Tehnologiile Internet Explorer

Aplicații intranet bazate pe Internet Explorer — *Marian Vețeanu*

Care este browserul dv. preferat? Probabil că unii dintre cititori vor răspunde *MS Internet Explorer*, alții *Netscape Navigator* sau *Communicator* iar o altă parte vor spune că au ales *Lynx*, *Opera*, *Arena* sau propriul browser, grăbindu-se să argumenteze fiecare de ce browserul său este cel mai bun. Indiferent de ce parte sunteți, vă recomand să citiți următoarele rânduri în care încerc să fac lumină asupra unora din tehnologiile MS introduse în browserele Internet Explorer 4, 5 și 5.5.

Începutul Web-ului

Astăzi puțini oameni își mai amintesc acele zile întunecate în care web-ul nu era nimic altceva decât un mediu de prezentare a unor documente scrise într-un limbaj de formatare a textului, independent de platformă, numit *HTML* (*Hyper Text Markup Language*). Tot ce trebuia să facă un utilizator era introducerea în browser a adresei unei astfel de pagini. Serverul de web primea cererea și-i returna utilizatorului pagina solicitată. După consultarea ei se putea accesa o altă pagină prin urmarea eventualelor link-uri prezente în pagina precedentă.

A urmat apoi a doua etapă din istoria Web-ului și anume generarea dinamică a paginilor web în momentul solicitării lor prin folosirea de programe executabile, externe serverului de web, apelate printr-o interfață simplă, bine definită, denumită *CGI* (*Common Gateway Interface*). Conceptul de *pagini dinamice* desemna în acele vremuri doar acțiunile efectuate de server în vederea producerii de conținut variabil, clientul rămânând în continuare un simplu program de formatare și afișare a textului.

Începutul DHTML-ului

În prezent, orice utilizator care navighează pe web se așteaptă să vadă pagini dinamice, pline de animații și interactivități, în locul unor simple documente statice. Conceptul de *pagină dinamică* s-a extins, el desemnând în momentul de față atât acțiunile pe care le efectuează serverul de web cât și cele efectuate de browser în vederea prezentării de conținut non-static.

Interactivitățile client își au începutul odată cu versiunea 2 a browserului *Navigator* când firma *Netscape* a introdus limbajul *JavaScript*. Deși un limbaj de scripting destul de puternic, JavaScript nu și-a putut arăta în primele navigatoare adevărata lui putere datorită unui acces mic la obiectele din interiorului paginii HTML. Astfel, principala utilizare a lui în browserele de acum câțiva ani a fost validarea formurilor înainte de trimiterea lor la server. Această situație s-a schimbat odată cu apariția *DHTML* (*Dynamic HTML*). Din nefericire, DHTML este înțeles diferit de browserele *Netscape Communicator* și *Microsoft Internet Explorer*. În articolul de față voi încerca să prezint DHTML din perspectiva Microsoft (deci a navigatorului *Internet Explorer*), existând o serie de argumente în favoarea acestuia așa cum va reieși și din informațiile prezentate în continuare.

DHTML-IE și intraneturile

Primul browser care a reușit să schimbe imaginea generală despre ceea ce este sau trebuie să fie un navigator web a fost *Internet Explorer 4*. El a fost urmat de *Internet Explorer 5* și la puțin timp de *IE5.5*. Principala și cea mai interesantă facilitate a browserelor *Microsoft* este oferirea unui *DOM* (*Document Object Model*) mult mai complex decât al celorlalte programe de pe piață. Acesta permite limbajelor de scripting (*JavaScript*, *Jscript*, *VBScript*) să acceseze aproape fiecare element vizibil (sau invizibil) din pagină oferindu-se în acest mod posibilitatea de schimbare a conținutului și după ce pagina s-a încărcat. Și astfel a apărut DHTML care nu este altceva decât ansamblul HTML 4 + DOM + Limbaje scripting.

Așa cum spuneam și mai devreme, DHTML nu mai este universal recunoscut de browsere cum este părintele său HTML. Acest lucru determină pe programatorii și designerii web care doresc să profite de această tehnologie să-și construiască paginile în mai multe variante, de obicei pentru browserele Communicator și pentru Internet Explorer. Cu toate acestea, firma Microsoft împinge din ce în ce mai mult facilitățile oferite în domeniul DHTML de către IE domeniul vizat nefiind *Internetul* ci mai degrabă *intraneturile,* care se consideră a fi, în general, rețele omogene la care nu se mai pune problema compatibilităților cu diversele platformele. Pe o astfel de rețea se poate opta pentru folosirea unui anumit tip de browser, ca de exemplu cel de la Microsoft. Având certitudinea că toți utilizatorii vizați folosesc browserul potrivit, nu mai apare nici o piedică în exploatarea la maximum a facilităților DHTML.

Articolul de față încearcă să prezinte aceste facilități din prisma programatorului de aplicații și a nu a designerului web. Deoarece majoritatea care își îndreaptă atenția către DHTML o fac în ideea realizării de efecte grafice și de animație mai deosibite s-a generat proasta concepție asupra destinației și scopurilor DHTML. Ca suport pentru articolul de față vom crea pas cu pas o aplicație DHTML cu o interfață similară aplicațiilor Windows.

Proiectarea aplicației DHTML

Iată cum ar suna tema de proiectare pentru o asemenea aplicație:

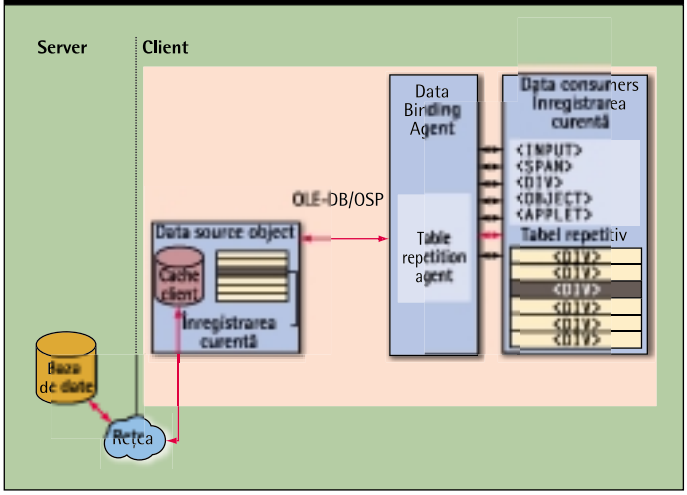
Se cere realizarea unei aplicații educaționale care să ruleze pe o rețea Windows a unei instituții, care să ofere posibilitatea utilizatorilor de a susține teste psihologice. Se vor avea în vedere următoarele aspecte:
(1) administrare zero pe client;
(2) solicitarea minimă a rețelei;
(3) aspect similar programelor obișnuite Windows.

Aceste cerințe fac trimitere directă către folosirea ca platformă de dezvoltare a aplicației navigatorului IE. Întrucât vom încerca exploatarea la maximum a browserului, ne vom alege ca versiune minimă IE5. Cu alte cuvinte cerința necesară și suficientă pentru rularea aplicației o constituie disponibilitatea browserului amintit. Aceasta nu ar trebui sa fie o problemă în condițiile date deoarece acesta vine înglobat în sisteme precum *Windows 98*, *Windows 2000* sau *Me* (*Windows Me* conține chiar versiunea mai nouă IE5.5) iar pe celelalte sisteme Windows poate fi instalat cu ușurință.

Administrare zero pe client. Făcând alegerea pentru acest navigator 1/3 din cerințele aplicației au și fost rezolvate. Prin încărcarea paginilor ce formează aplicația direct de pe serverul de web s-a realizat prima condiție și anume: *administrare zero pe client*. Deși nu pare aceasta ar putea fi considerată cea mai importantă cerință a aplicației industria software actuală tinzând spre realizarea de produse ce necesită *administrare zero*. Să presupunem că peste o lună dorim să instalam o nouă versiune a aplicației noastre. Aceasta presupune doar instalarea pe server, clienții nesuferind nici o muncă administrativă.

Solicitarea minimă a rețelei. Următoarea cerință ca importanță o constituie: *solicitarea minimă a rețelei.* Acest lucru poate fi realizat prin

Figura 1: Arhitectura Data Binding



minimizarea excursiilor la server și folosirea a cât mai multor operații efectuate pe client. Microsoft a prevăzut această problemă și a adăugat modelului HTML o nouă facilitate și anume *Data Binding*.

Ce înseamnă *Data Binding*? În loc ca paginile să vină generate de către server în funcție de cerințele clienților, Internet Explorer 4.0 introduce conceptul de legare a datelor pe client prin intermediul unor obiecte ActiveX integrate în browser cunoscute sub numele de DSO (*Data Source Objects*). Se asigură în felul acesta o separare a datelor pure de pagina HTML și totodată o creștere a performanței în ceea ce privește timpul de încărcare și afișare a paginilor. Datele sunt descărcate asincron de către clienți și apoi prin prelucrări într-unul din limbajele de script client se modifică conținutul paginii HTML chiar după ce aceasta a fost afișată integrându-se în ea datele primite. Totodată se reduce și numărul de excursii la server deoarece datele existând deja pe client, acestea pot fi folosite de ori de câte ori este nevoie fără a mai genera un nou transfer în rețea.

Data binding constă dintr-o arhitectură compusă din patru componente principale:
• Data Sourse Objects – DSO (Surse de date)
• Data Consumers (Consumatori de date)
• Binding agent (Agent ce se ocupă cu legarea datelor)
• Table repetition agent (Agent ce leaga datele la elemente HTML ce conțin elemente repetitive – de ex. tabele)

Data Source Objects pune la dispoziție datele în cadrul unei pagini web, elementele *Data Consumers* afișează aceste date iar elementele *Agent* au grijă să sincronizeze permanent consumatorii cu obiectele DSO.

Pentru a susține *Data Binding*, *Microsoft* a extins modelul HTML prin adăugarea anumitor elemente (așa numiții consumatori de date), câteva atribute suplimentare necesare pentru legarea lor la sursele de date. De fapt, majoritatea elementelor HTML au primit aceste atribute enumerate în continuare:
`DATASRC` – specifică obiectul DSO la care se leagă consumatorul;
`DATAFLD` - identifică coloana din cadrul înregistrării la care este legat consumatorul (datele sunt prezentate de DSO în format tabular);
`DATAFORMATAS` - specifică modul în care trebuie afișate datele;
`DATAPAGESIZE` - specifică numărul de înregistrări care sunt afișate la un moment dat.

O declarație HTML care să includă atributele prezentate poate arăta în felul următor:

```
<SPAN DATASRC=#agendatelefonica DATAFLD=nume></SPAN>
```

În cazul consumatorilor repetitivi (de ex. tabelele HTML), declarația se poate scrie astfel:

```
<TABLE DATASRC=#agendatelefonica>
   <TR><TD><SPAN DATAFLD=nume></SPAN></TD></TR>
   <TR><TD><SPAN DATAFLD=prenume></SPAN></TD></TR>
</TABLE>
```

În primul exemplu de sincronizarea datelor se ocupă agenții simpli de legare (*Data Binding Agents*) iar în al doilea exemplu sarcina sincronizării între elementele vizibile și sursa de date o au agenții repetitivi (*Table Repetition Agents*). Acești agenți lucrează permanent în background pentru a nu se pierde desincronizarea.

Să vedem acum ce este de fapt un *Data Source Object*. Aceasta este o denumire generică pentru unul din multele obiecte de acest fel incluse în Internet Explorer. Acestea includ:
• Tabular Data Objects (TDC)
• Remote Data Services (RDS)
• JDBC Data Source Applet
• XML Data Source
• MSHTML Data Source

Un DSO poate fi implementat ca un obiect ActiveX sau ca un Applet Java, și deci pentru includerea unui astfel de obiect într-o pagină HTML se vor folosi tagurile <OBJECT> sau <APPLET>.

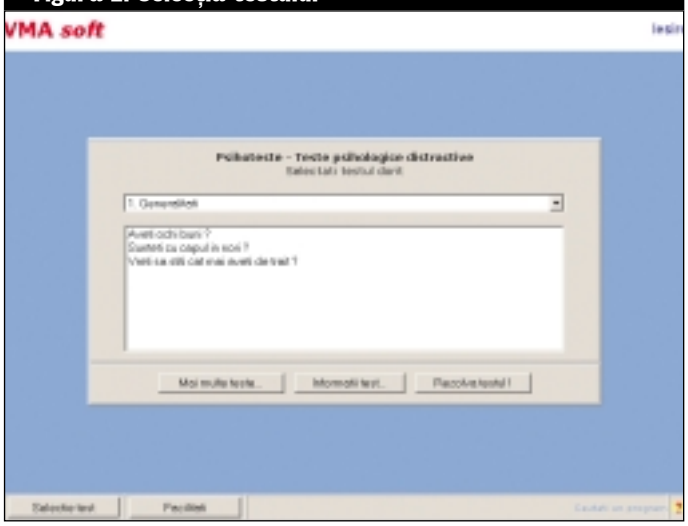
Pentru aplicația noastră vom folosi primul tip de DSO și anume obiectele TDC. Acesta primește datele în format CSV (*Comma Separated Values*) prin protocol HTTP sau dintr-un fișier local. Metoda generală de folosire constă în legarea acestui obiect la un script server ce generează datele în format CSV prin extragerea lor dintr-o bază de date. În cazul aplicației noastre demonstrative am creat direct niște fișiere CSV pe care le-am pus pe serverul de Web ca fișiere statice (vezi sursele ce însoțesc articolul).

O declarație pentru includerea acestui obiect într-o pagină HTML este prezentată în continuare. În general, această declarație poate fi copiată ori de câte ori este necesar și apoi schimbate doar valorile parametrilor.

```
<OBJECT id="tdc_tstdesc" CLASSID="clsid:
   333C7BC4-460F-11D0-BC04-0080C7055A83" VIEWASTEXT>
   <PARAM NAME="UseHeader" VALUE="True">
   <PARAM NAME="FieldDelim" VALUE="|">
   <PARAM NAME="DataURL" VALUE="./teste/tests.files.txt">
</OBJECT>
```

Se poate observa că parametrul DataURL pointează către un fișier TXT. După cum am spus, acesta este în format CSV și are de ex. următorul conținut (pe post de separator este folosit caracterul *pipe*):

Figura 2: Selecția testului



```
filename|name|domainid
tst1|Aveti ochi buni ?|1
tst2|Stiti sa stabiliti relatii inter-umane ?|2
```

Întrucât dorim un mare control asupra datelor pe client nu vom folosi întocmai modelul Data Binding așa cum a fost prezentat în *Figura 1* ci vom accesa direct prin intermediul limbajului de scripting VBScript proprietățile și evenimentele obiectului ActiveX TDC.

Principalul eveniment folosit este `OnDataSetComplete` care apare când toate datele au fost încărcate în TDC. Este necesară folosirea acestui eveniment pentru sincronizarea diferitelor surse de date între ele precum și cu conținutul paginii HTML. Nefolosirea acestuia poate duce la erori ce se depistează greu pe o conexiune rapidă. În cazul unei conexiuni lente pot apare defazări mari în timp între încărcarea diferitelor TDC-uri și dacă utilizatorul acționează în aceste momente asupra aplicației poate produce erori.

Iată în continuare un exemplu de folosire a acestui eveniment în cadrul aplicației noastre. În cadrul screen-ului de selecție a testului am folosit un *combobox* în care sunt trecute categoriile de teste disponibile și un *listbox* ce este umplut automat în momentul unei selecții din combobox.

Toate datele care apar în cele două elemente le legăm pe client prin Data Binding. Ele provin din două TDC-uri, fiecare primind de pe server câte un set de date CSV. Primul TDC conține lista cu categorii de teste iar al doilea lista cu teste și apartenența lor într-un anumit domeniu. Este clar ca dacă cele două seturi de date nu ar fi sincronizate atunci ar apare erori severe la încercarea selecției unei categorii.

Sincronizarea lor o realizăm prin folosirea evenimentului `OnDataSetComplete` într-o modalitate ca cea prezentată în continuare:

```
sub tdc_tstdesc_ondatasetcomplete
  rst_loaded=rst_loaded+1
  if rst_loaded=2 then
    call parent.recordsetst_onloadcompleted()
  end if
end sub

sub tdc_tstdomains_ondatasetcomplete
  rst_loaded=rst_loaded+1
  if rst_loaded=2 then
    call parent.recordsetst_onloadcompleted()
  end if
end sub
```

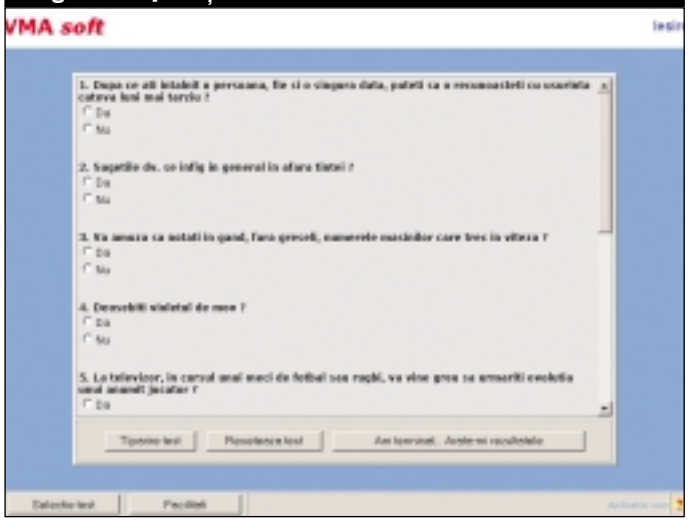
Figura 3: Pagina premergătoare aplicației

Subprocedura `recordsetst_onloadcompleted` este apelată astfel doar în momentul încărcării ambelor TDC-uri. În acest moment avem siguranța că datele sunt prezente pe client în formă completă și putem trece la manipularea lor.

În vederea accesării datelor, obiectul ActiveX TDC expune o proprietate numită `recordset` ce conține chiar un ADO recordset. Acesta se manipulează întocmai ca un *recordset* normal folosit de obicei în paginile ASP pe partea de server. Nu se va insista pe aceste detalii în acest articol, cititorii care doresc informații suplimentare pot face apel la documentația *ADO* (*ActiveX Data Objects*).

Aspect similar programelor Windows. A treia cerință impusă aplicației o constituie cea referitoare la interfața utilizator. Din motive evidente, interfața Windows este una dintre cele mai intuitive și productive GUI-uri existente. Așadar se dorește implementarea unei interfețe similare și aplicației noastre. Cum întreaga aplicație rulează în IE, este necesară ascunderea într-un fel a elementelor specifice browserului. S-a optat pentru rularea în full-screen a întregii aplicații. Mai întâi, în browser sosește o pagină normală HTML pentru ca la apăsarea butonului *Lansează aplicația* să se facă trecerea în full-screen.

Figura 4: Aplicația in full-screen



Deși știm că pe rețeaua noastră nu sunt decât calculatoare Windows cu IE5 vom lua o măsură de prevedere înainte de a intra în aplicație și anume verificarea tipului și a versiunii browserului.

```
<SCRIPT LANGUAGE="JavaScript">
<!-
function EnterSite()
 {
   sAgent = navigator.userAgent;
   bIsMac = sAgent.indexOf(„Mac") > -1;
   bIsIE = sAgent.indexOf(„MSIE") > -1;
   bIsIE4 = sAgent.indexOf(„IE 4") > -1;
   bIsIE5 = sAgent.indexOf(„IE 5")  > -1;
   bIsNav = sAgent.indexOf(„Mozilla") > -1 && !bIsIE;
   bDoesAll = (bIsIE4 || bIsIE5) && !bIsMac;

   if (bIsIE5)
                {
     window.open(„frames_index.html",null,"fullscreen=yes,
   toolbar=no, menubar=no, location=no, status=no");
                }
   else
```

```
    {
    if (bIsNav)
         {
    alert(„Acest site nu poate fi vizualizat cu browserul
    Netscape.\nVa rugam folositi Microsoft Internet Explorer
    5.0\nVa multumim pentru intelegere.“)
    }
    else
    if (bIsIE4)
         {
         alert(„Acest site nu poate fi vizualizat cu
    browserul IE4.\nVa rugam folositi Microsoft Internet
    Explorer 5.0\nVa multumim pentru intelegere.“)
         }
    else
         {
    if (bIsMac)
         {
           alert(„S-a detectat un sistem Mac. Acest site nu
    a fost testat inca pe acest sistem\nVeti fi lasat sa-l
    vizitati dar nu garantam rezultatele“)
      window.open(„frames_index.html“,null,“fullscreen=yes,
    toolbar=no, menubar=no, location=no, status=no“);
         }
    else
                 {
         alert(„Acest site nu poate fi vizualizat decat cu
    browserul Microsoft Internet Explorer 5.0\nVa multumim
    pentru intelegere.“)
         }
    }
    }

 }
//–>
</SCRIPT>
```

Deoarece este posibil ca prima pagină să fie atinsă de browsere cât mai variate vom scrie partea de script în *JavaScript*. Se asigură astfel funcționarea corectă a ei și în navigatorul Netscape. Browserele care nu suportă scripting nu trebuie să ne îngrijoreze deoarece ele vor ignora partea de script și ca urmare nu vor putea lansa aplicația ceea ce ar fi produs erori.

Screen-ul aplicației l-am machetat, prin folosirea frame-urilor HTML, în trei zone: zona de sus cu butonul de ieșire, zona din mijloc unde vor fi prezentate majoritatea informațiilor și zona de jos cu butoanele. Această machetare oferă un aspect plăcut dar totodată asigură și o modalitate de pasare a diferitelor variabile între paginile HTML. Astfel prin intermediul zonei de jos vom transmite conținutul variabilelor între diferitele pagini ce se încarcă în zona din mijloc.

După cum se observă și din figurile 2 și 4 zona din partea de jos are culoarea gri, această culoare potrivindu-se cu culoarea butoanelor ce sunt conținute în această zonă. Apare acum întrebarea: „Dar dacă un utilizator are o altă schemă de culori în Windows diferită de *Windows Standard* nu cumva culoarea butoanelor se va schimba și în acest caz nu se va mai asorta cu culoarea barei ?”

Pentru a rezolva această problemă navigatorul Internet Explorer pune la dispoziție un set de culori ce mapează culorile sistem Windows. Folosind aceste culori aplicația Web își va schimba înfățișarea în funcție de schema de culori aleasă în Windows.

De exemplu stilul pentru bara cu butoane din partea de jos este declarat în felul următor:

```
TABLE.Window
{
     BACKGROUND-COLOR: BUTTONFACE;
     BORDER-BOTTOM: outset thin;
     BORDER-LEFT: outset thin;
     BORDER-RIGHT: outset thin;
     BORDER-TOP: outset thin
}
```

Acum chiar dacă în Windows se alege o altă schemă de culori, butoanele și bara vor rămâne în permanență în aceeași culoare: *ButtonFace*.

Pentru schimbarea dinamică a conținutului pe client a zonei din mijloc, vom folosi în general trei modalități:

- ascunderea și afișarea diferitelor informații cu ajutorul *CSS* (*Cascading Style Sheets*);
- modificarea conținutului diferitelor taguri HTML cu ajutorul proprietății *innerHTML*;
- crearea dinamică a elementelor cu ajutorul metodei *createElement*.

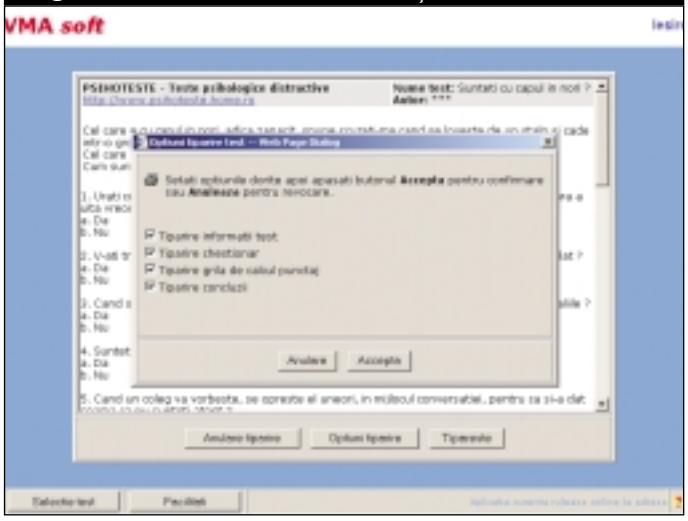
Să vedem cum funcționează fiecare din cele trei modalități în cadrul navigatorului IE. Ne vom folosi pentru acest lucru de mici exemple.

Prima modalitate constă în punerea de la început a tuturor informațiilor pe pagină urmată de comutarea lor la momentul potrivit. În exemplul următor, mesajul „Conținut dinamic” va fi afișat sau ascuns la apăsarea butoanelor *Hide* sau *Show*.

```
<span id=myspan>Continut dinamic</span><br><br>

<input type=button id=hidebtn value=“Hide“>
<input type=button id=showbtn value=“Show“>

<script language=vbscript>
sub hidebtn_onclick
 myspan.style.display=“none“
end sub

sub showbtn_onclick
 myspan.style.display=“„
end sub
</script>
```

În cadrul aplicației, această metodă o vom folosi pentru comutarea între zona cu test și zona cu rezultate. Ambele zone ocupă pe ecran aceeași poziție iar comutarea între ele se face prin ascunderea, respectiv afișarea comutată a lor.

Figura 5: Fereastră modală cu conținut HTML



A doua posibilitate de generare de conținut dinamic pe client constă în folosirea proprietății *innerHTML* a diferitelor atribute HTML. Această proprietate setează sau citește conținutul dintre tagurile elementului specificat.

Un exemplu de folosire al ei este dat în fragmentul următor:

```
<span id=myspan>Net Report</span><br><br>
<input type=button id=textbtn value="Text">
<input type=button id=linkbtn value="Link">

<script language=vbscript>
sub textbtn_onclick
 myspan.innerhtml = „Net Report"
end sub

sub linkbtn_onclick
 myspan.innerhtml = „<a href='http://www.pcreport.ro'>Net
   Report</a>"
end sub
</script>
```

La apăsarea butonului *Text*, în fereastra browserului va apare scris *„Net Report"* sub formă simplă de text, iar la apăsarea butonului *Link* textul se va transforma într-un link către site-ul revistei. Această modalitate am folosit-o în cadrul aplicației la generarea conținutului paginii ce se va tipări la imprimantă.

Așa cum am spus, a treia modalitate de generare de conținut dinamic constă în folosirea metodei *createElement* a obiectului *document*. Iată și un exemplu ce crează un *checkbox* dinamic.

```
<body>
<div id="mydiv"></div>

<script language=vbscript>
set cb=document.createelement(„<INPUT name=intrebarea1>")
cb.type="checkbox"
cb.id="ctrl1"
cb.value="2"
cb.style.cursor="hand"
mydiv.appendchild cb
set cb=Nothing
</script>

</body>
```

Nu-i așa că e simplu? Folosind aceste metode putem crea interfața aplicației așa cum dorim și totul numai prin prelucrări pe client.

Ar mai fi totuși ceva de spus despre facilitățile puse la dispoziție de *Internet Explorer* pentru realizarea interfețelor utilizator în DHTML. După cum se știe, orice aplicație Windows normală are pe lângă fereastra principală și o serie de alte ferestre secundare, unele *modale* altele *nemodale*. În DHTML-ul IE, acestea pot fi create prin apelarea metodelor *ShowModalDialog* și *ShowModalLessDialog* ale obiectului *Window*.

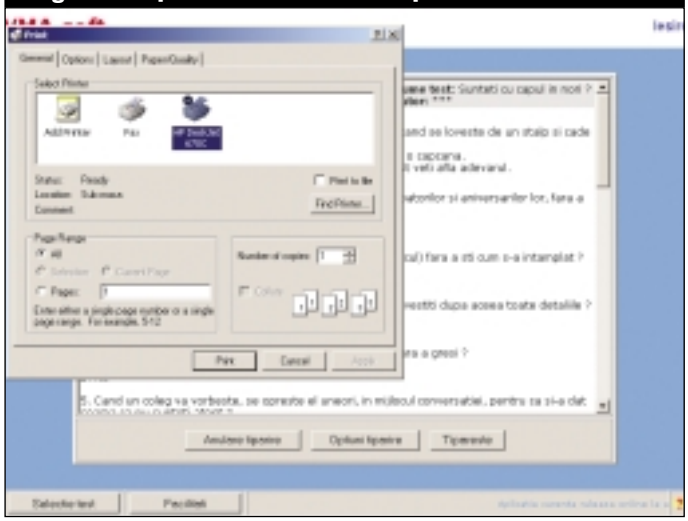
Modul de apelare este următorul:

```
vReturnValue = window.showModalDialog(sURL [, vArguments] [,
   sFeatures])
vReturnValue = window.showModelessDialog(sURL [,vArguments]
   [, sFeatures])
```

unde:

`sURL` = URL-ul paginii HTML care se dorește a se afișa în fereastră

Figura 6: Tipărirea unui test la imprimantă



`vArguments` = argumente care se doresc transmise ferestrei nou create (variant)

`sFeatures` = string ce conține informații despre modul de afișare al ferestrei

`vReturnValue` = variant întors

Cu puțină atenție aceste ferestre pot fi făcute să arate întocmai ca cele întâlnite în aplicațiile Windows. În figura 5, se poate observa o fereastră modală în care se specifică opțiunile de tipărire la imprimantă a testului.

Modul de apel al acestei ferestre este următorul:

```
…
printoptiuni=showmodaldialog(„printtestdlg.html",printoptiu-
   ni, „dialogWidth=500px;dialogHeight=300px; scrollbars=no;
   scroll=no; center=yes; border=thin; help=no; status=no")
…
```

Un lucru destul de important pe care trebuie să-l efectueze o aplicație Windows este tipărirea la imprimantă. Vom înzestra și noi aplicația DHTML cu această facilitate.

Deși pare un lucru complicat, această operație se realizează destul de simplu prin apelul metodei *print* a obiectului *window*. Apelarea metodei *print* are același efect ca alegerea comenzii *Print…* din meniul *File* al browserului:

Totuși înainte de apelul metodei *print* trebuie să se facă următoarele prelucrări:

- testul care se dorește a se tipări trebuie inclus într-un obiect de tip window (în cazul nostru am folosit un `IFRAME`)
- înainte de tipărire trebuie dat focusul acestui obiect.

Având luate aceste măsuri subrutina care realizează tipărirea se scrie destul de simplu în felul următor:

```
sub printdocframe
   window.frames(„docframename").focus
   window.print
end sub
```

Având implementat și mecanismul de tipărire putem spune că am realizat o veritabilă aplicație Windows ce rulează într-un browser Web. Pentru detaliile de implementare ale aplicației vă rog să consultați sursele disponibile la: www.netreport.ro/surse/.

Marian Veteanu este student la Universitatea din Pitești; poate fi contactat la: vmasoft@yahoo.com, http://vmasoft.hypermart.net. ■ 57