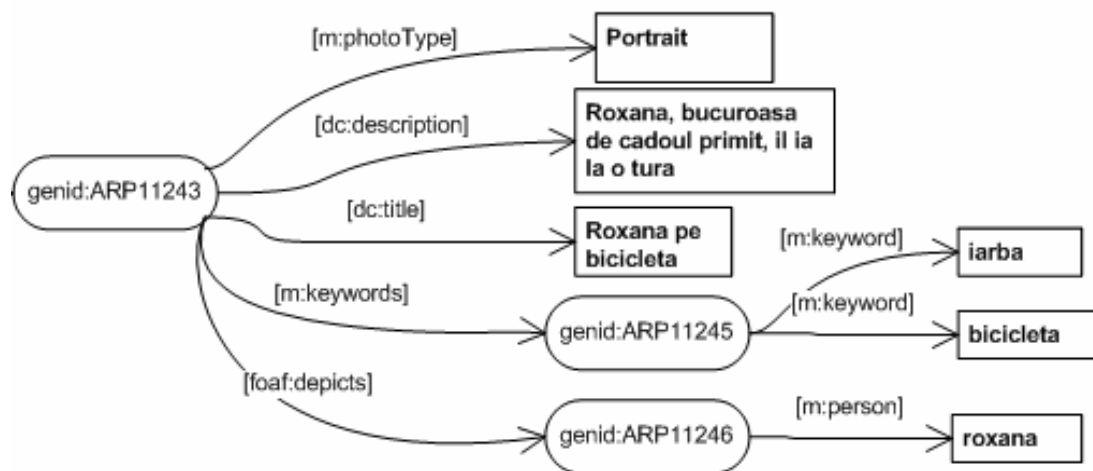


Figura 4. Reprezentarea containerului „content”

Mențiuni:

Pe lângă toate aceste proprietăți, un fișier de descriere mai conține și tipul MIME al fișierului descris, același pentru toate fișierele, și anume `http://xmlns.com/foaf/0.1/Image` (după cum se poate observa în figura 1).

Întrucât fișierele XML/RDF sunt în fișiere separate de resursele descrise, aceste resurse (imaginile) trebuie identificate în mod unic, cu ajutorul unui URI (*Uniform Resource Identifier*). Am construit acest URI pe baza fișierului imaginii, cu ajutorul unui algoritm MD5, care generează un cod unic pentru fiecare flux de caractere primit la intrare. Astfel, fiecare fișier poate fi identificat în mod unic, și un fișier de descriere se poate referi la un singur fișier imagine:

```
<rdf:Description
  rdf:about="urn:md5:208124566719892772926951571692247823477">
```

Această metodă de identificare poate fi luată în considerare ca o alternativă la metoda clasică:

```
<rdf:Description
  rdf:about="www3.infoiasi.ro/~sebicu/ourphotos/dscn2005.jpg">
```

Iată în continuare și fișierul RDF corespunzător grafului prezentat în imaginile de mai sus (acest fișier a fost generat de aplicație):

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:m="http://www3.infoiasi.ro/~sebicu/Licenta/termeni/"
xmlns="tag:default/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description
    rdf:about="urn:md5:2541781321761091912401125513318450112725569">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Image" />
    <m:context rdf:parseType="Resource">
      <dc:author>Bogdan</dc:author>
      <m:weather>Clear</m:weather>
      <m:occasion>Ziua Simonei si a Roxanei</m:occasion>
      <m:timeOfDay>Afternoon</m:timeOfDay>
      <dc:coverage>Iasi, Bucium</dc:coverage>
      <m:season>Summer</m:season>
    </m:context>
    <m:content rdf:parseType="Resource">
      <m:photoType>Portrait</m:photoType>
      <dc:description>Roxana, bucuroasa de cadoul primit,
        il ia la o tura...</dc:description>
      <dc:title>Roxana pe bicicleta</dc:title>
      <m:keywords rdf:parseType="Resource">
        <m:keyword>iarba</m:keyword>
        <m:keyword>bicileta</m:keyword>
        <m:keyword>sport</m:keyword>
      </m:keywords>
      <foaf:depicts rdf:parseType="Resource">
        <m:person>roxana</m:person>
      </foaf:depicts>
    </m:content>
    <m:technical rdf:parseType="Resource">
      <m:YRes>768</m:YRes>
      <m:takenOn>2005:05:03 17:18:51</m:takenOn>
      <m:Exposure>1/160</m:Exposure>
```

```

    <m:XRes>1024</m:XRes>
    <m:Flash>24</m:Flash>
    <m:Camera>Canon PowerShot A75</m:Camera>
    <m:ISO>Unknown</m:ISO>
    <m:Aperture>95/32</m:Aperture>
  </m:technical>
</rdf:Description>
</rdf:RDF>

```

Trebuie menționat că fișierul este scris în format de descriere RDF/XML *Stripped Syntax*, o sintaxă RDF prescurtată, bazată pe nivele de imbricare, realizată cu ajutorul notației XML. Mai multe detalii se pot găsi la adresa <http://www.w3.org/2001/10/stripes/>.

2.2 Fișierele de index XML

Procesarea detaliilor RDF pentru toate fotografiile incluse într-o căutare poate deveni un proces costisitor, mai ales atunci când numărul fotografiilor crește. De aceea este necesară optimizarea procesului de căutare, lucru realizabil cu ajutorul indexării căutărilor.

Am implementat această metodă cu ajutorul unor fișiere de index XML, care conțin înregistrări referitoare la căutările deja efectuate. Astfel, pentru unele cuvinte (cele existente în index în urma căutărilor anterioare) se evită căutarea prin detaliile tuturor fotografiilor, rezultatul putând fi obținut doar prin consultarea indexului.

Pentru fiecare cuvânt indexat, fișierul index trebuie să conțină id-urile fotografiilor în detaliile cărora apare acel cuvânt. Astfel, la o nouă căutare pentru acel cuvânt, tot ce trebuie de făcut este să căutăm cuvântul în index și să aflăm id-urile corespunzătoare, pe care le vom folosi mai apoi la construirea setului de fotografii.

Pentru conceperea fișierului index, am mai ținut seama de următoarele aspecte:

- pentru fiecare căutare, vom dori o ordonare a fotografiilor în funcție de relevanța acestora, adică de gradul de potrivire a cuvintelor din căutare peste detaliile fotografiei;
- un cuvânt se poate potrivi parțial sau complet peste un element al fotografiei, fapt ce influențează relevanța;
- un cuvânt poate apărea sub forma diferitelor elemente ale fotografiei. O fotografie care are cuvântul „iarna” în titlu are o relevanță diferită de una care are „iarna” ca și cuvânt cheie. Vrem să facem deosebirea între locurile de potrivire ale unui cuvânt în detaliile fotografiei.

După considerarea și analiza mai multor variante de abordare, am ales-o pe următoarea:

Indexul va fi structurat pe cuvinte, iar fiecare cuvânt va conține 0 sau mai multe „intrări”, fiecare „intrare” („*entry*”) având exact un element „*appears_as*” („apare ca”) și cel puțin un element „*photo*”. Un element „*photo*” va conține atributul „*phid*” (*photo ID*) și atributul „*full_match*”, care ne spune dacă potrivirea e totală sau parțială:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<index>
  <search word="iarna">
    <entry>
      <appears_as>http://purl.org/dc/elements/1.1/title</appears_as>
      <appears_in>
        <photo phid="98" full_match="False" />
      </appears_in>
    </entry>
    <entry>
      <appears_as>
        http://www3.infoiasi.ro/~sebicu/Licenta/termeni/keyword
      </appears_as>
      <appears_in>
        <photo phid="96" full_match="True" />
        <photo phid="95" full_match="True" />
      </appears_in>
    </entry>
  </search>
  .....
</index>
```

Întrucât aplicația noastră este orientată-album, va exista câte un astfel de fișier index pentru fiecare album, care va ține evidența căutărilor efectuate pe albumul respectiv.

3. ASPECTE PRIVIND IMPLEMENTAREA

Vom vedea în continuare cum folosim fișierele definite mai sus în cadrul unui motor de căutare integrat în aplicație. Am selectat fragmentele care lucrează efectiv cu aceste fișiere și, prin urmare, au avut rolul cel mai important în implementarea procesului de căutare.

Utilități pentru căutare

Elementul central în cadrul motorului de căutare este dat de clasa numită *PhotoFinder*. Un „*finder*” caută în fișierul de index, reține ID-uri de fotografii, apelează metode externe și în final construiește seturi de fotografii ce vor fi folosite de nivelul superior.

Dar înainte de a putea explica modul de funcționare al unui astfel de obiect, este necesar să descriem o serie dintre utilitățile (metode statice) oferite de clasa *SearchUtilities* și folosite de *PhotoFinder*.

1. Metoda *ParsePhoto()*

Funcția *ParsePhoto()* a clasei *SearchUtilities* folosește facilitățile bibliotecii *DriveRDF* pentru a procesa detaliile unei fotografii. Rezultatul este un obiect de tip graf.

```
internal static IRdfGraph ParsePhoto(SessionToken token, Photo photo)
{
    IRdfGraph graph = null;
    string xmlPath = token.MapPath(photo.XmlFullVirtualPath);
    //calea fișierului XML/RDF cu detaliile fotografiei
    if (File.Exists(xmlPath))
    {
        try
        {
            graph = ParseRdfXml(xmlPath);
        }
        catch (ApplicationException exc)
        { //..... }
    }
    else { //..... }
    return graph; //in caz de eroare, se va returna null
}
```

2. Metoda *CreateIndexEntries()*

Această funcție primește la intrare un cuvânt și un set de fotografii și are rolul de a căuta potrivirile aceluia cuvânt în detaliile tuturor fotografiilor și de a genera un mic „index” pentru acel cuvânt și acel set de fotografii.

Am folosit cuvântul *index* tocmai pentru că structura generată simulează fișierul de index folosit și descris anterior. Această alegere este firească, dacă ne gândim că *PhotoFinder*-ul are rolul de a actualiza fișierul index pe baza rezultatelor căutării. El va avea nevoie de o structură cât mai apropiată fișierului index, pentru ușurința procesării.

Structura folosită va fi o listă (*ArrayList*) de elemente de tip *IndexEntry*.

DescribeAndRetrieve::IndexEntry
+word : string
+appears_as : string
+part_of : bool
+photold : int
+IndexEntry()
+IndexEntry()

Elementele de tip *IndexEntry* sunt asemănătoare semantic elementelor XML din index, cu deosebirea că un singur element *appears_as* nu **conține** un set de ID-uri, ci **corespunde** unui set de ID-uri. Setul de înregistrări este liniar, asemănător unei tabele, în timp ce elementele XML sunt organizate, cum era firesc,

arborescent. Simularea structurii arborescente se va realiza asemănător structurării în cazul bazelor de date, și anume prin ordonare. Ordonând elementele după „*word*” și „*appears_as*”, *PhotoFinderul* va ști că atunci când, iterând prin colecție, descoperă că valoarea „*appears_as*” s-a schimbat, înseamnă că trebuie să introducă un nou element de tip „*entry*”.

Să luăm exemplul de fișier index considerat anterior:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<index>
  <search word="iarna">
    <entry>
      <appears_as>http://purl.org/dc/elements/1.1/title</appears_as>
      <appears_in>
        <photo phid="98" full_match="False" />
      </appears_in>
    </entry>
    <entry>
      <appears_as>
        http://www3.infoiasi.ro/~sebicu/Licenta/termeni/keyword
      </appears_as>
      <appears_in>
        <photo phid="96" full_match="True" />
        <photo phid="95" full_match="True" />
      </appears_in>
    </entry>
  </search>
  .....
</index>
```

Acest index ar putea fi generat în urma procesării unei structuri formate din următoarele 3 elemente *IndexEntry*:

	<i>word</i>	<i>appears_as</i>	<i>part_of</i>	<i>photoId</i>
<i>Entry1:</i>	"iarna"	"title"	<i>true</i>	98
<i>Entry2:</i>	"iarna"	"keyword"	<i>false</i>	96
<i>Entry3:</i>	"iarna"	"keyword"	<i>false</i>	95

Subliniem că obiectele de tip *IndexEntry* sunt obținute în urma procesării detaliilor fotografiilor și introduse în ordinea găsirii lor. Obiectul *PhotoFinder* este cel care se va ocupa de sortarea și procesarea lor.

Vom prezenta în continuare funcționarea metodei *CreateIndexEntries()*.

Inițializăm structura pentru rezultate:

```
ArrayList entries = new ArrayList();
```

Declarăm o variabilă de tip *IrdGraph*, care va conține graful detaliilor fotografiei curente:

```
IRdfGraph sgraph;
```

Pentru fiecare fotografie din setul dat, generăm arborele RDF și identificăm spațiile de nume folosite.

```
foreach(Photo photo in photos)
{
    sgraph = SearchUtilities.ParsePhoto(token, photo);
    string foafNs = sgraph.NameSpaces["xmlns:foaf"];
    string rdfNs = sgraph.NameSpaces["xmlns:rdf"];
    string dcNs = sgraph.NameSpaces["xmlns:dc"];
    string mineNs = sgraph.NameSpaces["xmlns:m"];
}
```

Amintim că graful RDF conține informațiile efective legate de detaliile fotografiei în nodurile sale frunză, mai exact în ID-urile acelor noduri. Muchiile reprezintă predicate, iar în cazul nostru, nodurile intermediare reprezintă containere pentru proprietăți. Pentru a exemplifica, faptul că o fotografie este făcută iarna va fi reprezentat printr-un nod cu ID „iarna” legat de nodul „context” printr-o muchie cu ID-ul având valoarea *http://www3.infoiasi.ro/~sebicu/Licenta/termeni/season*.

Biblioteca *DriveRdf* oferă posibilitatea căutării printre nodurile grafului a unui nod cu un anumit ID (conținut), dar noi suntem interesați și de acele cuvinte care se potrivesc doar parțial cu informațiile din noduri, de aceea este nevoie să apelăm la colecția de „3-uple” corespunzătoare grafului RDF. Astfel, prin iterarea tuturor 3-uplelor, vom putea găsi toate acele „obiecte” (câmpul *object* din cadrul 3-uplului; proprietatea), cu care cuvântul dat se potrivește, fie total, fie parțial. Când găsim un astfel de 3-uplu, construim o înregistrare *IndexEntry* și o adăugăm la rezultate.

```
IRdfNTripleCollection triples = sgraph.GetNTriples();
foreach(IRdfNTriple triple in triples)
{
    if(triple.Object.ToLower().IndexOf(word) >= 0
        && !triple.Predicate.EndsWith("#type>")
        && !triple.Object.StartsWith("_:"))
    {
        IndexEntry entry = new IndexEntry(word, triple.Predicate, photo.Id);
        if(triple.Object.Trim(' ').Length != word.Length)
            //daca nu se potriveste întreg cuvântul
            entry.part_of = true;
        entries.Add(entry);
    }
}
```

În testul de potrivire a cuvântului am mai adăugat unele condiții pentru evitarea erorilor logice: nu dorim potrivirea cuvântului peste nodul „*type*” (*MIME type*), care este întotdeauna „*image*”, și nici peste nodurile de tip *container*, reprezentate prin noduri anonime cu un identificator atribuit de funcția de procesare.

În final, nu rămâne decât să returnăm rezultatele obținute:

```

    }
return entries;
}

```

Clasa *PhotoFinder* - metoda *GetPhotoIds()*

Un „*finder*” folosește un câmp *photoIds*, de tip *Hashtable*, pentru a reține ID-uri de fotografii. ID-urile sunt adăugate prin funcția *GetPhotoIds()*, care găsește fotografiile ce se potrivesc peste o colecție de cuvinte primită la intrare. Pe baza ID-urilor stocate la un moment dat, *finder*-ul poate construi și returna un set de fotografii prin apelul uneia dintre funcțiile *GetPhotos()* – pentru căutare simplă, după cuvinte cheie – sau *GetFilteredPhotos()* – pentru căutare avansată, după cuvinte cheie și diverse filtre. Nu vom prezenta aceste funcții aici, dar este suficient să știm că ele pot construi setul de fotografii necesar, odată ce ID-urile au fost selectate.

PhotoIdEntry
+phid : int
+relevance : int
+words : int
+PhotoIdEntry()

În cadrul obiectului de tip *PhotoFinder*, ID-urile sunt reținute într-o colecție de tip *Hashtable*, sub formă de perechi [*photoId*, *PhotoIdEntry*]. Am construit clasa *PhotoIdEntry* pentru a reține date suplimentare legate de fiecare ID memorat, cum ar fi relevanța (importanța) fotografiei pentru căutarea curentă și numărul de cuvinte care s-au potrivit peste această fotografie. Reținerea relevanței ne va ajuta să ordonăm eficient și corect fotografiile găsite la o căutare, iar numărul de cuvinte ne va fi necesar pentru folosirea operatorilor logici, după cum vom vedea.

Inițializări:

- se pregătește o colecție locală de tip *Hashtable* (similară câmpului *photoIds*) pentru ID-urile ce vor fi selectate;
- se încearcă deschiderea documentului index pentru albumul dat; dacă nu se reușește (documentul fie nu există, fie nu e corect formatat), se creează un document nou;
- se creează un navigator *XPath* pentru fișierul de index; menționăm că folosim două tipuri de obiecte asociate cu documentul index: deși obiectele de tip *XmlDocument* au capacități de regăsire de noduri, le vom folosi doar pentru modificarea fișierului; pentru căutare vom folosi navigatorul *XPath*, care sunt optimizate pentru o căutare rapidă.

```

/// <summary>
/// Selectează id-urile fotografiilor care se potrivesc peste
/// cuvintele cheie într-o colecție dată și le adaugă la colecția
/// curentă de id-uri.
/// Actualizează și fișierele de index atunci când un cuvânt dat
/// nu fusese indexat anterior
/// </summary>
/// <param name="coll">The given collection of keywords</param>
/// <param name="album"></param>
internal void GetPhotoIds(ArrayList coll, Album album, bool and)
{
    string indexPath="";
    Hashtable selectedIds = new Hashtable();
    if(album != null)
        indexPath = album.XmlFullVirtualPath
    XPathDocument doc = null;
    try
    {
        doc = new XPathDocument(indexPath)
    }
    catch
    {///the document is not well-formed or missing
        if(File.Exists(indexPath)) File.Delete(indexPath);
    }
    if(!File.Exists(indexPath))
    {///create a new well-formed index document
        XmlDocument newFile = new XmlDocument();
        newFile.AppendChild(newFile.CreateXmlDeclaration("1.0", "UTF-
8", "yes"));
        newFile.AppendChild(newFile.CreateElement("index"));
        newFile.Save(indexPath);
    }

    if(doc == null)
        doc = new XPathDocument(indexPath);

    XPathNavigator nav = doc.CreateNavigator();
    IEnumerator wordEnum = coll.GetEnumerator();
    int maxId = DBFacade.MaxId(token);
    ...

```

Procesarea:

În continuare, se iterează în colecția de cuvinte primită la intrare și se realizează următoarele operații:

- caută apariții ale cuvântului în index;
- pentru fiecare apariție, selectează ID-urile fotografiilor în care apare;
- pentru fiecare ID, dacă există deja în colecție, îi crește relevanța; dacă nu, este adăugat, cu relevanță 1.

Șirul de biți *found* este folosit în procesul de numărare a cuvintelor potrivite pentru fiecare fotografie. O simplă incrementare a variabilei *words* pentru un

idEntry nu este suficientă, deoarece un cuvânt se poate potrivi de mai multe ori cu o fotografie, ceea ce ar duce la inconsistențe la nivel logic, cum ar fi o fotografie care s-a potrivit cu 3 cuvinte, când de fapt în căutare au fost introduse doar două cuvinte. De aceea, pentru fiecare *idEntry* găsit ca potrivire, verificăm mai întâi dacă acea fotografie a mai fost găsită pentru cuvântul curent. Dacă nu a mai fost găsită, îi putem crește numărul de cuvinte potrivite, și o marcăm ca fiind „dirty” (găsită) pentru cuvântul curent.

```

...
while(wordEnum.MoveNext())// pentru fiecare cuvânt de cautare,
{
    BitArray found = new BitArray(maxId,false);
    //initializam tabloul de biti cu fotografiile
    //gasite pt cuvantul curent
    string word = wordEnum.Current.ToString().ToLower();
    XPathNodeIterator it = nav.Select("//search[@word = '" + word +
    "'"]);//cautam aparitii ale cuvantului in index
    while(it.MoveNext()) // pentru fiecare aparitie,
    {
        XPathNodeIterator itp = it.Current.Select("entry/appears_in/photo");
        //selectam fotografiile in care apare
        while(itp.MoveNext())//pentru fiecare poza,
        {
            //luam id-ul
            int photoId =
                int.Parse(itp.Current.GetAttribute("phid","").ToString());
            if(selectedIds.ContainsKey(photoId))
                //daca mai am idul in colectie, ii cresc relevanta
                {
                    ((PhotoIdEntry)selectedIds[photoId]).relevance += 1;
                    //aici se poate aduna in functie de entry[appears_as] sau
                    //orice alte definitii alese pentru relevanta
                    //daca avem operator "and" si poza nu a mai fost gasita,
                    //o marchez ca gasita si ii cresc nr de cuvinte matched
                    if(and)
                        if(!found[photoId])
                        {
                            found[photoId] = true;
                            ((PhotoIdEntry)selectedIds[photoId]).words++;
                        }
                    //daca avem operator "or", nu ne intereseaza daca
                    //se potriveste sau nu cu toate cuvintele
                    //ci doar sa-i crestem relevanta, lucru facut deja
                }
            else//daca nu am id-ul in colectie, creez o noua intrare pt ID
            {
                PhotoIdEntry newEntry = new PhotoIdEntry(photoId, 1);
                if(and)
                {
                    found[photoId] = true;
                    //fotografia a fost gasita pentru cuvantul curent
                    newEntry.words = 1;//si se potriveste cu un cuvânt pana acum
                }

                selectedIds.Add(photoId, newEntry);
                //daca nu mai am id-ul in colectie,il adaug
            }
        }
    }
}

```

Până în acest punct, funcția procesează corect cuvintele existente în fișierul index, adică cele pentru care s-a mai efectuat anterior o căutare. În continuare, ne ocupăm de cuvintele neindexate, respectiv cele care nu au fost găsite în fișierul index. Pentru acestea vom căuta potriviri în detaliile fiecărei fotografii din albumul selectat, vom adăuga ID-ul fotografiei la ID-urile curente și vom actualiza indexul.

Furnizăm o descriere a acestor operații:

- găsim aparițiile cuvântului în setul de fotografii, și creăm înregistrări de tip *IndexEntry*, cu ajutorul funcției *CreateIndexEntries()* din clasa *SearchUtilities*, prezentată anterior;
- ordonăm setul de înregistrări după „word” și „appears_as”, pentru a simula structura arborescentă din index;
- parcurgem înregistrările cu apariții, actualizăm colecția de ID-uri și fișierul index.

```

if(it.Count == 0)
{
    Photos photos = DBFacade.GetPhotosFromAlbum(token, album.Id
    ArrayList entries = SearchUtilities.CreateIndexEntries(token, word,
    photos);
    IndexEntryComparer comp = new IndexEntryComparer();
    entries.Sort(comp);
    string curr_appas="";
    IEnumerator entryEnum = entries.GetEnumerator();
    XmlDocument docUpdater = new XmlDocument();
    docUpdater.Load(indexPath);

    XmlElement indexNode = docUpdater.DocumentElement;
    XmlElement entryNode = null, app_asNode = null, wordNode = null;
    XmlElement app_inNode = docUpdater.CreateElement("appears_in");

    wordNode = docUpdater.CreateElement("search");//creaza nodul cuvant
    XmlAttribute att = docUpdater.CreateAttribute("word");
    att.InnerText = word;
    wordNode.Attributes.Append(att);

    while(entryEnum.MoveNext())//parcurgem inregistrările cu aparitii
    {
        IndexEntry entry = (IndexEntry)entryEnum.Current;
        if(selectedIds.ContainsKey(entry.photoId))
        //daca am deja fotografia asta in photoIds, ii cresc relevanta
        {
            ((PhotoIdEntry)selectedIds[entry.photoId]).relevance++;
            if(and)
                if(!found[entry.photoId])
                {
                    found[entry.photoId] = true;
                    ((PhotoIdEntry)selectedIds[entry.photoId]).words++;
                }
        }
    }
}

```

```

else//daca nu o am, o adaug
{
    PhotoIdEntry newEntry = new PhotoIdEntry(entry.photoId, 1);
    if(and)
    {
        found[entry.photoId] = true;
        newEntry.words = 1;
    }
    selectedIds.Add(entry.photoId, newEntry);
    //daca nu mai am id-ul in colectie,il adaug
}
XmlElement photoNode = docUpdater.CreateElement("photo");
//nodul fotografia de adaugat pt cuvantul curent
XmlAttribute phid = docUpdater.CreateAttribute("phid");
//id-ul fotografiei
phid.InnerText = entry.photoId.ToString();
XmlAttribute fullMatch = docUpdater.CreateAttribute("full_match");
//cuvantul se potriveste in totalitate sau partial?
fullMatch.InnerText = (!entry.part_of).ToString();
//inregistrarea curenta (entry) ne spune acest lucru
photoNode.Attributes.Append(phid);
photoNode.Attributes.Append(fullMatch);

if(entry.appears_as.CompareTo(curr_appas) != 0 )
//cand se schimba 'appears_as'-ul, adaug nodul curent si
//incep sa construiesc alt nod
{
    curr_appas = entry.appears_as;
    if(entryNode != null)//daca entryNode este null,
    //inseamna ca sunt la inceput; de-abia urmeaza sa il construiesc
    { //aceste operatii se realizeaza la schimbarea de appears_as,
        //dar nu si pentru primul nod creat
        entryNode.AppendChild(app_asNode);
        entryNode.AppendChild(app_inNode);
        wordNode.AppendChild(entryNode);
        app_inNode = docUpdater.CreateElement("appears_in");
    }
    //aceste operatii se realizeaza la schimbarea de appears_as,
    //inclusiv pt primul nod creat
    entryNode = docUpdater.CreateElement("entry");
    //creaza o noua intrare(entry)

    app_asNode = docUpdater.CreateElement("appears_as");
    //creaza nodul appears_as
    app_asNode.InnerText = entry.appears_as.Trim('<', '>');
}
app_inNode.AppendChild(photoNode);//adaug nodul fotografie construit
}

if(entryNode != null)
{ //la iesirea din bucla, pot ramane cu un nod entry pregatit
//dar neinserat
//daca acesta este cazul, il inserez acum
entryNode.AppendChild(app_asNode);
entryNode.AppendChild(app_inNode);
wordNode.AppendChild(entryNode);
}

indexNode.AppendChild(wordNode);
//adaug nodul cuvant (elementul <search>) la index

```

```
docUpdater.Normalize();//indentez fisierul de index
docUpdater.Save(indexPath);//salvez
}
```

În acest moment, tabela locală *selectedIds* conține ID-urile tuturor fotografiilor relevante pentru setul de cuvinte procesat. Acum se poate face selecția ID-urilor ce vor fi adăugate la colecția curentă în funcție de operatorul logic folosit:

- dacă acesta este „sau”, atunci se adaugă toate ID-urile găsite la colecția curentă,
- dacă el este „și”, se adaugă doar acele fotografii care s-au potrivit cu toate cuvintele date.

```
foreach(object key in selectedIds.Keys)
{
    if(((PhotoIdEntry)selectedIds[key]).words == coll.Count || !and)
        photoIds.Add(key, selectedIds[key]);
}
```

În acest moment, ID-urile corespunzătoare căutării după cuvinte cheie și operatori logici sunt selectate și stocate în obiectul de tip *PhotoFinder*, iar fișierele de index sunt actualizate. Construirea colecției de fotografii (un obiect de tip *Photos*) se va face plecând de la aceste ID-uri și, eventual, aplicând filtrele de căutare avansată.

4. CONCLUZII

Noile tehnologii bazate pe XML fac posibilă dezvoltarea unei întregii noi game de produse și servicii Internet. Prin realizarea acestei aplicații am ilustrat doar una dintre posibilele direcții de dezvoltare, bazată pe descrierea și identificarea semantică a resurselor.

Posibilitățile de dezvoltare sunt, însă, mult mai mari, ele fiind practic restricționate mai degrabă de resursele disponibile (umane și materiale) pentru realizarea unui proiect decât de suportul tehnologic.

Referințe

Brickley, D., Miller, L., *FOAF Vocabulary Specification*, 2005: <http://xmlns.com/foaf/0.1/>

Brickley, D., *Photo metadata: the co-depiction experiment*, 2002:
<http://rdfweb.org/2002/01/photo/>

Brickley, D., *RDF: Understanding the Striped RDF/XML Syntax*:
<http://www.w3.org/2001/10/stripes/>

Buraga, S., *Semantic Web. Fundamente și aplicații*, Matrix Rom, București, 2004:
<http://www.infoiasi.ro/~sweb/>

Carroll, J., *RDF Validation Service*, W3C: <http://www.w3.org/RDF/Validator/>

Dublin Core Metadata Initiative, *Dublin Core Metadata Element Set*, 2004:
<http://dublincore.org/documents/dces/>

McBride, B., Beckett, D., *RDF/XML Syntax Specification*, W3C, 2004:
<http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>

McBride, B., Hayes, P., *RDF Semantics*, W3C, 2004:
<http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>

McBride, B., Manola, F., Miller, E., *RDF Primer*, W3C, 2004:
<http://www.w3.org/TR/rdf-primer/>

Capitolul 5

MUMSAI – UN SISTEM DE AUTENTIFICARE AUTOMATĂ

Lenuța Alboai^{1,2}, Sînică Alboai²

¹Facultatea de Informatică, Universitatea Al.I.Cuza
Str. General Berthelot, nr. 16, Iași, România
adria@infoiasi.ro – <http://www.infoiasi.ro/~adria/>

²S.C Axiologic Research
Str. C. Negri, nr. 39, Iași, România
{adria, abss}@axiologic.ro – <http://www.axiologic.ro/>

Rezumat. MUMSAI este un sistem care face posibilă autentificarea automată pe mai multe situri ale unei organizații. În cadrul capitolului de față vom prezenta aspectele principale care fac posibilă funcționarea unui astfel de sistem. Implementarea se bazează pe serverul de aplicații PHP (PHP: *Hypertext Processor*) și pe protocolul SOAP (*Simple Object Access Protocol*).

Cuvinte-cheie: autentificare, SOAP, *iframe*, PHP.

1. INTRODUCERE

Vom începe discuția plecând de la un simplu scenariu general în care un utilizator dorește să se autentifice pe un anumit sit. La un prim pas se va trimite o cerere către server. Acesta îi va întoarce ca răspuns o pagină și va seta în antetul răspunsului HTTP mai multe *cookie*-uri. Simultan, pe partea de server se inițiază o sesiune.

Cookie-urile pot fi văzute ca o pereche (*nume, valoare*) și sunt folosite pentru identificarea sesiunii (Buraga, 2001). Într-o sesiune se pot stoca diferite informații cum ar fi: utilizator, parolă, adresă de e-mail etc. Aceste informații însă nu pot fi plasate în *cookie*-uri pentru că nu ar mai exista securitate. Este important să remarcăm că nu se pot citi *cookie*-urile setate de alt domeniu și de aici necesitatea apariției aplicației noastre.

2. ABORDAREA MUMSAI

MUMSAI (*Management of Users on Many Sites and Auto-login all the Internet*) este un sistem de autentificare centralizat compus dintr-un sit **server** și mai multe situri **client**.

În discuțiile noastre viitoare vom folosi următoarele notații:

- **S** – serverul principal care autentifică utilizatorii ;
- **C₁, ..., C_n** – clienții care autentifică utilizatorii cu ajutorul **S**.

Serverul principal execută două funcții:

1. Autentificare : se verifica autenticitatea pentru perechea (*user,parola*) trimisă de un client și se trimite răspunsul corespunzător împreună cu permisiunile utilizatorului în caz afirmativ.
2. Conectare (*logged*) automată: conectarea pe un sit, implică desigur conectarea pe toate siturile organizației.

Vom prezenta în această secțiune cele două situații în care se poate afla utilizatorul atunci când dorește să se autentifice și vom descrie în paralel mecanismele care stau la baza funcționării sistemului MUMSAI.

Avem două cazuri:

- cel în care utilizatorul s-a conectat direct pe server;
- cel în care utilizatorul s-a conectat pe unul din siturile client.

Cazul 1. Utilizatorul este conectat pe server

În această situație, utilizatorul și-a introdus ID-ul și parola pe situl server **S** (suntem la momentul T_0).

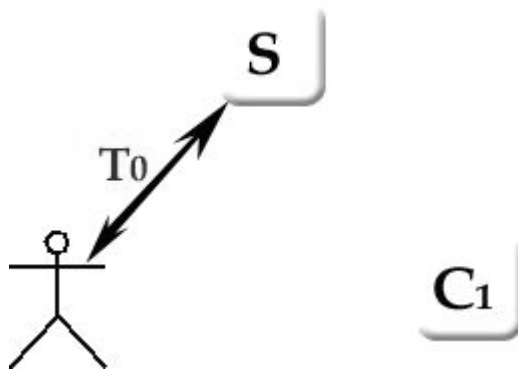


Figura 1. Cazul 1 – Momentul T_0

Întrebarea care se pune este următoarea: care este mecanismul care permite utilizatorului să fie conectat automat pe toate celelalte situri ale organizației? Altfel spus dacă utilizatorul intră pe situl client, de exemplu **C₁**, acesta ar trebui să știe cine este utilizatorul.

Considerăm că la momentul T_1 utilizatorul face cerere la **C₁**.

La momentul T_2 au loc:

- **C₁** generează un număr aleatoriu;
- **C₁** îi va întoarce utilizatorului pagina în care sunt incluse două *iframe*-uri.

Aceste *iframe*-uri au forma:

```
<iframe src="http://S/getFrame.php?key=nr_random&client=C1" />
<iframe src="http://C1/getFrame.php?key=nr_random" />
```

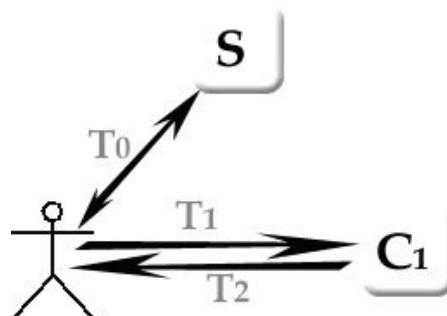


Figura 2. Cazul 1 – Momentele T_1 și T_2

La momentul T_3 , S și C_1 primesc cererile din *iframe*-uri. S va ști valoarea lui *key* și, în plus, S deține o sesiune cu utilizatorul. Deci, în acest moment S cunoaște că o anumită valoare (a parametrului *key*) este asociată cu un utilizator.

La un moment T_3' S va comunica această informație (folosind SOAP) sitului client C_1 .

Facem observația că inițierea acestei comunicări de către C_1 sau de către S depinde de implementare.

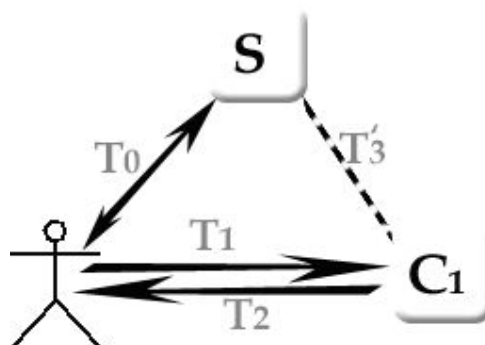


Figura 3. Cazul 1 – Momentul T_3

Iar la momentul T_3'' C_1 comunică utilizatorului faptului că este conectat – se realizează, de fapt, un *refresh* al *iframe*-ului pentru care avem:

```
src="http://C1/getFrame.php?key=nr_random"
```

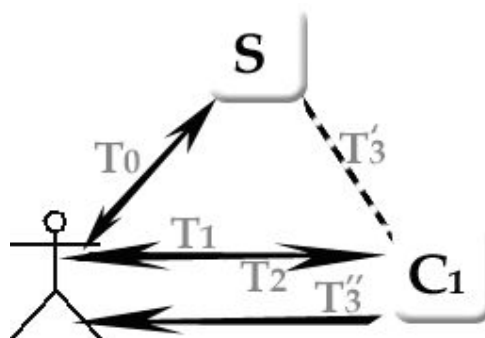


Figura 4. Cazul 1 – Momentul T_3''

Utilizatorul va avea senzația că s-a conectat automat.

Cazul 2. Utilizatorul este conectat pe unul din siturile client C_1, \dots, C_n

În această situație utilizatorul și-a introdus ID-ul și parola pe un sit client, să zicem C_1 (suntem la momentul T_0). Problema care trebuie rezolvată în acest caz este: cum se face autentificarea pe server?

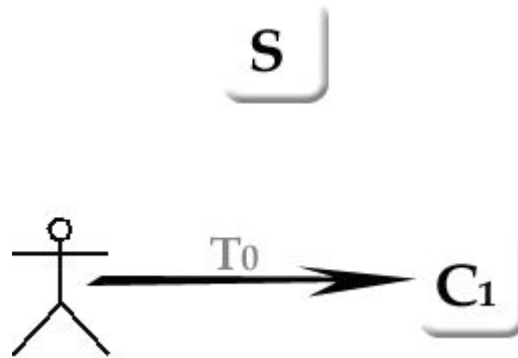


Figura 5. Cazul 2 - Momentul T_0

Facem observația că la un moment T_0' , imediat ce C_1 a primit cererea se va încerca *log*-area utilizatorului. Fiindcă această acțiune nu funcționează, la momentul T_1 C_1 îi întoarce două *iframe*-uri. Unul care conține câmpurile în care utilizatorul trebuie să introducă ID-ul și parola (care face apel la C_1) și un *iframe* cu apel la S :

```
<iframe src="http://S/getframe.php?client=C1&key=nr_random" />
<iframe src="http://C1/getFrame.php?key=nr_random" />
```

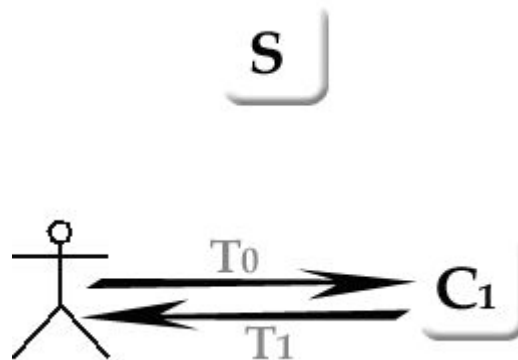
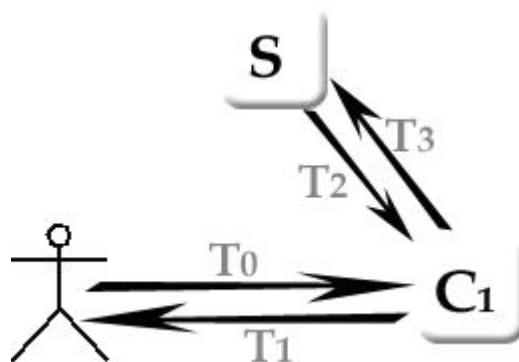


Figura 6. Cazul 2 - Momentul T_1

La momentul T_2 , serverul S creează o sesiune și întreabă prin SOAP pe C_1 cine este utilizatorul conectat căruia îi corespunde acel *key*. C_1 îi răspunde (tot folosind SOAP) prin perechea ID de utilizator și parolă (la momentul de timp T_3).

Figura 7. Cazul 2 - Momentele T₂ și T₃

Din acest moment, orice vizită pe siturile client C₂, ..., C_n ne duce la cazul 1, discutat anterior.

Mecanismul de autentificare asigură faptul că o a treia parte care citește neautorizat traficul nu poate realiza o autentificare frauduloasă.

3. TEHNOLOGII FOLOSITE. DETALII DE IMPLEMENTARE

Pentru implementarea sistemului MUMSAI, tehnologiile principale folosite sunt: SOAP (*Simple Object Access Protocol*), PHP, *iframe*-uri.

După cum s-a putut vedea în secțiunea anterioară, folosirea *iframe*-urilor este cheia sistemului MUMSAI. Prin inserarea unui *iframe* în pagina, se poate executa scriptul specificat de atributul *src*. Acesta poate fi localizat și pe alt domeniu și, în cazul setării în sesiune a unei variabile, aceasta rămâne valabilă pe tot timpul cât *browser*-ul este pornit. Dacă se vizitează acel domeniu, variabila din sesiune poate fi accesată de alt script provenind din același domeniu.

În MUMSAI, comunicarea între client și server se realizează prin apeluri SOAP.

Protocolul SOAP este un protocol simplu, utilizat pentru schimbul de informații într-un mediu distribuit, descentralizat.

SOAP permite (Alboaie & Buraga, 2003; SOAP, 2003):

- schimbul de informații structurate într-un mediu distribuit și descentralizat;
- accesarea de servicii, obiecte într-o manieră independentă de platformă.

Scopul principal al SOAP este facilitarea interoperabilității între platforme și limbaje de programare.

În MUMSAI un client SOAP are următoarea formă:

```
// $address=adresa serverului soap,
$client = new soapclient($address );
// (ex:http://domain/server.php)
// param=vector de parametri ce sunt trimisi
serverului
$params =
array("param1"=>$param1, "param2"=>$param2...);
// namespace-ul functiei
$namespace="urn:xmethods-BNNumefunctie";
// metoda call din clasa soap are ca parametrii
functia din server
// parametrii ei , namespace
$result = $client-
>call('NumeFunctie', $params, $namespace);
if (isset($fault)) {
print "Error: ". $fault;
return false;
}
else
// rezultatul intors de serverul soap(este un
array)
return $result;
```

În serverul SOAP se specifică funcția din client cu același nume și parametrii din vectorul *\$params*:

```
Function NumeFunctie ($param1, $param2...) {
// prelucrare param
// ...
// serverul poate intoarce catre client un
array
return
array("param1"=>$param1, "param2"=>$param2);
}
// instantierea unui server soap
$server = new soap_server;
// inregistrare servicii
$server->register('NumeFunctie');
// in caz de eroare se poate apela alt
server, fisier
$fault = $server-
>fault('soap:Server', '', $error);
// Trimite rezultatul SOAP
$server->service($HTTP_RAW_POST_DATA);
```

Vom considera mai departe tabelele pentru autentificare folosite în cadrul sistemului MUMSAI.

Unui utilizator conectat îi corespunde o linie în ambele tabele, pe client și pe server. O înregistrare este ștearsă în momentul acțiunii de *logout*. Dacă utilizatorul nu urmează *link-ul logout* și închide direct *browser-ul*, linia rămâne în tabel, dar este suprascrisă la următorul *login*.

Pentru server avem:

- Tabela LOGIN:

Camp	Tip	Descriere
User	Varchar	Nickul userului logat
User_key	Varchar	Cheia generata pentru un user
Logintime	datetime	Timpul cand s-a logat

- Tabela AutoIncIDs:

Camp	Tip	Descriere
IDResources	int	Id-ul site-ului pentru care este valabil un auto id
maxNumber	int	numar incrementat automat la fiecare apel soap
Date	datetime	ziua de valabilitate pentru un maxNumber

Pentru client avem:

- Tabela LOGIN:

Camp	Tip
User	Varchar
User_key	Varchar
Logintime	datetime

- Tabela Autoinc:

Camp	Tip	Descriere
id	int	nr incrementat automat la fiecare apel catre server
valid_date	datetime	ziua de valabilitate al id-ului

În secțiunea 2 am discutat despre modul cum se face autentificarea automată a utilizatorilor. Pe lângă această facilitate, MUMSAI asigură și de-conectarea automată de pe toate siturile organizației.

Și în acest caz avem două situații:

1. *de-conectarea de pe situl server*

Se parcurg pașii următori:

- ștergerea din tabela proprie a liniei *userlogat*, ștergerea din sesiune;

- parcurgerea pe rând a siturilor din tabela *Sites* și apelarea serverelor SOAP de pe fiecare;

```

$sql="select url from ".TABLE_RESOURCES."";
$result=$db->Query($sql);
while (($r=mysql_fetch_array($result))) {
    $c=new CLOGOUTClient(); $c-
    >Connect("http://".$r['url']."/SERVER/
    server.php");
    $user=$c->SendLogout($usr);
    $c->Close();
}

```

- se trimite utilizatorul care trebuie de-conectat. Un server de pe un sit C_n șterge din tabela proprie linia corespunzătoare, astfel că la următorul *refresh* sau apel de pagină nu va găsi nici o linie în tabelă care să corespundă cu cheia lui, întreabă serverul S și acesta nu are nimic înregistrat în sesiune, deci nu e conectat nici un utilizator pe acea mașină.



Figura 8. Logout și apeluri SOAP

2. de-conectarea de pe unul dintre siturile client

Se parcurg pașii următori:

- ștergerea din tabela proprie a liniei (*user*, cheie), ștergerea din sesiune a cheii;
- apelul SOAP către S;

```

$c=new CLOGOUTClient();
$c->Connect("http://www.your-
domain.com/server.php");
$c->SendLogout($user);
$c->Close();

```

- serverul la rândul lui șterge din tabela proprie LOGIN linia corespunzătoare (*user*, cheie) și apoi trimite în aceeași manieră (a se vedea figura 8) un apel SOAP la fiecare sit C_n să șteargă din tabelele corespunzătoare acel utilizator. La finalul *logout.php* de pe un C_n , se apelează *logoutServer.php* de pe S pentru a șterge și din sesiunea serverului utilizatorul conectat. Apelul se realizează prin intermediul unui *iframe* avînd drept parametru cheia curentă de pe un sit C_n .

4. TESTARE

Pentru a testa modul de funcționare a sistemului MUMSAI, vă sugerăm să vă creați un cont pe <http://www.axiologic.net/> (serverul S din discuțiile noastre). După autentificare, veți observa că puteți naviga pe oricare situri care fac parte din sistemul de autentificare comun, fără a mai fi nevoie de nici o operație de *login*.

O listă a acestor situri cuprinde:

- <http://www.ro.free-test.info/>
- <http://today-news.info/>
- <http://www.intrebare.ro/>

5. CONCLUZII

MUMSAI este un sistem de autentificare care permite ca un utilizator odată autentificat pe un sit, să poată fi automat autentificat pe alte situri care fac parte din aceeași organizație. Mai mult, de-conectarea de pe un sit presupune de-conectarea automată de pe toate siturile organizației.

Un alt proiect a cărui funcționalitate este similară cu MUMSAI este *Passport Network* creat de Microsoft și care permite conectarea pe *MSN Messenger*, *MSN Hotmail*, *MSN Music*, precum și a altor situri și servicii înrudite. Însă diferența constă în faptul că sistemul nostru poate fi folosit de către orice organizație care dorește utilizarea de tehnologii neproprietare.

Referințe

Alboaie, L., Buraga, S., „Dialoguri despre SOAP”, *NET Report*, feb. 2003:
<http://www.infoiasi.ro/~busaco/publications/articles/SOAP.pdf>

Buraga, S., *Tehnologii Web*, Matrix Rom, București, 2001:
<http://www.infoiasi.ro/~busaco/books/web.html>

Buraga, S. (coord.), *Aplicații Web la cheie*, Polirom, Iași, 2003:
<http://www.infoiasi.ro/~phpapps/>

***, SOAP: <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>

***, PHP: <http://www.php.net/>

***, IFRAME: <http://www.htmlhelp.com/reference/html40/special/iframe.html>

***, MUMSAI: <http://www.axiologic.net/MUMSAI/>

Capitolul 6

DEZVOLTAREA EXTENSIILOR PENTRU FIREFOX

Sergiu Dumitriu

Facultatea de Informatică, Universitatea Alexandru Ioan Cuza din Iași
Str. General Berthelot, nr. 16, Iași, România
sdumitriu@info.uaic.ro – <http://purl.org/net/sdumitriu>

Rezumat. În cadrul capitolului de față se face o descriere a structurii extensiilor pentru navigatorul *Mozilla Firefox* și a tehnologiilor implicate în dezvoltarea unei astfel de extensii. Materialul va trece în revistă o serie dintre aspecte privitoare, printre altele, la XUL (*Extensible User-Interface Language*), JavaScript, *Mozilla Application Framework*.

Cuvinte-cheie: *Firefox*, extensie, XUL, XPI, XPConnect, JavaScript.

1. INTRODUCERE

Mozilla Firefox este un navigator *web* gratuit, dezvoltat de *Mozilla Foundation* și sute de voluntari din comunitatea *open-source*.

Spre deosebire de alte aplicații, intenția este ca *Firefox* să fie distribuit ca un navigator minimal, fără funcționalități care, în general, prezintă interes pentru un grup relativ restrâns de persoane, oferind însă un suport vast pentru dezvoltarea de extensii ce pot fi adăugate de către cei interesați de o anumită funcționalitate (*feature*). Astfel, navigatorul este menținut rapid, eficient, cu un pachet mic de instalare și ușor adaptabil necesităților fiecărui utilizator.

Dintre funcționalitățile incluse în *browser* amintim suportul pentru navigare utilizând ferestre interne (*tab-uri*), pentru afișarea știrilor distribuite prin fișiere

RSS (*live bookmarks*), suport extins pentru standarde *web*, suport pentru localizarea în diferite limbi și pentru teme (*skins*).

1.1 about:mozilla

The Book of Mozilla, 12:10 – Netscape Communicator @ Netscape Communications Corporation

La baza fundației Mozilla stă *Netscape Communications Corporation* și navigatorul *Netscape*, versiunea 1.0 a acestuia fiind lansată pe 10 decembrie 1994, sub numele *Netscape Navigator*. Aplicația s-a extins, devenind o soluție completă pentru aplicații internet, incluzând, printre altele, un navigator, client de poștă electronică și editor de cod *HTML*. Suita de aplicații a fost redenumită *Netscape Communicator* începând cu versiunea 4.0 pentru a evita confuzia între suita de aplicații și navigatorul propriu-zis.

The Book of Mozilla, 3:31 – Mozilla Application Suite @ Mozilla Organization

Un pas important în istoria *Firefox* este deschiderea codului sursă al suitei de aplicații *Netscape Communicator*, și pornirea proiectului *Mozilla*, la data de 31 martie 1998. Astfel a fost creată organizația *Mozilla (Mozilla Organization)* ca echipa responsabilă cu menținerea codului și luarea deciziilor de strategie legate de suita *Netscape Communicator*.

Codul aplicației era relativ mare și greu de întreținut, astfel încât s-a preferat abandonarea codului existent și conceperea unui nou model pentru aplicații. Această decizie a condus la dezvoltarea unei noi biblioteci de componente de interfață, un nou motor de randare și poziționare (*layout*) numit *Gecko*, noi tehnologii de descriere a interfeței (*XUL*) și de dezvoltare a componentelor (*XPCOM*, *XPCoconnect*), ș.a., proiectul fiind redenumit *Mozilla Application Suite*.

The Book of Mozilla, 7:15 – Mozilla Firefox @ Mozilla Foundation

Pe 15 iulie 2003, AOL (care între timp devenise deținătorul *Netscape Communications Corporation*), a anunțat închiderea diviziei responsabile de navigatoarele *Web*, deci încetarea suportului din partea companiei fondatoare pentru dezvoltarea suitei de aplicații *Mozilla*.

În aceeași zi, a fost creată **Fundația Mozilla** (*Mozilla Foundation*), o organizație non-profit compusă în principal din foștii angajați *Netscape*, ce înlocuiește *Mozilla Organization*. Această fundație gestionează în prezent codul navigatorului și al proiectelor adiacente.

Pe data de 2 aprilie 2003, s-a decis ca efortul programatorilor să fie redirecționat spre dezvoltarea aplicațiilor de sine stătătoare, și nu a unei soluții complete pentru activitățile desfășurate pe *Web*. Astfel a fost creat proiectul *Firefox* (inițial numit *Phoenix*) ca navigatorul oficial dezvoltat de *Mozilla*, și proiectul *Thunderbird* (inițial *Minotaur*), clientul de poștă și de știri electronice. *Mozilla*

Application Suite a continuat însă să fie dezvoltat de *Mozilla Foundation* până la data de 10 martie 2005, când a fost anunțat oficial că nu vor mai fi lansate versiuni noi dincolo de *Mozilla* 1.7. Proiectul a fost redenumit *SeaMonkey* și a intrat sub coordonarea consiliului *SeaMonkey* (*the SeaMonkey Council*).

1.2 Firefox

Termenul de *bloatware* se referă la *soft* multifuncțional care încearcă să răspundă unei game cât mai diversificate de cerințe din partea utilizatorilor, necesitând însă mari cantități de resurse (spațiu pe disc, memorie, timp de procesare). Funcționalitățile (considerate în parte) puse la dispoziție sunt însă adesea folosite de un număr foarte mic de utilizatori. O lege empirică, cunoscută ca mitul 80/20, spune că 80% dintre utilizatori folosesc doar 20% din funcțiile unui program avansat.

Chiar și despre variantele de început ale *Netscape Communicator* se putea spune că sunt prea funcționale, ajungându-se ca ultimele variante ale suitei de aplicații *Mozilla* să includă, pe lângă navigatorul efectiv, un client de poștă și știri electronice, un client *IRC*, un editor pentru pagini *Web* și un *debugger* pentru codul *JavaScript*.

Abundența funcționalităților oferite într-un singur pachet de instalare era deranjantă pentru cei care doreau un simplu program de navigare, sau doar un client de *IRC*, astfel încât s-a decis dezvoltarea de aplicații de sine stătătoare în locul unei suite întregi.

Proiectul numit inițial **m/b** (*mozilla/browser*) și desprins din codul navigatorului inclus în suita de aplicații *Mozilla*, a devenit apoi *Phoenix* (lansat în septembrie 2002, versiunile 0.1 - 0.5), *Mozilla Firebird* (0.6, lansat în mai 2003, până la 0.7.1, octombrie 2003), și în final *Mozilla Firefox*, începând cu versiunea 0.8.

Versiunea 1.0 a fost oficial lansată pe data de 9 noiembrie 2004, având peste un milion de descărcări în prima zi, 25 de milioane de descărcări în primele 99 de zile, și deja peste 100 de milioane de descărcări în prezent. Versiunea stabilă (curentă la momentul redactării acestui text) este 1.5, lansată pe data de 29 noiembrie 2005.

2. STRUCTURA FIREFOX

Pachetul de instalare al navigatorului pune la dispoziție un set minimal de funcționalități, oferind însă un suport excelent pentru adăugarea facilă a unor pachete specializate. Această extensibilitate este asigurată de organizarea eficientă a diverselor sub-proiecte ce compun o aplicație *Mozilla*, separarea clară pe nivele a părților componente și definirea protocoalelor de comunicare între ni-

vele, crearea de protocoale și limbaje noi pentru descrierea componentelor de interfață, a interfețelor cu utilizatorul, a structurii logice a unei aplicații, a punctelor de extensie etc.

2.1 Mozilla Application Framework

Inima unei aplicații Mozilla este *Mozilla Application Framework*, cunoscut inițial ca *XPFE (Cross Platform Front-End)*, o colecție de tehnologii și componente *software* disponibile pe mai multe platforme. Dintre acestea amintim *Gecko*, motorul de afișare și poziționare a elementelor grafice, *Necko*, o bibliotecă de clase pentru comunicarea în rețea, *XPCOM*, o variantă multiplatformă pentru comunicarea între componente, *XPCConnect*, tehnologie ce permite accesarea componentelor *XPCOM* din *JavaScript*, și multe altele.

2.2 Chrome

Pentru a separa o aplicație Mozilla în părți logice, a trebuit să se găsească o metodă de a referenția o parte logică a unei aplicații nu prin calea fizică a unui fișier în care se află componenta, ci printr-o cale logică. Astfel a fost introdus pseudo-protocolul URL **chrome** (cunoscut și drept **spațiul de nume chrome**, văzut ca o colecție de nume logice, și nu în legătură cu spațiile de nume pentru XML), prin care se pot referi subcomponente ale unei aplicații.

Astfel:

- „*chrome://navigator/*” se referă la navigatorul propriu-zis, ca o componentă de sine stătătoare,
- „*chrome://navigator/content/*” se referă la partea de conținut a navigatorului, fișierele ce țin de interfață și de funcționare,
- „*chrome://navigator/skin/*” se referă la tema curentă aplicată navigatorului,
- „*chrome://global/skin/*” se referă la tema globală a unei aplicații,
- „*chrome://navigator/locale/*” se referă la fișierele ce țin de localizarea navigatorului,
- „*chrome://myextension/skin/background.png*” se referă la un fișier efectiv din tema curentă a extensiei cu numele „*myextension*”, și anume fișierul imagine „*background.png*”. Locația efectivă a fișierului nu poate fi determinată direct din URL-ul specificat. Acest fișier poate fi utilizat în aplicație prin locația logică, locația fizică fiind gestionată de platforma *Mozilla Application Framework*. În plus, nefiind specificată în URL o temă grafică anume, la schimbarea temei este posibil ca și fișierul utilizat să se schimbe, independent de URL-ul care îl referă. În acest fel, o nouă temă grafică nu trebuie decât să ofere locații fizice valide pentru locațiile logice, și nu să modifice efectiv codul aplicației de bază.

Uniformitatea structurii unui URL, precum și faptul că acesta reprezintă o referință la nivel *logic* a unei componente, permit dezvoltatorului unei extensii să

trateze identic pe diferite platforme aspecte legate de accesarea resurselor (separatorii diferiți de directoare utilizați în Windows sau în Linux, regăsirea directorului unde se află instalată aplicația sau o anumită extensie, regăsirea unui fișier generic din partea de localizare în funcție de o setare anume a limbii).

Acest protocol permite referențierea unor macrocomponente logice sau module ale aplicației (*global*, *navigator*, *inspector*, *talkback*, *greaseonkey*,...), părți ale unei componente (*content*, *skin*, *locale*), sau fișiere dintr-o anumită parte a unei componente.

De exemplu, „*chrome://myextension/locale/mainWindow.dtd*” se referă la partea de localizare a ferestrei principale a extensiei *myextension*, și nu la fișierul ce traduce în limba engleză conținutul ferestrei principale a extensiei *myextension*; există un astfel de fișier pentru fiecare localizare disponibilă, automat fiind ales cel corespunzător setării curente.

2.3 Navigatorul Firefox

Navigatorul poate fi privit ca o extensie a lui însuși, deoarece respectă aceleași principii ca orice extensie. Este înregistrat ca o colecție de URI-uri valide în protocolul *chrome*, este separat în părțile de conținut, temă grafică și localizare, și, după cum vom vedea în secțiunea următoare, este dezvoltat utilizând tehnologiile uzuale pentru o extensie, *JavaScript*, *XUL*, *CSS*, *DTD*.

Navigatorul efectiv se constituie din fișierele *browser.jar*, *classic.jar*, *en-US.jar* (în cazul instalării variantei în limba engleză), aflate în subdirectorul *chrome* din directorul principal al aplicației. Conținutul acestor fișiere poate fi dezarhivat, modificat și rearhivat, efectele modificărilor fiind observabile imediat după repornirea navigatorului, fără a necesita o recompilare a codului.

2.4 Plugin-uri

Un *plugin* este un program care poate să se integreze într-un alt program gazdă (sau de bază, ce definește un set de reguli de comunicare cu *plugin*-urile) pentru a îndeplini anumite funcții neexistente în gazdă (sau existente, dar cu funcționare incompletă sau incorectă).

În general, un *plugin* constă în cod compilat, și comunică cu partea de jos a platformei, spre deosebire de o extensie, care se suprapune părții de sus a unei aplicații (părțile unei aplicații vor fi descrise în secțiunea 4.2).

Exemple de *plugin*-uri sunt: *Adobe Reader*, *Macromedia Flash Player*, *Java Runtime Environment*, *Quicktime*, *RealPlayer*. Se preferă menținerea unui număr minim de *plugin*-uri în favoarea extensiilor.

2.5 Fișierele *extensions.ini/extensions.rdf*

Pentru a fi recunoscute de aplicație și regăsite în spațiul de nume *chrome*, extensiile curente trebuie să fie înregistrate. Punctul central de gestionare a extensiilor este format din fișierele *extensions.ini* și *extensions.rdf*. Acestea se află în subdirectorul personal al fiecărui utilizator (oferind posibilitatea de particularizare la nivel de utilizator, și nu de calculator). În plus, există câte un subdirector pentru fiecare profil creat (deci și particularizare la nivel de profil pentru fiecare utilizator). Astfel, în WindowsXP calea este:

```
<directorPersonal>\Application Data\Mozilla\Firefox\Profiles\<<profil>\
```

Fișierul *extensions.ini* este un fișier text ce conține locațiile fizice ale directorilor în care se află extensiile. Un exemplu de astfel de fișier este¹:

```
[ExtensionDirs]
Extension0=C:\Program Files\Mozilla Firefox\extensions\
  inspector@mozilla.org
Extension1=C:\Documents and Settings\user\Application Data\Mozilla\
  Firefox\Profiles\tuznrsm.default\extensions\
  {3d7eb24f-2740-49df-8937-200blcc08f8a}
Extension2=C:\Documents and Settings\user\Application Data\Mozilla\
  Firefox\Profiles\tuznrsm.default\extensions\
  {cf2812dc-6a7c-4402-b639-4d277dac4c36}
[ThemeDirs]
Extension0=C:\Program Files\Mozilla Firefox\extensions\
  {972ce4c6-7e08-4474-a285-3208198ce6fd}
```

În secțiunea *[ExtensionDirs]* sunt precizate directoarele extensiilor efective, atât cele globale, instalate în directorul principal al aplicației, cât și cele instalate de fiecare utilizator în parte în directorul personal. În secțiunea *[ThemeDirs]* sunt precizate directoarele în care se află instalate temele grafice aplicației, atât cele globale, disponibile tuturor utilizatorilor, cât și cele instalate de fiecare utilizator în parte.

În *extensions.rdf* apar date de identificare a fiecărei extensii, la nivel global: numele afișat, numele intern, versiunea curentă, pagina oficială a extensiei, versiunile aplicației gazdă suportate, autori, descriere etc.; de asemenea, extensiile sunt înregistrate ca sub-module ale aplicației. Sintaxa este **RDF** (*Resource Description Framework*), descriind efectiv resurse logice ale aplicației.

Fragmentul următor realizează înregistrarea sub-modulelor aplicației:

¹ În fișierul efectiv fiecare extensie este trecută pe o singură linie.

```

...
<RDF:Seq RDF:about="urn:mozilla:item:root">
  <RDF:li RDF:resource="urn:mozilla:item:inspector@mozilla.org"/>
  <RDF:li RDF:resource="urn:mozilla:item:talkback@mozilla.org"/>
  <RDF:li RDF:resource=
    "urn:mozilla:item:{972ce4c6-7e08-4474-a285-3208198ce6fd}"/>
  <RDF:li RDF:resource=
    "urn:mozilla:item:{3d7eb24f-2740-49df-8937-200b1cc08f8a}"/>
  <RDF:li RDF:resource=
    "urn:mozilla:item:{cf2812dc-6a7c-4402-b639-4d277dac4c36}"/>
</RDF:Seq>

```

Fragmentul următor descrie extensia *Flashblock*:

```

...
<RDF:Description
  RDF:about="urn:mozilla:item:{3d7eb24f-2740-49df-8937-200b1cc08f8a}"
  NS1:installLocation="app-profile"
  NS1:version="1.3.3"
  NS1:name="Flashblock"
  NS1:description="Replaces Flash objects with a button."
  NS1:creator="The Flashblock Team"
  NS1:homepageURL="http://flashblock.mozdev.org/"
  NS1:updateURL="http://flashblock.mozdev.org/update.php"
  NS1:optionsURL="chrome://flashblock/content/options.xul"
  NS1:iconURL="chrome://flashblock/content/flashblock.png">
  <NS1:type NC:parseType="Integer">2</NS1:type>
  <NS1:contributor>Jesse Ruderman</NS1:contributor>
  <NS1:contributor>Lorenzo Colitti</NS1:contributor>
  <NS1:contributor>Ovidiu Dan (Romanian translation)</NS1:contributor>
  <NS1:targetApplication RDF:resource="rdf:#$Fu3JX1"/>
  <NS1:targetApplication RDF:resource="rdf:#$Gu3JX1"/>
  <NS1:targetApplication RDF:resource="rdf:#$Hu3JX1"/>
</RDF:Description>
...

```

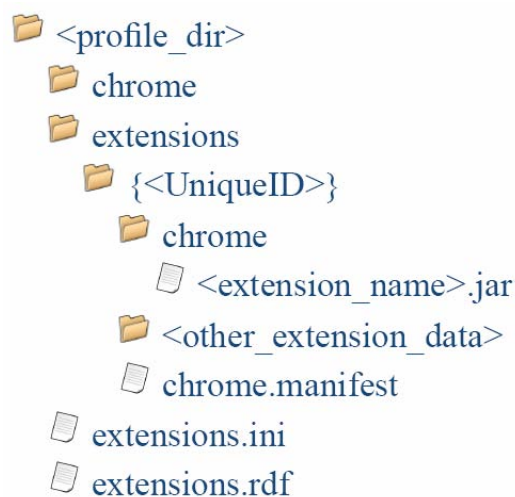


Figura 1. Locațiile implicate în regăsirea și configurarea unei extensii

2.6 Extensiile efective

Peste această infrastructură, sunt instalate extensiile efective, unele globale, accesibile de către toți utilizatorii, altele instalate pentru un anumit utilizator.

Extensiile globale se află în subdirectorul *extensions* din directorul aplicației, fiecare în subdirectorul propriu. Extensiile fiecărui utilizator se află în subdirectorul *extensions* al directorului destinat unui anumit profil. Acestea sunt locațiile recomandate; se poate specifica o altă locație în fișierele de configurare a extensiilor. În general, extensiile sunt distribuite ca arhive *.jar*, pentru a reduce numărul de fișiere și dimensiunea unei extensii.

3. EXTENSIILE FIREFOX

3.1 Introducere

O extensie *Firefox* este o *completare* cu noi funcționalități adusă navigatorului. Aceste funcționalități pot varia ca importanță și complexitate, de la simple butoane în meniu care afișează un text predefinit (cum este cazul banalei extensii „*I must not fear!*”), la funcționalități avansate (suportul pentru standardul *XForms* este oferit ca o extensie). Astfel, se permite particularizarea navigatorului în funcție de preferințele fiecărui utilizator, păstrând în același timp navigatorul de bază la dimensiuni foarte mici.

3.2 Exemple

Extensiile pot fi împărțite în mai multe categorii.

Pentru programare

Cea mai importantă unealtă pentru programatori, *DOMInspector*, este inclusă în pachetul de instalare oficial, dar nu este instalată în mod normal. Permite explorarea și modificarea în timp real a arborelui DOM (*Document Object Model*) al unui document, fie acesta o pagină *Web*, o fereastră a navigatorului, sau chiar fereastra inspectorului.

Alte extensii foarte utile pentru dezvoltatorii de pagini *Web* sunt *JavaScript Debugger* (cunoscut sub numele de *Venkman*), ce permite un *debugging* avansat al codului *JavaScript* din pagini *Web* sau din însuși *browser*-ul și extensiile sale, și *Web Developer*, ce permite numeroase modificări și teste ale unei pagini *Web*. Alte instrumente utile includ editoare vizuale pentru codul *HTML*, validatoare, emulatoare pentru *Internet Explorer*, diverse editoare pentru componentele unei pagini.

Pentru navigare

La limita dintre programare și navigare se află extensia numită *Greasemonkey*, o meta-extensie, sau un suport pentru extensii, cu abilitatea de a crea scripturi speciale pentru fiecare pagină, scripturi ce produc asupra unei pagini modificări persistente (cu efect la fiecare accesare). Astfel, se pot realiza acțiuni precum:

- eliminarea unor secțiuni nedorite din anumite pagini (reclame, animații, informații neinteresante),
- schimbarea modului de interacțiune cu o pagină (se pot introduce funcții AJAX într-o pagină inițial statică)
- reorganizarea completă a unei pagini.

Peste *Greasemonkey* au fost realizate numeroase alte extensii.

Altă unealtă utilă atât pentru programatori, cât și pentru navigare, este *Fangs Screen Reader Emulator*, ce afișează conținutul unei pagini ca și cum ar fi citit de un instrument de citire a ecranului.

Flashblock permite blocarea obiectelor de tip *flash* din pagini, cu posibilitatea pornirii manuale a acestora; *NoScript* blochează complet codul JavaScript din anumite pagini (spre deosebire de opțiunea globală de a dezactiva *JavaScript*).

Numeroase alte extensii facilitează navigarea, prin utilizarea unui *proxy* extern pentru toate protocoalele (*vbrowseit*), ascunderea paginii sursă la urmarea unei legături (*refspoof*), gestionarea dinamică a *cookie*-urilor (*CookieWatcher*) etc.

Pentru amuzament

S-au dezvoltat extensii cu diferite grade de utilitate, cum ar fi jocuri ce pot fi accesate direct din navigator (*Mines*, *Card Games*, *Blockfall*), afișarea definiției unui cuvânt dintr-o pagină (*Dictionary Tooltip*), afișarea stării vremii (*ForecastFox*) etc.

Mai multe extensii și informații despre acestea se găsesc la pagina oficială: <https://addons.mozilla.org/extensions>.

3.3 Structura unei extensii

O extensie constă în mai multe componente:

- Arhiva de instalare, ce cuprinde fișierele efective ale extensiei și fișierele pentru instalare, menite să înregistreze extensia în aplicație (în *extensions.rdf*) și să înregistreze componentele extensiei în spațiul de nume *chrome*.

- Arhiva cu fișierele efective ale extensiei, separate în trei părți:
 - partea de conținut a unei extensii;
 - partea de localizare;
 - partea de aspect.
- Fișierele de parametrizare a extensiei, ce oferă valori inițiale pentru parametrii configurabili.
- Componentele de tip *plugin* necesare extensiei (dacă este cazul).

Fișierul de instalare este o arhivă *zip* cu extensia *.xpi* (*Cross Platform Installer*), cu structura din figura 2.

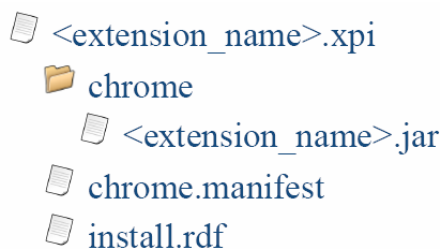


Figura 2. Structura unui fișier *xpi*

Arhiva cu extensia *jar* din directorul *chrome* conține fișierele extensiei. Fișierul *chrome.manifest* conține detaliile cu privire la modulele unei extensii. În acest fișier sunt declarate modulul de conținut al extensiei, modulele de stil și de localizare incluse, punctele de inserție în aplicație a extensiei.

```

content ext jar:chrome/ext.jar!/content/ext/
skin ext classic/1.0 jar:chrome/ext.jar!/skin/classic/ext/
style chrome://browser/content/browser.xul chrome://ext/skin/ext.css
locale ext en-US jar:chrome/ext.jar!/locale/en-US/ext/
locale ext ro-RO jar:chrome/ext.jar!/locale/ro-RO/ext/
overlay chrome://browser/content/about.xul chrome://ext/content/my.xul
  
```

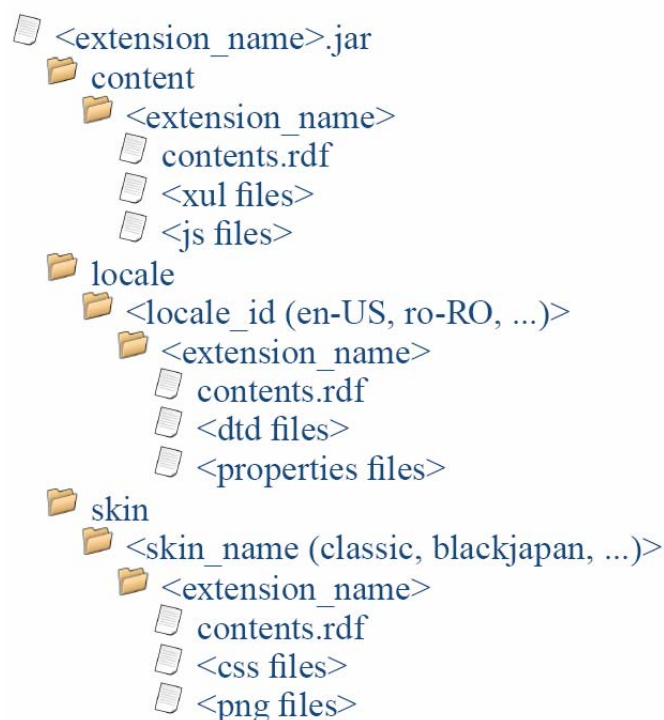


Figura 3. Structura unei arhive *jar* corespunzătoare unei extensii

Fișierul *install.rdf* conține informații de identificare a extensiei, conținut care va fi copiat în mare parte în fișierul *extensions.rdf* din directorul profilului unde va fi instalată extensia.

Arhiva *.jar* ce conține fișierele extensiei are structura din figura 3.

Fiecare parte a unei extensii se află într-un director separat. Partea de conținut se află în subdirectorul *content/<extensie>/* din arhiva *jar*. Fiecare parte de localizare se află în subdirectorul *locale/<localeID>/<extensie>/* din arhiva extensiei. Pot fi distribuite mai multe localizări pentru o extensie într-o singură arhivă, fără ca acestea să interfereze. Fișierele conținând proprietăți de stil sunt stocate în subdirectoarele corespunzătoare fiecărei teme grafice suportate, după modelul *skin/<nume stil>/<extensie>/*.

3.4 Instalarea unei extensii

Pentru a instala o extensie este suficient să se activeze un *link* către fișierul *xpi*. Extensia va fi instalată în profilul curent și va fi activată după repornirea aplicației.

Pașii care se execută sunt:

- se descarcă arhiva;
- se dezarchivează și se verifică dacă extensia este compatibilă cu versiunea curentă a aplicației;
- se copiază arhiva *jar* și *chrome.manifest* în subdirectorul destinat extensiei;
- se copiază conținutul fișierului *install.rdf* în *extensions.rdf*;
- se adaugă calea către subdirectorul extensiei în *install.ini*.

Se poate instala o extensie și dintr-un fișier local, prin tragerea fișierului *xpi* în fereastra navigatorului (*drag and drop*).

Pentru a instala o extensie globală, este necesară rularea din linia de comandă a aplicației, specificând ca parametri „-install-global-extension <fișier.xpi>”.

4. TEHNOLOGII

Extensiile *Firefox* sunt dezvoltate utilizând tehnologii de nivel înalt din *Mozilla Application Framework*, independente de platformă, având la bază tehnologiile de nivel jos, dezvoltate pentru fiecare platformă în parte, în general cod compilat.

Modelul aplicațiilor *Mozilla* permite o separare foarte bine definită a atribuțiilor în părți ale unei extensii. Astfel, se pot dezvolta separat elementele de interfață, logica aplicației, parametrii configurabili, stilul și localizările lingvistice.

4.1 XUL – interfețe independente de platformă

Interfața unei aplicații *Mozilla* este configurabilă, extensibilă și stilizabilă, însemnând că prezența sau absența unor controale nu este fixată în codul compilat al aplicației, ci este precizată într-un fișier separat de descriere a interfeței. Întreaga interfață a navigatorului și interfețele extensiilor sunt descrise utilizând acest mecanism.

Limbajul de descriere utilizat este *XUL* (*Extensible User-Interface Language*), un limbaj din familia *XML* dezvoltat și întreținut de Fundația *Mozilla*. Este disponibil ca parte integrantă a motorului *Gecko*, permițând dezvoltatorilor să creeze aplicații multiplatformă utilizând un limbaj descriptiv de definire a interfeței (*XUL*) și un limbaj de programare interpretat (*JavaScript*). A fost conceput pentru a fi cu adevărat portabil și utilizabil pe platformele *Windows*, *Macintosh*, *Linux* și *Unix*. Spre deosebire de alte platforme pentru crearea de aplicații portabile, cum ar fi *QT* sau *GTK*, o aplicație *Mozilla* nu necesită compilarea interfeței pentru fiecare sistem de operare, ci doar existența executabilului *Gecko* pentru acea platformă.

Deși nu este un standard complet acceptat, *XUL* are multe avantaje față de alte metode de a descrie interfața unei aplicații:

- *XUL* este un limbaj XML;
- structura este simplă și concisă;
- sintaxa este simplă și intuitivă, utilizând nume cunoscute de componente ca elemente;
- permite extinderea facilă a interfețelor prin *overlay-uri*;
- permite atașarea de stiluri CSS;
- este modificabil în timp real prin manipulări *DOM*.

Specificația 1.0, dezvoltată de *Mozilla*, este nefinisată, și în variantele mai noi ale motorului *Gecko* nu mai este pe deplin respectată. Specificația 2.0 este în curs de standardizare, cu o implicare și din partea Consorțiului *Web*.

Un exemplu simplu de interfață descrisă în *XUL* este următorul (a se vedea figura 4):

```
<?xml version="1.0"?>
<window id="example-window" title="XUL este simplu"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <hbox>
    <label value="Apasă aici:"/>
    <button id="close" label="Închide" onclick="window.close()"/>
  </hbox>
</window>
```

Fișierul descrie o fereastră, intitulată „*XUL este simplu*”, ce conține o etichetă (elementul *label*) și un buton (elementul *button*), poziționate orizontal (elementul *hbox*).



Figura 4. Fereastră *XUL*

Extensibilitate

Extensibilitatea unei interfețe *XUL* este asigurată prin tehnica numită *overlay* (suprapunere).

În fișierele *XUL* ce descriu interfața unei extensii se utilizează ca element rădăcină elementul *overlay* în loc de *window*. O interfață astfel definită nu va fi de

sine stătătoare, ci va fi introdusă ca și conținut al unei alte interfețe (*window* sau *overlay*). Pentru aceasta, trebuie urmați doi pași:

1. În *chrome.manifest* se înregistrează fișierul *XUL* al extensiei ca *overlay* al unui element existent deja în aplicație (identificat prin *URL*-ul *chrome*), fie ca fereastră a aplicației de bază, fie ca un alt element al unei extensii.
2. În fișierul *XUL* se identifică, prin scrierea unui element *XUL* de același tip și cu același identificator (atributul *id*), componenta de interfață extinsă (menu, fereastră, colecție de scurtături pentru tastatură etc.). Conținutul acestui element nu va înlocui componenta existentă, ci va fi adăugat pe lângă conținutul deja existent.

De exemplu, pentru a extinde o bară de unelte din fereastra principală a navigatorului, vom scrie:

- În fișierul *myToolbar.xul* al extensiei:

```
<overlay id="myextension_menu" xmlns="...">
  <toolbar id="navigator-toolbox">
    <toolbar ...>
      ...
    </toolbar>
  </toolbar>
</overlay>
```

- În fișierul *chrome.manifest*²:

```
overlay chrome://browser/contents/browser.xul
chrome://myextension/content/myToolbar.xul
```

Alte limbaje

XUL poate fi combinat cu alte limbaje de marcare, cel mai adesea cu *XHTML*. Poate fi utilizată în acest sens orice tehnologie suportată de motorul *Gecko*:

- *SVG* (*Scalable Vector Graphics*),
- elementul *canvas*,
- *applet*-uri Java,
- cod X3D (eventual împreună cu *plugin*-urile corespunzătoare).

² În fișierul efectiv textul trebuie să fie trecut pe o singură linie

Un alt limbaj de marcare, destinat descierii formularelor, este *XForms*, standard W3C, de asemenea suportat de *Mozilla*. Pentru extensii cu caracter științific devine util *MathML*, un puternic limbaj de descriere a expresiilor matematice.

Aceste tehnologii pot fi utilizate fie în combinație cu *XUL*, fie ca documente de sine stătătoare sau generate dinamic prin transformări *XSLT* sau prin *scripting*, și afișate în cadrul unei ferestre a navigatorului.

4.2 *JavaScript* – mic, dar puternic

Întreaga funcționalitate a navigatorului este realizată în *JavaScript*, utilizând componente din *Mozilla Application Framework*, scrise și compilate pentru o anumită platformă, dar oferind o aceeași funcționalitate în toate distribuțiile.

În crearea unei asemenea platforme ce permite scrierea de cod interpretat cu funcționalități avansate, independent de platformă, intervin mai multe nivele.

XPCConnect

La bază este codul C/C++ dependent de platformă responsabil cu afișarea propriu-zisă a elementelor grafice, comunicarea în rețea, accesul la sistemul de fișiere etc. Peste acest cod este pus un nivel de abstractizare menit să descrie funcționalitățile unei clase, ascunzând în același timp detaliile de implementare.

Acest nivel se numește *XPIDL* (*Cross Platform Interface Description Language*), o extensie a standardului *IDL* menținut de *OMG*, ce permite descrierea unor interfețe ce precizează metodele și atributele oferite de o clasă (comportamentul), fără a specifica implementarea efectivă a codului din interiorul unei metode și fără a preciza care clasă va oferi acest cod. În acest mod, o componentă va putea apela metodele unei interfețe la fel pe orice platformă, având același nume și aceiași parametri. O funcționalitate nu va fi deci identificată prin numele clasei și al funcțiilor ce o implementează, acestea putând să difere de la o platformă la alta, ci prin interfața ce oferă acea funcționalitate, indiferent de implementarea efectivă.

Utilizând acest nivel, s-au definit componentele multiplatformă, *XPCOM* (*Cross Platform Component Object Model*). Acesta este o abordare multiplatformă a ceea ce Microsoft a încercat să facă prin *COM*. Pentru aceasta sunt definite interfețe standard *XPIDL* ce trebuie extinse sau utilizate de o componentă pentru a fi recunoscută drept componentă *XPCOM*. La bază se află *nsISupports*, o interfață asemănătoare cu *IDispatch* din Microsoft *COM*, interfață ce trebuie extinsă de fiecare componentă *XPIDL*. Un modul este identificat de un punct central al modului, o componentă ce implementează interfața *nsIModule*. Printre componentele oferite deja de platforma *Mozilla*, amintim:

- *nsIFile*, *nsIDirectoryService*, *nsIFilePicker* pentru accesul la sistemul de fișiere,
- *nsIInputStream*, *nsIOutputStream*, *nsIScriptableInputStream* pentru abstracțizarea fluxurilor de date,
- *nsIClassInfo*, *nsIComponentManager*, *nsIInterfaceRequester*, *nsIFactory* pentru gestiunea componentelor,
- *nsIObserver*, *nsIObserverService*, *nsIWebProgress* pentru observarea operațiilor,
- și multe altele.

Componentele *XPCOM* pot fi folosite din cod C++, dar și direct din *JavaScript*, fără a necesita cod intermediar sau tehnici deosebite. Acest lucru este posibil datorită unei tehnologii numite *XPCConnect* (cunoscut și ca *Scriptable Components*). Aceasta este utilizată împreună cu *XPIDL*, compilarea interfețelor generând, pe lângă fișierele C++ atașate unei interfețe, și fișiere *typelib* utilizate de *XPCConnect* pentru îmbinarea ușoară a codului componentelor compilate cu codul *JavaScript*. O altă facilități oferită de *XPCConnect* este conectarea inversă, adică manipularea obiectelor *JavaScript* din codul compilat. Prin *XPCConnect* în codul *JavaScript* devin vizibile noi componente, implementate de fapt în C++, pe lângă obiectele *JavaScript* predefinite.

Pe baza conectivității oferite de *XPCConnect*, funcționalități oricât de avansate pot fi scrise direct în *JavaScript*, dispărând necesitatea compilării codului ce implementează *comportamentul* unei aplicații. În final, pe baza unor componente de nivel jos, ce oferă funcționalități elementare, se creează la nivel înalt funcționalități complexe.

JavaScript = ECMAScript + E4X + Gecko + DOM + ...

În esență, limbajul *JavaScript* utilizat de *Mozilla Application Framework* este bazat pe *ECMAScript* versiunea 3, limbaj standardizat de consorțiul *ECMA International*. Acesta a apărut din necesitatea de a standardiza un limbaj cu multiple implementări incompatibile, cu un mare potențial în dezvoltarea spațiului Web. *ECMAScript* fixează sintaxa limbajului, câteva componente esențiale predefinite și, mai ales, ideologia limbajului.

JavaScript este un limbaj **bazat pe obiecte, orientat prototip**. Programarea orientată prototip a fost inițial utilizată în limbajul de programare *Self*, și, spre deosebire de limbajele orientate obiect, bazate pe clase și instanțe, nu există decât obiecte. Prototipul unui obiect este, de asemenea, un obiect, în care se caută metodele și proprietățile cerute în cazul în care nu sunt găsite în obiectul inițial. Se creează astfel un lanț de obiecte ce definesc un comportament, într-o manieră asemănătoare moștenirii din POO.

Deși este foarte des utilizat în paginile Web, puțini programatori cunosc adevărata putere a acestui limbaj. În ciuda faptului că mulți îl critică pentru dificultatea verificării corectitudinii, securitatea și predictibilitate precare, și lipsa de eficiență (lucruri care pot fi semnalate în multe programe scrise în limbaje „si-

gure”, precum C, Java sau C#), JavaScript este mult mai flexibil decât limbajele stricte (*strict typed*), nefiind dificil pentru programatori experimentați să elimine neajunsurile menționate mai sus. Tocmai această flexibilitate dă puterea limbajului, dar atrage și cele mai multe critici.

ECMAScript for XML, pe scurt *E4X*, este o extensie a limbajului *ECMAScript*, facilitând manipularea datelor în format *XML*, format tot mai des utilizat pentru stocarea și transferarea datelor de orice fel, și strâns legat de spațiul *Web*, locul unde este utilizat cel mai frecvent *JavaScript*. De curând, această specificație este implementată și în *Mozilla Application Framework*, permițând procesarea mai eficientă a datelor *XML*.

Prin tehnologia *XPConnect* descrisă mai sus, codul *JavaScript* are acces la majoritatea funcționalităților puse la dispoziție de platforma *Mozilla*. Însă acest acces este permis codului aplicației, celui ce ține de navigator sau de extensiile acestuia, și restricționat în cazul codului executat în paginile încărcate, din motive de securitate. Astfel, pentru dezvoltatorii de extensii, codul *JavaScript* satisface aproape toate necesitățile unui programator.

Mozilla implementează o gamă generoasă de specificații DOM:

- DOM Level 1 – complet;
- DOM Level 2 Core – suport aproape complet;
- DOM Level 2 HTML – complet;
- DOM Level 2 Style Sheets – complet;
- DOM Level 2 CSS – suport major;
- DOM Level 2 Events – suport major;
- DOM Level 2 Views – complet;
- DOM Level 2 Traversal – suport parțial;
- DOM Level 2 Range – complet;
- DOM Level 2 SVG – suport major;
- DOM Level 2 XForms – suport major;
- DOM Level 2 XUL – suport complet.

DOM Level 3 încă nu este implementat suficient, dar în special pentru DOM Level 3 Core deja există un suport parțial.

Comportamentul unei extensii

O extensie, după cum am văzut, își descrie elementele de interfață prin limbaje de marcare (în special *XUL*). Comportamentul din spatele acestor componente

este realizat în parte de cod nativ (funcționarea de bază ca *widget-uri*), în parte de cod *JavaScript* (ca elemente active de interacțiune ale aplicației). Codul *JavaScript* este atașat fișierelor *XUL* cu ajutorul elementului *script*, asemănător celui din *HTML*. Astfel, se poate scrie cod *JavaScript* direct în interiorul unui document *XUL* (nerecomandat însă), sau se poate face referință la un fișier ce conține cod *JavaScript* (un fișier *.js*). Se recomandă ca fișierele să fie referite utilizând spațiul de nume *chrome*, și nu o cale relativă.

Un exemplu de atașare de cod *JavaScript*:

```
<overlay xmlns="...">
  <script type="application/x-javascript"
    src="chrome://myextension/content/tools.js"/>
  ...
</overlay>
```

4.3 Localizare – aceeași aplicație, oricâte locații

În modelul aplicațiilor *Mozilla*, pentru a face o aplicație disponibilă în mai multe limbi, nu este necesară rescrierea codului aplicației, sau a fișierelor de interfață. Pentru a schimba textul vizibil, orientarea controalelor, scurtăturile de tastatură utilizate, sau chiar elementele de interfață vizibile, sunt definite module de localizare pentru fiecare extensie. Un astfel de modul conține două tipuri de resurse: fișiere *DTD* (*Document Type Definition*) și fișiere *.properties*.

Fișierele *DTD* sunt utilizate pentru definirea entităților, referențiate în fișierele de interfață. Un fișier *XUL* nu va preciza textul efectiv ce va fi utilizat pentru un atribut, ci va face referință la o entitate. Această entitate va fi definită într-un fișier *DTD* particular unei anumite localizări. Pentru a fi regăsit acest fișier în localizarea curentă, se va preciza o adresă din spațiul de nume *chrome*, și nu o adresă absolută. În acest fel această adresă va conduce la un fișier fizic în funcție de limba curentă a aplicației.

Exemplu de utilizare. În fișierul *content/myextension/mainwindow.xul*:

```
<?xml version="1.0"?>
<!DOCTYPE window SYSTEM "chrome://myextension/locale/mainwindow.dtd">
<window id="main-window" title="&me.mainWindowTitle;"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <label value="&me.lbl.Value;"/>
  <button id="close" label="&me.btn.Label;" onclick="window.close()"/>
</window>
```

În fișierul *locale/en-US/myextension/mainwindow.dtd*:

```
<!ENTITY me.mainWindowTitle "My extension window">
<!ENTITY me.lbl.Value "Click here:">
<!ENTITY me.btn.Label "Close">
```

În fișierul *locale/ro-RO/myextension/mainwindow.dtd*:

```
<!ENTITY me.mainWindowTitle "Fereastra extensiei mele">
<!ENTITY me.lbl.Value "Apasă aici:">
<!ENTITY me.btn.Label "Închide">
```

Entitățile *DTD* sunt utile pentru înlocuirea textelor ce apar static în interfață. Pentru partea dinamică însă, este mai dificil pentru un cod *JavaScript* să utilizeze entități. În acest sens se utilizează fișierele de resurse *.properties*. Acestea sunt

fișiere text ce conțin perechi cheie-valoare ce pot fi accesate ușor din script, cu ajutorul elementelor XUL de tip *stringbundleset* și *stringbundle*. Acestea primesc ca sursă URL-ul unui fișier de resurse, și în timpul rulării, prin DOM oferă metode de obținere rapidă a valorii pentru o cheie dată.

Exemplu de utilizare. În fișierul *content/myextension/mainwindow.xul*:

```
<?xml version="1.0"?>
<!DOCTYPE window SYSTEM "chrome://myextension/locale/mainwindow.dtd">
<window id="main-window" title="&me.mainWindowTitle;"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <script type="application/x-javascript"
    src="chrome://myextension/content/mainscript.js"/>
  <stringbundleset id="stringbundleset">
    <stringbundle id="myextension_bundle"
      src="chrome://myextension/locale/mainwindow.properties"/>
  </stringbundleset>
  <label value="&me.lbl.Value;"/>
  <button id="close" label="&me.btn.Label;" onclick="confirmClose()"/>
</window>
```

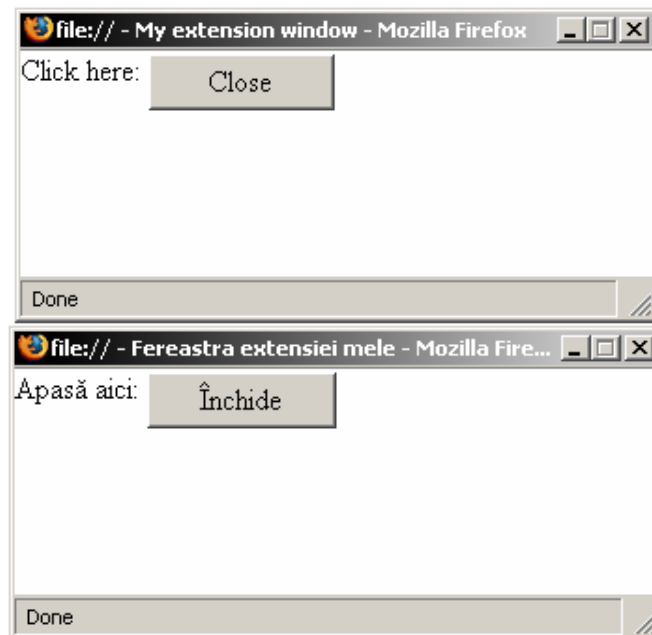


Figura 5. O aplicație cu două localizări

În fișierul *content/myextension/mainscript.js*:

```
function confirmClose() {
  var stringBundle = document.getElementById("myextension_bundle");
  if (confirm(stringBundle.getString("me.confirmClose")))
    window.close();
}
```

În fișierul *locale/en-US/myextension/mainwindow.properties*:

```
me.confirmClose = Are you shure?
```

În fișierul `locale/ro-RO/myextension/mainwindow.properties`:
`me.confirmClose = Sunteți sigur?`



Figura 6. Două dintre cele mai populare teme pentru Firefox: *BlackJapan* (sus) și *Noia* (jos)

4.4 Stiluri, teme, aspecte

Aspectul unei aplicații *Mozilla* nu este stabilit în momentul compilării platformei. Acesta este creat utilizând imagini, *CSS* și *XSLT*, și este personalizabil prin intermediul temelor grafice.

Această flexibilitate din punctul de vedere al aspectului apare deoarece limbajul de descriere a interfeței, *XUL*, fiind limbaj din familia *XML*, permite, nativ, stilizarea prin transformări *XSLT* sau prin aplicarea de foi de stiluri *CSS*. Momentan sunt suportate specificațiile *CSS* 1.0, 2.1 aproape complet, și deja o parte

considerabilă din CSS 3.0, permițând crearea de efecte vizuale avansate pentru o aplicație.

O foaie de stiluri este referențiată tot prin spațiul de nume *chrome*, astfel încât schimbarea temei grafice să fie transparentă din punctul de vedere al programatorului. La fel se pot accesa și imaginile ce apar într-o interfață, permițând redefinirea lor într-o altă temă.

O extensie poate reutiliza stiluri definite la nivel global pentru a facilita dezvoltarea ulterioară a unei teme grafice.

```
<?xml-stylesheet href="chrome://global/skin" type="text/css"?>
<?xml-stylesheet href="chrome://myextension/skin/window.css"
  type="text/css"?>
```

5. CONCLUZII

Mozilla Application Framework este un cadru suficient de matur pentru a permite dezvoltarea de aplicații multiplatformă, cu o împărțire bine definită în module, cu suport nativ pentru localizare, internaționalizare, extensibilitate și asociere de proprietăți de stil. Cele mai concludente exemple sunt navigatorul *Firefox* și clientul de poștă electronică și știri *Thunderbird*, a căror popularitate este în continuă creștere.

Cadrul de lucru *Mozilla* introduce un nivel avansat de personalizare, dincolo de selectarea unei teme grafice sau a unui set de funcționalități deja existente: permite unui utilizator ce deține cunoștințe legate de tehnologiile implicate să-și construiască propriile extensii, care să răspundă cel mai bine cerințelor sale.

O extensie, compusă din **conținut** (*XUL*, *XHTML*, *JavaScript*), **localizare** (*DTD* și *properties*) și **aspect** (*CSS*, *PNG* – recomandat, *XSLT*), se bazează pe tehnologii puternice, dar ușor de înțeles și manipulat.

Ușurința cu care se dezvoltă o extensie pentru aceste aplicațiile mai sus amintite a atras deja mulți programatori, în momentul actual existând peste 850 de extensii oficiale pentru *Firefox* și peste 150 pentru *Thunderbird*.

Referințe

***, *ECMA International*: <http://www.ecma-international.org/>

***, *ECMAScript for XML (E4X) Specification, 2nd edition*:
<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-357.pdf>

***, *ECMAScript language Specification, 3rd edition*:
<http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf>

***, *Mozilla Foundation*: <http://www.mozilla.org/>

***, *Mozilla Suite*: <http://www.mozilla.org/products/mozilla1.x/>

***, *Pagina Wikipedia despre istoria suitei Mozilla:*

http://en.wikipedia.org/wiki/History_of_Mozilla_Application_Suite

***, *SeaMonkey homepage:* <http://www.mozilla.org/projects/seamonkey/>

***, *XUL – specificația 1.0:* <http://www.mozilla.org/projects/xul/xul.html>

***, *XUL – pagina proiectului:* <http://www.mozilla.org/projects/xul/>

***, *XUL Planet:* <http://www.xulplanet.com/>

Capitolul 7

VALENȚE CSS

Marta Gîrdea

Facultatea de Informatică, Universitatea Alexandru Ioan Cuza din Iași
Str. General Berthelot, nr. 16, Iași, România
marta@info.uaic.ro – <http://purl.org/net/marta>

Rezumat. CSS este la momentul actual exploatat doar superficial, principalele motive ale acestui fenomen fiind cunoașterea insuficientă și suportul precar pentru această tehnologie oferit de o parte din navigatoare. Vom descrie în continuare, prin relativ scurte exemple demonstrative, câteva aspecte legate de CSS care subliniază puterea efectelor acestuia asupra documentelor Web din diverse puncte de vedere, precum și posibilitatea substituirii tehnicilor folosite curent (*e.g.*, JavaScript, Flash) pentru obținerea unor efecte grafice și dinamice avansate prin foi de stiluri în cascadă.

Cuvinte-cheie: CSS, efecte speciale, dinamicitate, conținut și formă.

1. INTRODUCERE

CSS (*Cascading Style Sheets* – foi de stiluri în cascadă) este un limbaj standardizat de Consorțiul Web pentru descrierea aspectului unui document *HTML*, *XHTML*, *SVG* sau, mai general, *XML*.

Acest limbaj a apărut în contextul creșterii popularității spațiului *WWW*. Constatându-se că, într-o oarecare măsură, imaginea și prima impresie pot face diferența dintre succes și eșec, a crescut cererea de modalități de îmbogățire a aspectului unei pagini. Inițial, s-a mers în direcția introducerii de noi elemente și atribute în formatul *HTML*, apărând elemente precum *color*, *font*, *center*, ne-numărate atribute de formatare a textului, a culorii sau a poziționării. Dezvoltatorii de navigatoare au introdus propriile elemente și atribute, nestandardizate, pentru a atrage cât mai mulți admiratori.

Astfel s-a ajuns la încărcarea nejustificată a documentelor cu marcaje care țin strict de modul de prezentare, deseori cantitatea de informație efectivă dintr-o pagină reprezentând o parte infimă din documentul *HTML*. Din cauza utilizării unor marcaje nevalide, un număr mare de pagini nu erau afișate corect decât în anumite navigatoare sau în anumite versiuni de navigatoare.

Soluția a fost introducerea unui limbaj care să separe conținutul de prezentare, astfel încât textul în format *HTML* să fie încadrat doar de elemente de organizare a conținutului, delimitarea în paragrafe, utilizarea titlurilor de secțiuni, marcarea conținutului cu semnificație particulară (abrevieri, citate, formulare), iar partea de prezentare, conținând tot ceea ce se putea specifica prin elementele și atributele de formatare din *HTML*, să apară în module separate. Acest limbaj a fost numit *Cascading Style Sheets – CSS*.

Fiecare fișier CSS este o „**foaie**” de reguli de formatare ce definesc „**stilul**” de afișare a unei pagini. Aceste reguli se pot suprapune, modificând „**în cascadă**” elementele unui document.

Prima versiune a limbajului CSS a fost lansată ca standard recomandat al Consorțiului *Web* pe data de 17 decembrie 1996, specificând cu precădere proprietăți de bază legate de cromatică, dimensionare și aliniere. Această versiune a încercat să elimine elementele și să înlocuiască atributele de formatare din *HTML*, în general păstrându-se numele atributelor ca nume al proprietăților CSS. În 1999 această specificație a fost revizuită și corectată pentru a se alinia la comportamentul majorității implementărilor existente.

Versiunea 2.0 a standardului, lansată la 12 mai 1998, a adus multe extensii limbajului „rudimentar” de precizare a aspectului unui document. La momentul curent, algoritmi implicați în determinarea formătărilor corecte nu sunt însă complet implementați în nici un navigator, în unele cazuri suportul pentru mare parte din specificație fiind cu precădere defectuos sau inexistent.

În prezent, se încearcă revizuirea acestui standard în specificația CSS 2.1, aflată în stadiul de ciornă de lucru („*working draft*”).

Versiunea CSS 3.0 aduce o abordare modularizată a specificației. În cadrul acestei variante nu mai există un singur standard, ci o colecție de standarde interconectate, fiecare fiind destinat unui singur aspect al specificației CSS 3.0. Există module generale (care tratează sintaxa CSS, algoritmul de suprapunere a regulilor, selectorii utilizați, gramatica valorilor, unitățile de măsură acceptate etc.), module de stilizare (pentru culoare, fundal și margini, text, tabele, formulare etc.), module pentru diverse medii de afișare (aural, paginat, ecran etc.) și module specializate pentru un anumit limbaj XML (*SVG*, *Ruby*, *MathML* și altele). Majoritatea acestor module sunt în curs de definire, multe dintre ele nefiind încă abordate. Cu toate acestea, versiunile recente ale navigatoarelor moderne implementează părți din modulele mai dezvoltate.

Deși, așa cum am precizat mai sus, CSS poate fi combinat cu diverse limbaje din familia XML, ne vom referi în continuare doar la aplicarea de foi CSS pentru documente (*XHTML*), ca posibilă alternativă, într-un viitor mai mult sau mai puțin apropiat, la tehnicile utilizate în mod frecvent la momentul curent pentru poziționare, interactivitate sau efecte grafice avansate.

2. MOTIVAȚIE

CSS a fost creat pentru a permite separarea formei de conținut. Necesitatea acestei separări este evidentă: documentele ar trebui să fie organizate semantic, și nu vizual. Deși interfața grafică pare a avea o importanță deosebită, nu trebuie uitat faptul că informația transmisă este scopul principal al unui document pe *Web*, iar modul de afișare este un aspect secundar și trebuie tratat ca atare.

Accesibilitate

Un document a cărui structură se focalizează pe conținut și nu pe grafică este mai ușor înțeles dintr-un navigator text, fără suport pentru imagini, animații și culori. O abordare greșită, frecvent întâlnită în documentele *Web*, este organizarea tabelară a documentului, cu un număr mare de celule care conțin doar imagini, uneori chiar imagini ce substituie textul. Dintr-un navigator text, un astfel de document este aproape de neînțeles pentru utilizator: în locul imaginilor apar legături pentru descărcarea acestora, celulele nu își păstrează dimensiunea din varianta grafică, textul devenind astfel greu de regăsit.

În plus, oricât de plăcut vederii ar părea un document în forma bogată în detalii grafice, poate constitui o adevărată provocare pentru cei cu deficiențe vizuale. În cazul în care forma de prezentare este inclusă în document, extragerea informației utile și afișarea într-un mod mai vizibil poate fi dificilă; însă, pentru documentele organizate pur semantic, foaia de stiluri poate fi ușor dezactivată. Mai mult, utilizatorul ar putea opta pentru o foaie de stiluri particulară, ce oferă un contrast puternic și o dimensiune mai mare a caracterelor, cu excluderea completă a imaginilor din document. Așadar, un document ar trebui să fie la fel de inteligibil și fără organizarea grafică a acestuia, și fără imaginile decorative.

Flexibilitate: medii de prezentare și stiluri alternative

CSS 2.0 a introdus noțiunea de „mediu de prezentare”, ce se referă la modul prin care este accesat un document: ecran (*screen*) pentru documentele vizualizate pe un monitor obișnuit, imprimat (*print*) pentru documentele imprimate, auditiv (*aural*) pentru documentele citite de un program de citire a ecranului, proiectat (*projector*) pentru documentele afișate cu ajutorul unui proiector etc. Se pot crea mai multe moduri de prezentare, în funcție de mediu. Pentru afișarea într-un navigator se poate concepe o prezentare bogată în efecte grafice (organizarea neliniară a conținutului, o cromatică atractivă, imagini statice și animate – deși abundența nu este recomandată). Pentru imprimare se va prefera, din motive evidente, un stil curățat de elemente pur decorative; de menționat că neprecizarea stilului pentru acest mediu atrage o formatare implicită, rezultatul fiind imprezvizibil și uneori inestetic. Un stil poate fi creat pentru cititoarele de ecran; în cadrul acestuia se poate stabili ordinea în care sunt citite părțile componente ale documentului și, mai mult, se poate atașa „personalitate” diverselor părți ale unui document. Un exemplu grăitor în acest sens este atașarea de

stiluri aurale pentru un document conținând fragmente din piese de teatru, unde paragrafele asociate fiecărui personaj pot fi citite de către o voce corespunzătoare.

CSS permite nu numai definirea de stiluri diferite pentru medii diferite, dar și definirea unor stiluri alternative pentru același mediu. Majoritatea navigatoarelor permit alegerea dintr-o colecție de stiluri a celui dorit. Astfel, se pot defini diverse „teme” de prezentare a informației, atât ca aspect vizual general, cât și privind organizarea categoriilor de informații (*layout*).

Eliminarea redundanței

Utilizarea atributelor de formatare din *HTML 3*, de exemplu, presupune precizarea acestora în mod repetat pentru fiecare element ce trebuie particularizat, ceea ce induce un grad ridicat de redundanță. În CSS este suficientă specificarea câte unei singure regulă întreaga colecție de elemente, ceea ce duce la diminuarea cantității de date transmise de la server la client. În cazul în care mai multe documente utilizează aceeași foaie de stiluri, regulile de formatare vor fi menținută în *cache* și nu vor fi retransmise, reducând și mai mult cantitatea de date transmise și timpul de încărcare a unei pagini.

Nimic nu e perfect...

Reticența față de trecerea la un design axat pe foi de stiluri CSS este cauzată de implementările diferite sau incomplete din partea diverselor navigatoare, chiar și pentru tehnicile simple din CSS 2.0. La rândul lor, dezvoltatorii de navigatoare nu investesc un efort prea mare în implementarea specificațiilor CSS deoarece doar un număr până de curând foarte mic de creatori de pagini *Web* au trecut la design bazat pe CSS. Exemplele descrise în continuare necesită unele reguli speciale pentru mascarea lacunelor de implementare sau pentru a asigura un aspect independent de navigator; din cauza implementărilor incomplete, unele tehnici mai avansate sunt total incompatibile cu anumite navigatoare.

Deși, din motivele mai sus amintite, nu pot fi complet utilizate în prezent, tehnicile descrise în secțiunea următoare – cu scop pur demonstrativ al puterii CSS și al modului în care poate influența impresia creată de o pagină – anticipază probabil direcțiile de viitor apropiat în folosirea stilurilor pentru paginile *Web*.

3. IMPACTUL FOILOR DE STILURI

3.1 Formă și conținut

Deja am menționat faptul că limbajul CSS a fost conceput pentru a separa forma de conținut. Separarea are loc doar la nivel de proiectare. În cadrul produsului final, forma și conținutul trebuie să conlucreze pentru atingerea scopului. În mod natural, o anumită prezentare poate evidenția semnificația conținutului, poate pune într-un anumit cadru tematic o colecție de documente.

Exemplul studiat în această secțiune este construit în jurul unui document XHTML ce conține prima strofă din poezia „Glossă” de Mihai Eminescu.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Glossă</title>
    <link rel="stylesheet" title="Normal Order" type="text/css"
      href="normal.css" />
    <link rel="alternate stylesheet" title="Reverse Order"
      type="text/css" href="reverse.css" />
  </head>
  <body>
    <div id="header">
      <h1 id="title"><span>Glossă</span></h1>
      <h4 id="author"><span>de Mihai Eminescu</span></h4>
    </div>
    <div class="poem">
      <p class="verse" id="verse1">
        Vreme trece, vreme vine<span class="punctuation">,</span></p>
      <p class="verse" id="verse2">
        Toate-s vechi și nouă toate<span
          class="punctuation">;</span></p>
      ...
    </div>
  </body>
</html>
```

Lipsit de stil, documentul este afișat ca în figura 1:

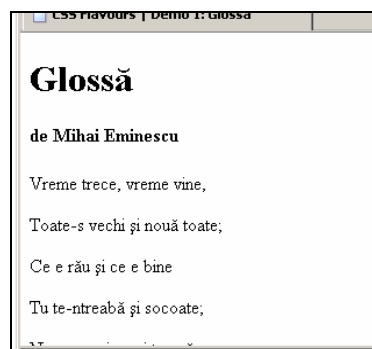


Figura 1. „Glossă” fără stiluri CSS aplicate

Prin aplicarea unui stil netrivial, care utilizează reguli simple din CSS 2, se obține o grafică mai sugestivă, într-o oarecare concordanță cu textul, după cum se poate vedea în figura 2.



Figura 2. Un cadru mai sugestiv pentru o poezie

```
body{
  /* Imaginea de fundal - hârtie veche */
  background: #363533 url(img/oldpaper.jpg) repeat-y fixed top center;
  color: #210;
  font-family: Georgia, serif;
  font-style: italic;
  /* Poezia va fi centrată și va avea o lățime fixă */
  width: 640px;
  margin: 0px auto;
  padding: 0px;
}
/* Titlul va fi înlocuit cu o imagine din CSS, astfel încât să poată
fi văzut și din navigatoarele ce nu pot afișa imagini */
#title{
  margin: 10px 96px 0px;
  padding: 0px 0px 8px 0px;
  background: transparent url(img/title.jpg) no-repeat top left;
  border-bottom: 1px solid;
  height: 121px;
}
#title span{
  display: none;
}
#author{
  text-align: right;
  font-size: 16px;
  margin: 1.33em 96px;
}
.poem{
  font-size: 24px;
```

```

    text-align: left;
    padding-left: 96px;
    height: 13.5em;
}
p.verse{
    font-size: 24px;
    line-height: 0em;
    padding: 0px;
    margin: 0px;
    margin-top: 1.5em;
}

```

Imagini în loc de text

Una dintre cele mai simple și eficiente modalități de a afișa texte având atașate stiluri este aceea de a folosi o imagine creată de un designer, întrucât gama de fonturi și de efecte grafice care pot fi utilizate cu siguranța de a fi recunoscute corespunzător pe partea de client este destul de redusă. O practică greșită, dar foarte des întâlnită, este aceea de a insera direct în documentul *HTML* imaginile în loc de text, ceea ce reduce mult nivelul de accesibilitate. Înlocuirea textului cu o imagine este destul de ușor de realizat din *CSS*, și are avantajul de a nu „stri-ca” semantica documentului. Cea mai simplă metodă este aceea de a folosi imaginea ca fundal al elementului înlocuit, împachetarea textului propriu-zis într-un element *span* și ascunderea acestuia din urmă.

```

HTML:
<h2 id="replacedElement"><span>Text ce va fi înlocuit</span></h2>
CSS:
/* Afișarea imaginii */
#replacedElement{
    width: 120px
    height: 80px;
    background: transparent url(replacing.png) no-repeat top left;
}
/* Ascunderea textului */
#replacedElement span{
    display: none;
}

```

Pagini centrate

Unul dintre șabloanele de proiectare a aspectului unei pagini presupune afișarea centrată și cu lățime fixă a unui document. În general acest lucru era obținut utilizând tabele, pentru care se pot preciza lățimea și alinierea în pagină. Tabelele sunt destinate includerii de *date tabelare*, și nu este recomandată utilizarea lor în scopul formătărilor vizuale. Efectul dorit se poate obține din *CSS*, fără a utiliza elemente de structurare auxiliare. Elementul cheie al acestei tehnici este setarea valorii *auto* pentru marginile laterale ale documentului, ceea ce are ca efect crearea unor margini egale, deci centrarea conținutului.

```

body{
    width: 640px;      /* Lățimea documentului */
    margin: 0px auto; /* Centrarea documentului */
}

```

Prin această tehnică se poate centra orice element bloc, nu doar întreaga pagină.

Reordonarea elementelor

După cum se știe, ultima strofă din „Glossă” conține aceleași versuri ca și prima, dar în ordine inversă, cu o punctuație diferită. Se poate utiliza documentul *HTML* ce conține prima strofă pentru afișarea ultimei, modificările având loc doar prin efecte *CSS*.



Figura 3. Ultima strofă, obținută din prima strofă prin *CSS*

Elementele bloc sunt poziționate astfel încât marginea superioară a unui element să fie adiacentă marginii inferioare a elementului anterior. Precizând margini negative, va rezulta că un rând se va afla deasupra marginii inferioare a rândului anterior. Dintre regulile specificate în stilul anterior, se observă că înălțimea unui rând (proprietatea *line-height*) a fost setată 0. Ca urmare, înălțimea calculată a unui rând de text (fără margini) va fi 0, dimensiunile „cutiei” (*box-ului*) unui vers fiind determinate doar de margini. Singura modificare necesară stilului anterior (pentru rearanjarea liniilor) este adăugarea următoarei reguli:

```
p.verse{
  margin-top: -1.5em;
}
```

Astfel, în ordinea „normală” a versurilor, dimensiunea unui rând va fi de 1.5 ori înălțimea textului, iar în ordinea inversă de -1.5 ori. Desigur, primul vers trebuie poziționat astfel încât să permită afișarea celorlalte cu evitarea suprapunerilor cu zona de titlu. Pentru aceasta se utilizează proprietatea *padding-top*, în-

cât marginea superioară a aparent primului vers să fie în același loc cu marginea primului vers din ordinea normală.

```
p#versel{
  padding-top: 13.5em;
}
```

Modificarea aparentă a conținutului

Pentru a obține ultima strofă a poeziei a fost necesară și schimbarea punctuației, problemă rezolvată doar prin tehnici CSS, și nu prin modificarea conținutului fișierului *HTML*. Această modificare se compune din ascunderea punctuației anterioare și afișarea noii punctuații.

Pentru ascunderea oricărui element este utilă proprietatea *display* din CSS, valoarea *none* a acestuia ascunzând complet elementele afectate.

Pentru afișarea unui conținut nou, există pseudo-elementele *:before* și *:after*, proprietatea *content* disponibilă aici putând genera conținut.

```
span.punctuation{
  display: none; /* Ascunderea vechii punctuații */
}
#versel:after{
  content: '.'; /* Afișarea noului semn de punctuație */
}
```

Acest conținut este doar desenat, nu adăugat la documentul inițial. În general, o foaie de stiluri CSS acționează doar la nivel de afișare, nu modifică efectiv conținutul, așa cum ar face un script. O acțiune de copiere a unei porțiuni din documentul astfel redat prin proprietăți de stil ne va releva varianta originală (atât din punctul de vedere al ordinii apariției versurilor, cât și al textului însuși), care poate diferi substanțial de ceea ce se vede în *browser*.

Utilitate

Exemplul de mai sus este un exemplu „jucărie”, care surprinde însă suficient de bine cum, via CSS, se pot rearanja diferitele părți ale documentului. Deoarece posibilitățile de modificare a poziției sunt vaste (folosind proprietăți ca *position*, *margin*, *float*), *layout*-ul tabelar *hard-coded* devine absolut inutil. Pe un document organizat liniar, se pot obține diverse variante de poziționare, într-un mod flexibil și elegant, cu un efect important asupra felului cum utilizatorul percepe conținutul.

3.2 Static și dinamic

Exemplul discutat în continuare vine în completarea celui anterior din punctul de vedere al poziționării părților componente ale documentului, ilustrând, în plus, modul cum o foaie de stiluri poate nu doar să afecteze modul de prezentare a unui document (partea statică), ci și să simuleze și o oarecare dinamicitate a documentului, desigur, la un nivel foarte redus. Elementele-cheie în acest scop sunt pseudo-selectorii (*:hover*, *:active*, *:target* etc.), care implică dinamicitate prin

dependența de interacțiunea cu utilizatorul (poziția cursorului *mouse*-ului, starea butoanelor acestuia etc.).

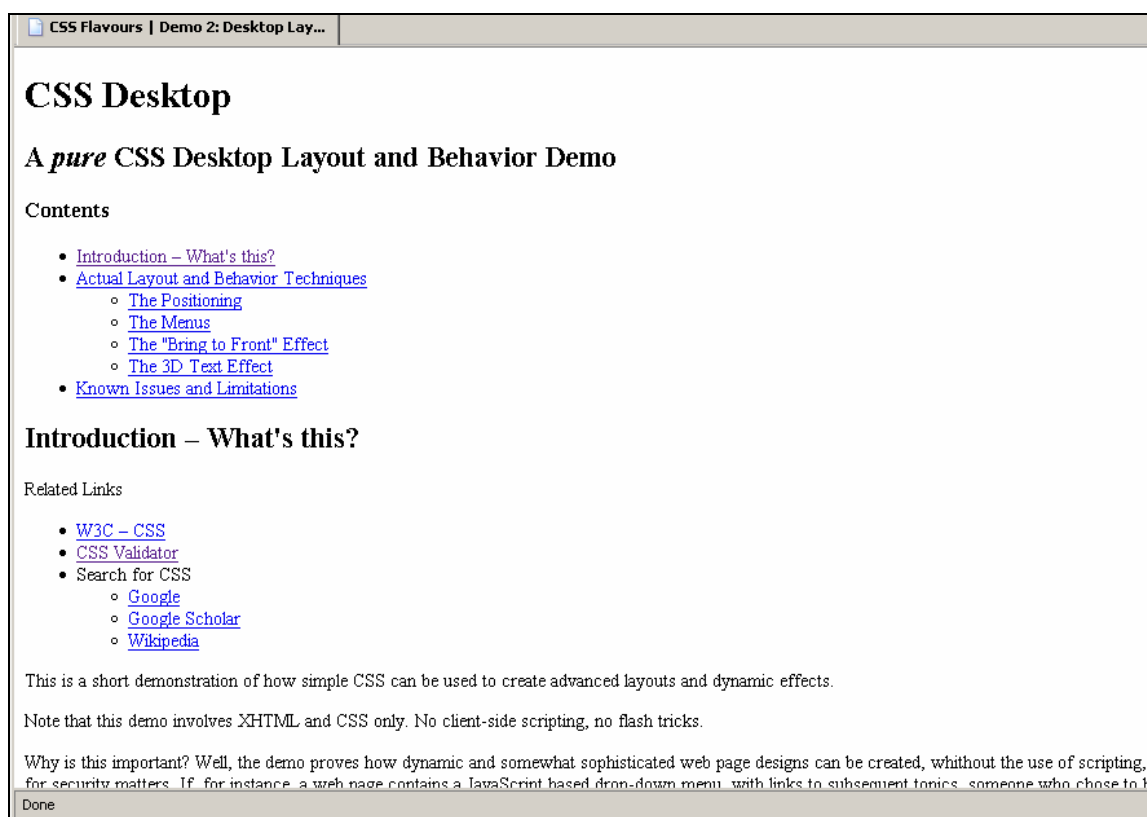


Figura 4. Un document HTML bine structurat, în absența stilului

Documentul HTML suport este structurat semantic (*heading-uri*, paragrafe, liste), fiind inteligibil și dintr-un navigator în mod text.

Acestui document i se aplică un stil ce creează impresia unui *desktop* (vezi figurile 5 și 6). Secțiunile documentului apar fiecare într-o fereastră proprie pe *desktop*, ferestrele au meniuri, cuprinsul documentului a fost transformat într-o bară de start, ferestrele pot să primească *focus*-ul, rămânând deasupra celorlalte.

Desigur, efecte similare (meniuri *popup*, schimbarea proprietăților legate de poziționare ale elementelor) se pot obține folosind *JavaScript*. Există însă argumente solide contra utilizării *scripting*-ului pe partea de client pentru aspecte legate de navigare. Dacă, de exemplu, o pagină conține un meniu expandabil realizat exclusiv în *JavaScript* sau în *Flash*, iar acest meniu este singura cale de acces spre paginile subsidiare, acestea nu vor putea fi accesate dintr-un navigator ce are dezactivat *JavaScript* din motive de securitate sau nu are instalat *plugin*-ul pentru *Flash*.

Pe de altă parte, nimeni nu va dezactiva CSS din motive de securitate, acesta fiind practic inofensiv. În plus, o abordare bazată doar pe *XHTML* și *CSS* pe partea de client va fi accesibilă și din navigatoare fără suport pentru foi de stiluri.

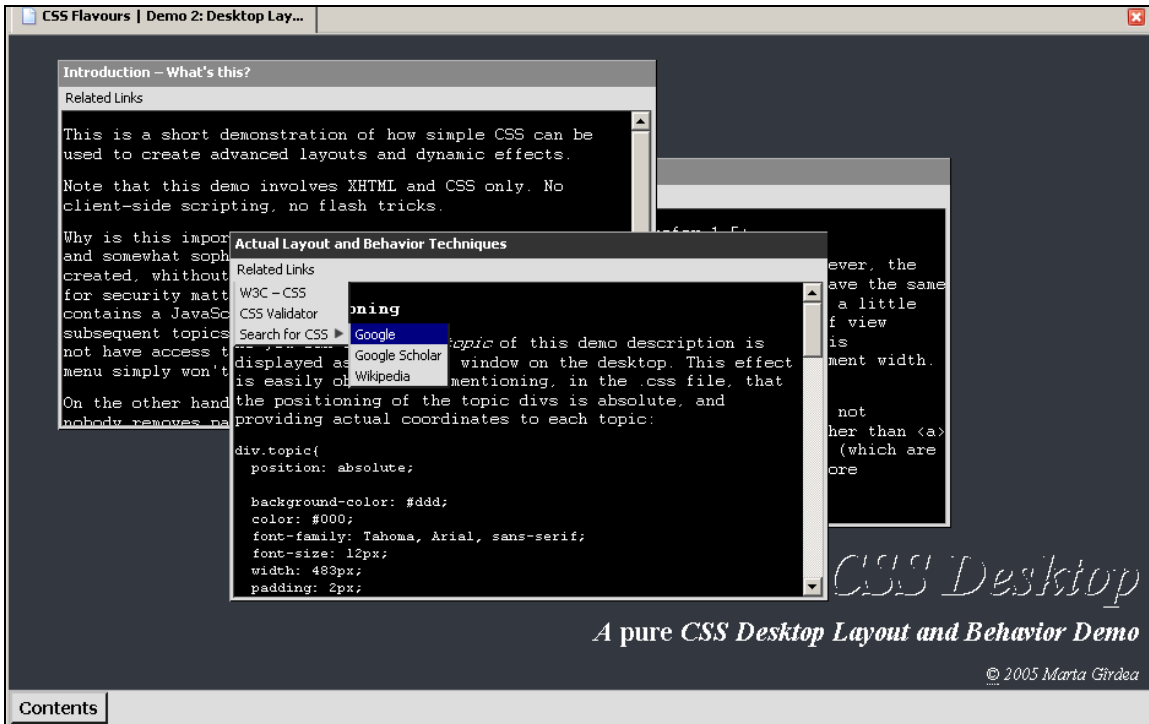


Figura 5. Impresia de aplicație desktop

Crearea de sub-ferestre

Fiecare secțiune logică din documentul HTML este afișat ca o sub-fereastră de dimensiune fixă, amplasată într-o poziție oarecare, spre deosebire de ordinea liniară impusă într-un document fără stil. Acest efect este foarte simplu de obținut și presupune:

- încadrarea fiecărei astfel de secțiuni într-un element *div* – acesta este un element generic care permite organizarea documentului în părți separabile;
- poziționarea absolută a fiecărui *div*;
- stabilirea dimensiunilor unei ferestre;
- precizarea faptului că o fereastră are dimensiune fixă, astfel încât în cazul în care textul conținut în fereastră este mare, să fie afișată o bară de derulare;
- stabilirea aspectului ferestrei.

```
div.topic{
  position: absolute; /* poziționare absolută */
}
div.contents{
  width: 480px; /* Stabilirea lățimii */
  height: 256px; /* și a înălțimii */
  overflow: auto; /* Apariția barei de defilare */
}
```

Meniuri expandabile

Ca organizare, meniurile sunt liste (elemente *ul*) imbricate. Afișarea acestora se face utilizând pseudo-selectorul *:hover*, combinat cu selectorul de descendent direct.

```

/* Meniuri și submeniuri */
.menuitems, .submenu {
    display: none;          /* Inițial meniurile sunt ascunse */
    position: absolute;
/* Elementele poziționate absolut nu sunt luate în considerație
    la calcularea dimensiunii elementului părinte */
}
/* Alinierea submeniurilor la: */
.submenu {
    top: 0px;      /* aceeași înălțime cu a elementului activator */
    left: 100%;   /* marginea dreaptă a meniului părinte */
}
/* Activatorul unui meniu */
.topicmenu > .menuname{
    cursor: pointer;
    display: table;
    padding: 4px;
}
/* O opțiune din meniu */
.menuitem{
    display: block;
    list-style-type: none;
    margin: 0px;
    padding: 0px;
}

/* Regulile de vizibilitate a meniurilor */
.menuname:hover + .menuitems, .menuitems:hover, .popup:hover >
.submenu {
    display: block;
}

```

Cea mai importantă regulă este cea de afișare a meniurilor și a sub-meniurilor. Primul selector alege meniul pentru care activatorul (elementul care *activează* sub-meniul) este în starea *hovered* (cursorul se află deasupra lui), selectând mai întâi elementele *hovered*, apoi vecinii imediați ai acestora. Al doilea selector menține vizibil meniul deschis atâta timp cât cursorul *mouse*-ului se află deasupra meniului. Al treilea selector afișează un sub-meniu atunci când cursorul se află deasupra elementului *li* ce conține sub-meniul și care are rolul de activator.

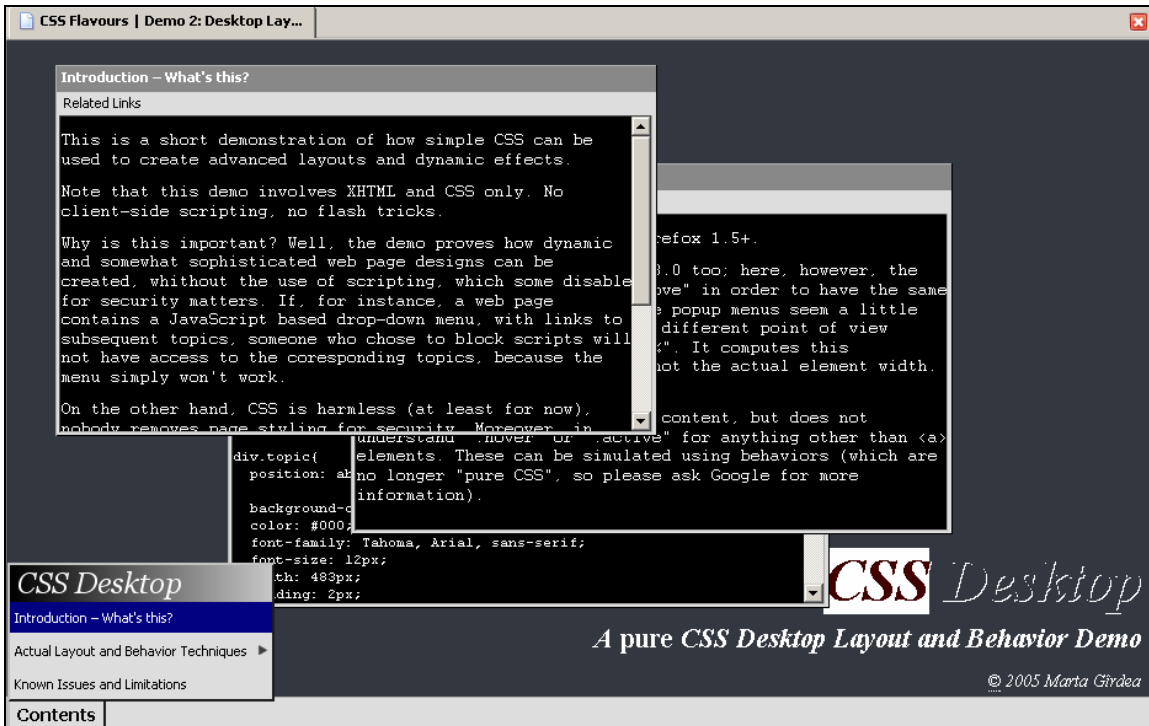


Figura 6. Alte aspecte ale acestui stil

Schimbarea ordinii ferestrelor

Cum era de așteptat, printr-un *click* pe o fereastră corespunzătoare unei secțiuni, aceasta va fi adusă deasupra celorlalte ferestre. Acest efect este posibil datorită proprietății *z-index*, ce permite specificarea unei ordonări pe axa *z*, cea care determină ordinea de afișare a elementelor. În mod normal, elementele sunt afișate în ordinea în care acestea se găsesc în document.

Prima regulă utilizată în acest efect este, deci, aceea de a specifica o valoare mai mare a proprietății *z-index* pentru fereastra deasupra căreia se află plasat *mouse-ul*.

O altă regulă trebuie să mențină o fereastră deasupra celorlalte atâta timp cât nu este apăsat butonul *mouse-ului*, adică nu se încearcă activarea altui element de pe *desktop* (pseudo-selectorul *:active*). Acest efect este obținut prin plasarea unui element transparent între fereastra activă și cele inactive, astfel încât ferestrele inactive să nu primească evenimentele *mouse-ului*.

În final, atunci când butonul *mouse-ului* este apăsat, se dorește ca și ferestrele din fundal să poată primi evenimente de la *mouse*, pentru a putea deveni active. Acest lucru este realizat prin ascunderea elementului transparent de blocare. Practic, evenimentul *click* pe o fereastră din fundal nu are ca efect apăsarea propriu-zisă pe acea fereastră, ci „spargerea” stratului transparent care capta evenimentele *mouse-ului*.

Revenind la prima regulă utilizată pentru acest efect, elementul menținut deasupra nu este fereastra vizibilă, ci elementul transparent din spatele acesteia.

Deoarece între cele două elemente există relația părinte-fiu, automat va fi adus deasupra și elementul fiu, fereastra.

```
div.topiccontainer{
  width: 100%;
  height: 100%;
  position: absolute;
  top: 0px;
  left: 0px;
  visibility: hidden;
}
div.topiccontainer:hover{
  z-index: 4;
  visibility: visible;
}
div.topiccontainer:active{
  visibility: hidden;
  z-index: 2;
}
div.topic{
  visibility: visible;
  position: absolute;
}
```

Meniul principal

Cuprinsul de la începutul paginii (meniul principal) a fost redat via proprietăți de stil printr-o bară de start. Aspectul de meniu expandabil a fost obținut utilizând aceleași tehnici ca și pentru meniurile ferestrelor.

Efectul de activare a ferestrelor prin apăsarea intrărilor din meniu a fost obținut cu ajutorul pseudo-selectorului *:target*. Acesta se referă la identificatorul de fragment ce poate fi precizat în URL-ul unui document. Astfel, „ținta” unei adrese identifică un element.

Legăturile din meniul principal duc fiecare către o fereastră. La „alegerea” unei intrări din acest meniu, una din ferestre devine „ținta” cerută. Atunci când utilizatorul alege o opțiune din meniul principal, nici una din ferestre nu va fi activă, deci ordinea de afișare va fi cea indicată de ordinea din documentul HTML, fiecare fereastră având valoarea implicită pentru proprietatea *z-index*, adică 0. Pentru a aduce deasupra fereastra „țintă”, vom utiliza o regulă specifică când o valoare strict pozitivă pentru proprietatea *z-index* a elementului *target*.

```
*:target{
  z-index: 1;
}
```

Texte tridimensionale

Un alt efect ilustrat de acest exemplu este acela de creare a elementelor aparent tridimensionale, fără implicarea imaginilor. Titlul documentului (*CSS Desktop*) este generat în acest mod. Specificația CSS 3.0 include proprietăți destinate creării unor efecte similare, care momentan nu sunt implementate.

Pentru a realiza acest efect, s-au urmat mai mulți pași:

- cu ajutorul pseudo-elementului *:before* este generat încă o dată același text cu al elementului inițial;
- textul generat va fi afișat ca un element bloc (implicit, se generează elemente *inline*), poziționat absolut;
- acesta va fi așezat astfel încât să apară puțin deplasat față de elementul inițial;
- textul generat și textul inițial vor avea culori diferite; în acest caz, textul inițial are aceeași culoare cu fundalul, iar textul generat este alb.

```
html > body #header h1{
  color: #33373f;
  font-size: 40pt;
  margin-top: 0px;
  line-height: 1em;
}
#header h1:before{
  content: "CSS Desktop";
  display: block;
  color: #FFF;
  margin-bottom: -1.02em;
  margin-right: -.03em;
}
```

Limitări

Deși efectele descrise par foarte utile și puternice, CSS prezintă o serie de limitări în acest context. De exemplu, o fereastră creată prin CSS poate rămâne activă atâta timp cât cursorul nu părăsește fereastra navigatorului. În momentul în care acest lucru se întâmplă, toate ferestrele vor reveni la ordinea inițială. Cauza este lipsa memoriei. Astfel, o fereastră activă se află pe stratul 4, o fereastră inactivă pe stratul 0; nu se poate incrementa poziția ferestrei active curente, fiind necesară o poziționare explicită. Se observă acest lucru și activând ferestrele în ordinea 3, 2, 1. Într-un mediu *desktop* real, ferestrele ar fi în ordinea, de sus în jos, 1 - 2 - 3. În mediul *CSS Desktop* însă ordinea este 1 - 3 - 2, deoarece fereastra a treia va fi deasupra celei de-a doua ferestre, atâta timp cât nu este activă.

O altă limitare este aceea că prin meniul de start doar legăturile direct către fereastră au efectul scontat; legăturile de pe al doilea nivel, care conduc spre părți din a doua fereastră, vor deplasa bara de defilare în interiorul ferestrei, astfel încât partea dorită să fie vizibilă, dar ordinea ferestrelor nu se schimbă. Acest lucru se întâmplă deoarece „ținta” este un element din interiorul ferestrei, și nu o fereastră.

3.3 Efecte speciale

Următorul exemplu este un alt document „jucărie”, fără conținut, al cărui unic scop este de a demonstra cum, jonglând cu imagini cu transparență *alpha*, diverse tipuri de poziționare a elementelor și proprietăți CSS 2 mai rar utilizate în practică (*clip*), se poate obține un efect de animație la derularea verticală.

În figurile de mai jos se pot urmări diverse faze ale acestei animații, poziția în cadrul documentului fiind sugerată de poziția barei de derulare vizibilă în partea dreaptă.

Elemente fixe

În CSS 2, pentru proprietatea *position* se poate asocia valoarea *fixed*. Elementele poziționate astfel păstrează aceeași poziție relativ la fereastră, și nu la document. Împreună cu proprietățile de dimensionare și determinare a poziției (*width, height, top, left, right, bottom*), anumite părți de document pot fi amplasate astfel încât să ocupe mereu aceeași zonă a ferestrei.

În acest exemplu, norii și clădirile, definite prin elemente *div* ce au imaginile corespunzătoare ca fundal, sunt poziționate fix (se observă că nu își schimbă poziția la derulare).

```
.elementeFixe{
  position: fixed;  bottom: 0px;  left: 0px;
  width: 100%;  height: 100%;
  background: transparent repeat-x center bottom;
}
```

Imagini transparente suprapuse

Animația cerului este compusă din mai multe straturi suprapuse, având următoarea structura de la stratul cel mai din spate la cel mai din față:

- culoarea cerului noaptea este dată de culoarea de fundal a documentului (elementul *body*);
- culoarea soarelui este dată de o imagine gradient galben (sus) spre roșu (jos); această imagine este fixă și este vizibilă prin decupările din straturile superioare, ce creează culoarea cerului;
- culoarea cerului ziua (exceptând norii) este dată de o imagine de fundal opacă, având decupată în mijloc o formă de disc;
- ceața roșiatică de la apus constă într-o imagine fixă;
- pentru a deveni vizibilă doar în momentul apusului, peste această imagine este suprapusă o imagine asemănătoare cerului, dar care are o bandă semitransparentă în jurul soarelui;
- norii din timpul zilei se obțin asemănător ceții; o imagine fixă și semitransparentă se află deasupra stratului anterior;

- imaginea ce permite afișarea norilor doar în timpul zilei este asemănătoare cerului, dar transparentă numai în partea de jos; tot această imagine ascunde și ceața după apusul soarelui.



Figura 7. Etape ale apusului de soare

Pentru elementele mobile s-a utilizat următorul stil:

```
.elementeMobile{
  position: absolute;
  top: 0px;
  width: 100%;
  height: 2000px;
  background: transparent repeat-x center bottom;
}
```

Ferestre glisante

Aprinderea luminilor la căderea nopții este realizată cu ajutorul proprietății *clip*. Este definită o zonă de vizibilitate într-un element mobil, ceea ce înseamnă că inclusiv zona de vizibilitate este mobilă și se va deplasa împreună cu bara de defilare. Acest element este fereastra glisantă (*sliding window*). În interiorul său este definit un sub-element cu poziționare fixă, având ca fundal imaginea ce trebuie să fie vizibilă numai în anumite momente. Imaginea respectivă va fi mereu în aceeași poziție, dar numai la trecerea ferestrei glisante pe deasupra ei va fi vizibilă. Acest element reprezintă ceea ce este vizibil prin fereastră.

```
.clippingContainer{ /* fereastra glisantă */
  position:absolute;
  top: 0px; left: 0px;
  width: 100%; height: 2000px;
}
.clippedContent{ /* peisajul mascat, fix */
  position: fixed;
  bottom: 0px; left: 0px;
  width: 100%; height: 100%;
  background: transparent repeat-x center bottom;
}
#streetlightsC{ /* luminile de pe stradă, fereastra mobilă */
  clip: rect(auto, auto, 1020px, auto);
}
#streetlights{ /* luminile de pe stradă, peisajul fix */
  background-image: url(img/streetlights.png);
}
```

În acest exemplu, luminile din oraș sunt realizate utilizând mai multe astfel de ferestre glisante, pentru a simula aprinderea aleatoare. Folosirea uneia singure ar fi generat o aprindere a luminilor de sus în jos. Fiecare „peisaj” utilizat constă în câteva ferestre luminate alese aleator, iar marginea de jos a zonei de tăiere (care definește momentul când devine vizibilă o asemenea imagine) se află la poziții diferite.

Integrarea într-un document

Deși în acest mod se pot crea efecte atractive fără implicarea unor tehnologii cum ar fi *Flash* sau *JavaScript*, efecte potrivite, de exemplu, pentru pagini personale, trebuie menționat că utilizarea acestor tehnici decorative poate implica o serie de probleme.

În primul rând, în documentul *XHTML* suport trebuie să existe un număr de elemente fără conținut, al căror unic scop este de a găzdui imaginile care compun animația dorită. Acest aspect contravine într-o oarecare măsură ideii de separare a formei de conținut.

O altă problemă importantă apare datorită suprapunerii de imagini cu transparență, cu diverse tipuri de poziționare. Aceasta implică, la derularea documentului, un efort sporit la desenare din partea navigatorului. Dacă acesta rulează pe o mașină cu resurse modeste, se pot crea aparente blocaje (navigato-

rul răspunde cu greu la cererea de derulare, fiind „ocupat” cu desenarea), ceea ce poate fi neplăcut pentru utilizator.

În consecință, aceste tehnici trebuie folosite cumpătat, trebuie evitate imaginile prea mari și structurile prea complexe.

4. DE LA IDEE LA PUNEREA ÎN PRACTICĂ

4.1 Cruda realitate

Am descris prin exemplele de mai sus puterea CSS ca specificație, utilitatea evidentă în documente și aplicații *Web*, precum și capacitatea acestuia de a constitui în anumite situații o alternativă mai sigură și inofensivă pentru *Flash* și *JavaScript*.

În realitate, între specificație și implementările efective există diferențe notabile, care împiedică, pentru moment cel puțin, exploatarea la maximum a valențelor CSS. Specificația CSS 1.0 este implementată aproape complet în cele mai multe dintre navigatoarele frecvent utilizate actual. Implementările pentru specificația CSS 2.0 variază puternic, de la „aproape tot” (*Firefox*, *Opera*, *Safari*), până la „strictul necesar” sau mai puțin (*Internet Explorer*).

Așadar, utilizarea de formătări ce implică CSS 2+ este primejdioasă, în sensul accesibilității. Paradoxul este acela că, deși conceput, ca orice alt standard al Consorțiului Web, în scopul independenței de instrumentul de navigare, CSS poate crea probleme majore tocmai în acest sens, din cauza inconsistenței implementărilor.

4.2 Posibile soluții

Desigur, se pot imagina soluții pentru problema menționată, fiecare dintre ele reprezentând însă un compromis.

O primă soluție este utilizarea, pentru formatare, doar a aspectelor prevăzute de specificațiile CSS implementate uniform. Aceasta asigură o afișare similară și corectă în navigatoarele vizuale, însă limitează considerabil orizontul opțiunilor.

Alte soluții constau în formătări sau chiar tehnologii diferite pentru navigatoare diferite, în funcție de nivelul de înțelegere al fiecăruia. Fie se pot trimite, la nivel de server, foi de stil în funcție de clientul care a lansat cererea, fie – în cadrul aceleiași foi de stil – se pot exploata anumite lacune ale navigatorului pentru mascarea altora. Spre exemplu, versiunile navigatorului *Internet Explorer* vor ignora regulile care implică selectori de tip descendent direct, această porțiune din specificație nefiind implementată. Așadar, se poate defini un stil de afișare înțeles de orice navigator, și, în cadrul aceleiași foi, se pot suprascrise diverse porțiuni, folosind reguli ce conțin selectori mai complecși. Aceste porțiuni pot

conține formătări avansate și vor fi ignorate de unele navigatoare, respectiv aplicate de altele.

Un caz concret este cel ilustrat de codul de mai jos, extras din exemplul CSS *Desktop* discutat în secțiunea anterioară.

```

/* Fundalul desktop-ului */
body{
  background-color: #33373f; color: #FFF;
  overflow: hidden;
}
/* Titlul (CSS Desktop) */
#header h1 {
  margin: 2px; padding: 4px 8px;
  text-align: right;
  /* Se mosteneste culoarea (#FFF) */
}
/* Navigatoarele care inteleg pseudo-elementul :before
vor crea efectul de text 3D */
html > body #header h1{ /* Selector ignorat de IE */
  color: #33373f; /* aceeași culoare ca și cea de fundal;
                  fara selectorul html > body, acest text ar fi
                  devenit invizibil in Internet Explorer */

  font-size: 40pt;
  margin-top: 0px;
  line-height: 1em;
}

/* Crearea impresiei de contur tridimensional alb: */
#header h1:before{ /* selector ignorat de IE */
  content: "CSS Desktop";
  display: block;
  color: #FFF;
  margin-bottom: -1.02em;
  margin-right: -.03em;
}

```

Desigur, un astfel de truc este o soluție temporară. O eventuală lărgire a gamei de prevederi ale specificațiilor CSS implementate incluzând selectorii de tip descendent direct, dar nu și alte aspecte implicate într-un *design* ca și cel de mai sus îi va anula efectul.

O consecință directă a abordării acestor ultime soluții este afișarea diferită în navigatoare cu grade diferite de înțelegere a specificațiilor CSS, aspect nu întotdeauna dezirabil. De pildă, în cazul meniurilor expandabile descrise în cadrul aceluiași exemplu, ignorarea regulilor referitoare la afișarea meniurilor va face imposibilă navigarea. Apare, deci, exact problema care se dorea evitată prin abordarea CSS. O variantă de îmbunătățire constă în combinarea soluțiilor de mai sus cu exploatarea de facilități specifice puse la dispoziție de navigator; în cazul de față poate fi vorba de utilizarea de *behaviors* pentru *Internet Explorer* pentru a obține același efect ca al regulilor CSS ai căror selectori implică *hover* sau *active*, necunoscute acestui navigator.

5. CONCLUZII

CSS a devenit o specificație puternică, ale cărei capacități depășesc cu mult formatarea de documente la nivel de font și cromatică. Oferă posibilitatea de a da documentului formatat un aspect complet nou și neașteptat de complex, de la poziționări pline de originalitate până la structuri dinamice de genul meniurilor expandabile, de la efecte speciale de fundal la efecte de schimbare a sensului conținutului afișat. Utilizarea CSS permite formatarea documentelor într-o manieră elegantă și flexibilă, iar prin efectele multiple pe care le poate avea asupra acestora devine un concurent important al unor tehnologii mai complexe.

Deși la momentul actual exploatarea ultimelor versiuni poate fi problematică din cauza implementărilor defectuoase sau insuficiente în cadrul navigatoarelor, CSS merită studiat la un nivel avansat și luat în considerare ca unul dintre actorii principali în *Web design*.

Referințe

- Cederholm, D., *Bulletproof Web Design : Improving flexibility and protecting against worst-case scenarios with XHTML and CSS*, New Riders Press, 2005
- Cederholm, D., *Web Standards Solutions: The Markup and Style Handbook*, Friends of ED, 2004
- Shea, D., Holzschlag, M. E., *The Zen of CSS Design : Visual Enlightenment for the Web (Voices That Matter)*, Peachpit Press, 2005
- Zeldman, J., *Designing with Web Standards*, New Riders Press, 2003
- ***, *CSS/Edge*: <http://www.meyerweb.com/eric/css/edge/>
- ***, *CSS Zen Garden, The beauty of CSS design*: <http://www.csszengarden.com>
- ***, *W3C – Specificațiile CSS*: <http://www.w3.org/Style/CSS>
- ***, *W3C – Validator CSS*: <http://jigsaw.w3.org/css-validator/>

