AJAX

Jordi Sánchez Cano

PID_00172686

UOC
Universitat Oberta de Catalunya

Índex

Introducció

Una vegada es coneixen les diferents tecnologies utilitzades per AJAX (HTML, XML, HTTP, JavaScript, DOM, etc.) és el moment d'aprendre com es poden utilitzar per a desenvolupar aplicacions web dinàmiques.

En primer lloc, s'estudien les diferents tècniques de comunicació asíncrona amb un servidor HTTP que permetran intercanviar dades i actualitzar parts d'una pàgina sense interferir amb la interacció de l'usuari.

El format predilecte d'intercanvi de dades utilitzat per AJAX és l'XML. En aquest mòdul s'estudiarà com es poden obtenir i manipular documents XML a partir de la seva representació mitjançant el DOM. Una vegada obtingut un document XML, s'aprendrà a transformar-lo a HTML mitjançant XSLT i posteriorment actualitzar parcialment la pàgina actual.

Finalment, s'estudiarà el format JSON, alternatiu a l'XML. Aquest format representa la informació d'una manera més lleugera i accessible des de JavaScript.

Objectius

Amb l'estudi d'aquest mòdul assolireu els objectius següents:

1. Conèixer què és l'HTML dinàmic i crear webs amb continguts actualitzables una vegada s'ha carregat la pàgina.

2. Conèixer les diferents tècniques de comunicació asíncrona amb el servidor emprades per AJAX i quan utilitzar cadascuna d'elles.

3. Estudiar a tall d'introducció què és l'accessibilitat web i com es fa una pàgina accessible.

4. Conèixer l'estructura d'un document representat pel DOM.

5. Aprendre a carregar documents XML i accedir-hi mitjançant el DOM.

6. Seleccionar parts d'un document XML mitjançant XPath.

7. Fer transformacions XSL i mostrar el document obtingut d'una manera dinàmica.

8. Intercanviar i manipular informació amb el servidor mitjançant JSON.

1. Tècniques de comunicació AJAX

En aquest apartat veurem les diferents tècniques de comunicació amb un servidor HTTP[1].

(1)*HTTP* és la sigla d'*hypertext transfer protocol*, 'protocol de transferència d'hipertext'.

1.1. Introducció

El model d'una aplicació web tradicional està basat en la navegació entre pàgines, de manera que cada vegada que es requereix mostrar informació nova, aquesta sol·licita una pàgina nova que substitueix l'anterior.

AJAX permet crear aplicacions web interactives mitjançant l'ús de diferents tècniques i tecnologies. D'una banda, permet intercanviar informació amb el servidor mitjançant l'ús de peticions asíncrones, sense que això influeixi en la interacció de l'usuari i sense la necessitat de navegar cap a una altra pàgina. D'una altra banda, AJAX també preveu l'actualització de parts d'una pàgina amb la informació intercanviada durant la comunicació asíncrona.

Vegeu també

Vegeu la diferència entre el model tradicional i una aplicació web que fa ús d'AJAX en l'apartat 2 del mòdul "Introducció a AJAX" d'aquesta assignatura.

1.1.1. Comunicació asíncrona

Inicialment es va fer ús d'algunes particularitats dels marcs HTML[2] per a l'intercanvi d'informació asíncrona. L'ús d'aquestes tècniques es va popularitzar de tal manera que els navegadors van començar a integrar un nou objecte anomenat XMLHttpRequest implementat especialment per a aquesta tasca.

(2)*HTML* és la sigla d'*hypertext markup language*.

XMLHttpRequest

XMLHttpRequest és una interfície que implementen la majoria de navegadors. Permet l'intercanvi d'informació d'una manera asíncrona amb un servidor HTTP.

Encara que una de les utilitats més importants d'AJAX és l'actualització parcial d'una pàgina, també n'hi ha moltes altres que només preveuen la comunicació asíncrona.

Exemple de comunicació asíncrona

Hi ha clients de correu basats en web (també anomenats *clients de Webmail*) que utilitzen AJAX per a enviar el text d'un nou missatge d'una manera periòdica mentre s'escriu. El servidor emmagatzemarà el missatge per a poder-lo recuperar en cas d'error i així continuar amb la seva edició.

1.1.2. Actualització parcial d'una pàgina

Una vegada carregada una pàgina HTML, és possible modificar-ne el contingut d'una manera dinàmica amb l'ús de DHTML. Es tracta de l'ús d'un conjunt de tècniques combinades amb HTML, JavaScript, CSS i DOM[3]. Mitjançant codi

(3)*DOM* és la sigla de *document object model*.

script, es pot accedir a l'estructura d'un document HTML i modificar-la a partir del DOM, els canvis del qual són mostrats a l'instant al navegador. L'ús d'HTML dinàmic és un dels pilars bàsics d'AJAX, ja que permet mostrar les dades que s'obtenen de manera asíncrona.

DHTML

DHTML és la sigla de *dynamic HTML* (HTML dinàmic). És un terme usat per a definir l'ús conjunt d'HTML, CSS i DOM amb un llenguatge de *script*.

Vegeu també

Els exemples numerats que es presenten al llarg de tot aquest subapartat estan també disponibles en línia.

Exemple 1

L'exemple 1 consta d'una pàgina que una vegada carregada permet modificar-ne el contingut en pressionar un botó. Aquesta acció crea dinàmicament una etiqueta `<h3>` amb el text "Text per mostrar" i l'afegeix al document.

```
<html>
<head>
<title>Exemple DOM 1</title>
<script type="text/javascript">
sContingut = 'Text a mostrar';
function mostrarText(){
     //Es crea un element<h3>
     var elH3 = document.createElement('h3');
     //Es crea un node de text amb el text contingut a sContenido
     var elText = document.createTextNode(sContingut);
     //S'afegeix el node de text al node <h3>
     elH3.appendChild(elText);
     //S'afegeix el node <h3> al cos d'aquesta mateixa pàgina
     document.body.appendChild(elH3);
}
</script>
</head>
<body>
    <button onclick="mostrarText()">
    Feu clic per a mostrar el text
    </button>
</body>
</html>
```

El botó de la pàgina té assignada una funció JavaScript a l'esdeveniment `click`. La primera línia d'aquesta funció crea un node encapçalat `<h3>` amb el mètode `document.CreateElement(nomTag)`. La segona línia crea un node de tipus text a partir de la variable `sContingut`. Posteriorment, afegeix el node de text creat a l'element `<h3>` per a formar el text de la capçalera. Per mostrar-lo, s'afegeix l'element `<h3>` al node `<body>` del document. Això canvia el model del document i força el navegador a refrescar la vista de la interfície d'usuari en què apareixerà la nova capçalera. Aquest mètode és poc pràctic i requereix molt de desenvolupament.

Hi ha una manera més senzilla d'actualitzar una pàgina sense haver de crear HTML dinàmic. Les etiquetes `<div>` i `<span>` són agrupacions lògiques que permeten diferenciar parts del document per a facilitar l'accés i així canviar-ne el contingut i/o estil. Totes dues disposen de la propietat `innerHTML`, que permet establir el text HTML del seu contingut.

Propietat *innerHTML*

En establir text HTML a la propietat `innerHTML`, el model de dades del document s'actualitza automàticament.

Exemple 2

L'exemple 2 mostra el mateix contingut que l'exemple anterior utilitzant l'element `<div>`.

```
<html>
<head>
<title>Exemple DOM amb una divisió</title>
<script type="text/javascript">
sContingut = '<h3>Text a mostrar</h3>';
function mostrarText(){
       var div = document.getElementById('seccioActualitzable');
       div.innerHTML = sContingut;
}
</script>
</head>
<body>
   <button onclick="mostrarText()">
      Feu clic per a mostrar el text
   </button>
   <div id="seccioActualitzable"></div>
</body>
</html>
```

En aquest exemple, la funció `mostrarText()` obté l'objecte que representa la divisió definida en el `<body>`. Posteriorment, s'estableix en la propietat `innerHTML` el codi HTML `<h3>Text a mostrar</h3>`. El navegador mostra els canvis de manera automàtica. En els exemples posteriors s'utilitza el contenidor `<div>` per a mostrar continguts nous.

1.2. Tècnica dels marcs ocults

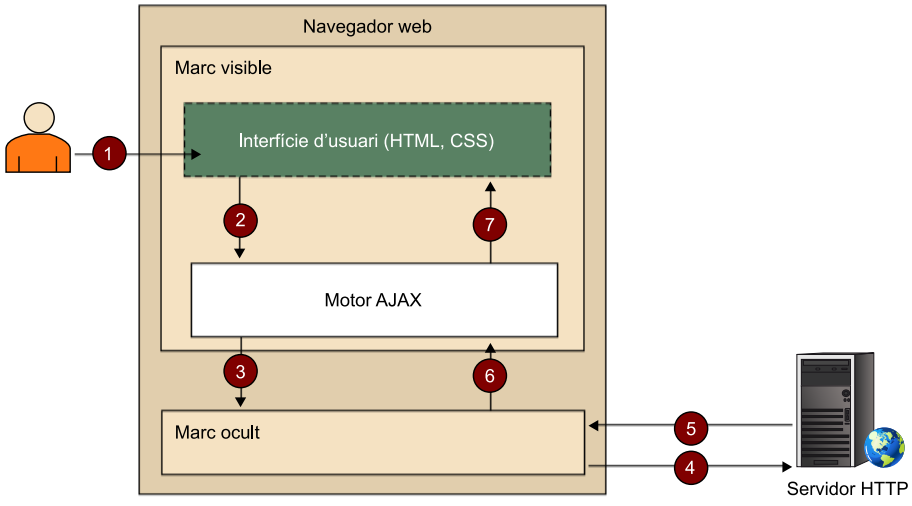
La primera tècnica que va ser utilitzada per a la comunicació asíncrona està basada en l'ús de marcs HTML. Consisteix a definir un conjunt de marcs en una pàgina en què almenys un d'ells és invisible per a l'usuari.

Cada vegada que sigui necessària l'obtenció de noves dades, es canviarà la localització del contingut del marc ocult per a generar una nova petició HTTP. Una vegada rebudes les noves dades en el marc ocult podran ser tractades i mostrades dins de marcs visibles.

Ocultar un marc

Per a ocultar un marc que veu l'usuari, s'ha d'establir l'alçària i amplària a 0 punts. Tot i així, alguns navegadors antics mostren les vores del marc.

Figura 1. Comunicació amb marcs ocults



A continuació, s'expliquen els passos duts a terme en una comunicació per mitjà de marcs ocults per a actualitzar una part del document (aquests passos es poden veure gràficament en la figura 1):

1) L'usuari fa una acció sobre un element HTML. Per exemple, seleccionar un element d'una llista o prémer un botó.

Accions sobre un element HTML

En alguns casos el desencadenant de la petició no és l'usuari. Per exemple, podria ser un procés JavaScript que generi una petició de manera periòdica.

2) L'element HTML notifica l'acció a una o més funcions *script* contingudes en la pàgina, també anomenades *motor AJAX*.

3) El motor AJAX recarrega el contingut del marc ocult cap a una nova URL en què hi ha el contingut que s'ha de descarregar.

4) El marc ocult fa la petició al servidor HTTP.

5) El servidor processa i genera les noves dades, que envia de tornada al navegador. En aquest cas, el destinatari n'és el marc ocult.

6) La pàgina carregada en el marc ocult disposarà d'una funció JavaScript que serà executada i notificarà al motor AJAX la recepció del contingut.

7) El motor AJAX processa la informació rebuda en el marc ocult i duu a terme els canvis en el marc visible.

La **interfície d'usuari** que apareix en la figura 1 fa referència a la part visual d'interacció amb l'aplicació, i es compon d'un conjunt de pàgines HTML i estils CSS.

El **motor AJAX** és el component compost pel conjunt de funcions JavaScript que controlen l'intercanvi d'informació i actualització de la interfície d'usuari. En el cas dels marcs ocults, el motor disposarà de dues funcions bàsiques:

1) Una funció que conté la lògica necessària per a desencadenar les peticions HTTP. Cada acció que requereixi obtenir nova informació cridarà aquesta funció.

2) Una funció per a tractar les dades allotjades dins del marc ocult una vegada obtinguda la resposta HTTP.

Els marcs ocults permeten la generació de peticions aïllades del marc principal en fer-se en un altre d'ocult. Les peticions HTTP generades poden ser de dos tipus, segons el propòsit de l'operació: **GET** i **POST**. A continuació, s'explica com dur-les a terme mitjançant marcs ocults.

1.2.1. Peticions GET

Marcs HTML

Els marcs corresponen a l'etiqueta `<frame>` en HTML.

Per a fer una petició de tipus GET cal especificar en la propietat `location` d'un marc HTML ocult l'URL amb el recurs que es vol obtenir. Per a disposar de marcs ocults, l'aplicació web ha de tenir una pàgina que defineixi els marcs visibles i ocults. En els marcs visibles es mostrarà la interfície d'usuari. Els marcs ocults es reserven per a fer les peticions que es vagin necessitant.

A continuació, s'explica cadascun dels passos d'aquesta tècnica amb dos exemples.

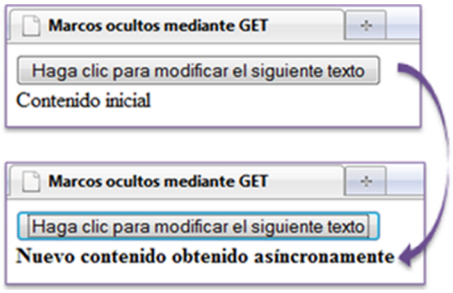
Primer exemple de petició GET

El primer exemple (exemple 3) consisteix en una pàgina amb un text i un botó.

Exemple 3

En fer clic sobre el botó, s'obté un text del servidor i es representa en la pàgina com es mostra en la figura 2.

Figura 2. Exemple simple de marcs ocults



A continuació, es mostra el contingut complet dels diferents fitxers que formen aquest exemple i, posteriorment, s'explica pas a pas com s'ha d'actualitzar el text.

- ***index.html***:

```
<html>
<frameset rows="100%,0" frameborder="no">
    <frame name="visibleFrame" src="ejemplo1.html" noresize="noresize" />
    <frame name="hiddenFrame" src="about:blank" noresize="noresize" />
</frameset>
</html>
```

Aquesta és la pàgina inicial de l'aplicació, en què es defineixen els marcs `visibleFrame` i `hiddenFrame`. La proporció entre els dos marcs és de 100% i 0% sobre l'alçària disponible; d'aquesta manera, el segon marc serà invisible per a l'usuari. L'atribut `noresize` establert en `"noresize"` evita que l'usuari pugui canviar la mida dels marcs.

En el marc visible s'establirà la pàgina següent:

- ***ejemplo1.html***:

```
<html>
<head>
    <script type="text/javascript">
        function requestNewContent(){
            top.frames['hiddenFrame'].location = 'nuevocontenido.html';
        }
        function updateNewContent(sDataText){
            var division = document.getElementById('divUpdatable');
            division.innerHTML = sDataText;
        }
    </script>
    <title>Marcos ocultos mediante GET</title>
</head>
<body>
    <input type="button" onclick="requestNewContent();"
    value="Haga clic para modificar el siguiente texto" />
    <div id="divUpdatable">
        Contenido inicial
    </div>
</body>
</html>
```

Aquesta pàgina serà carregada dins del marc visible i disposa de la interfície d'usuari i del motor AJAX: la interfície és formada per un botó i un text encapsulat en una divisió a la que se li ha establert l'identificador `divUpdatable`. El botó sol·licitarà una pàgina nova al motor AJAX i el contingut nou s'establirà dins d'aquesta divisió.

El motor AJAX és format per les funcions següents:

1) `requestNewContent()`: s'encarrega de sol·licitar el contingut nou. Localitza el marc ocult i modifica la propietat `location` amb l'URL del document que es vol obtenir, cosa que inicia el diàleg amb el servidor.

2) `updateNewContent(sDataText)`: modifica la interfície d'usuari establint el contingut textual passat per paràmetre dins de la divisió `divUpdatable`. Per a això localitza la divisió i estableix el codi HTML en la propietat `innerHTML` de la divisió.

Ús de DOM

Totes dues funcions fan ús del DOM per a accedir als diferents elements de la pàgina i modificar-los.

- ***nuevocontenido.html***

```
<html>
<head>
```

```
<script type="text/javascript">
    window.onload =
    function()
    {
        var division = document.getElementById('divNuevoContenido');
        top.frames['visibleFrame'].updateNewContent(division.innerHTML);
    };
</script>
</head>
<body>
    <div id="divNuevoContenido">
        <b>Nuevo contenido obtenido asíncronament</b>
    </div>
</body>
</html>
```

Aquesta pàgina és l'obtinguda pel marc ocult. Conté el text que es vol mostrar encapsulat en una divisió per a facilitar-ne l'accés. La funció assignada a l'esdeveniment `window.onload` localitza la divisió `divNuevoContenido` i hi passa el codi HTML que conté la funció `updateNewContent(sDataText)` del motor AJAX.

Esdeveniment *onload*

L'esdeveniment `onload` assignat a l'objecte `window` s'executarà quan la pàgina s'hagi carregat per complet en el marc ocult.

Una vegada presentats els fitxers que intervenen en l'exemple, s'explicarà pas a pas com s'actualitza el contingut de la pàgina:

1) Petició del contingut nou:

La petició del contingut nou s'inicia quan l'usuari fa clic sobre el botó del marc visible. Aquest té assignat en l'esdeveniment `onclick` la funció `requestNewContent()`:

```
...
<body>
    <input type="button" onclick="requestNewContent();"
    value="Haga clic para modificar el siguiente texto" />
...
```

Aquesta funció estableix l'URL `'nuevocontenido.html'` en la propietat `location` del marc ocult. Aquest URL és relatiu a l'URL de la pàgina que el conté. En ser en la mateixa localització, n'hi ha prou amb indicar només el nom del document.

```
function requestNewContent(){
   top.frames['hiddenFrame'].location = 'nuevocontenido.html';
}
```

En modificar la propietat `location` del marc, s'envia una petició GET al servidor.

2) Obtenció del contingut nou:

El marc ocult rep la pàgina `nuevocontenido.html`. En carregar-la, crida la funció assignada a l'esdeveniment `window.onload` d'aquest document:

```
window.onload =
function()
{
   var division = document.getElementById("divNuevoContenido");
   top.frames["visibleFrame"].updateNewContent(division.innerHTML);
};
```

La primera línia obté a partir del DOM la divisió en què es troba el contingut que es vol visualitzar. La segona localitza el marc visible, crida a la funció `updateNewContent()`, i hi passa per paràmetre el codi HTML de la divisió localitzada en la línia anterior.

3) Actualització del text:

La funció `updateNewContent()` cridada en el pas anterior executa el codi següent:

```
function updateNewContent(sDataText){
    var division = document.getElementById("divUpdatable");
    division.innerHTML = sDataText;
}
```

La primera línia obté la divisió en què s'ha de visualitzar el contingut nou. La segona línia estableix en la divisió el codi HTML obtingut per paràmetre. Fet això, la pàgina ha estat modificada sense necessitat de recarregar-la per complet.

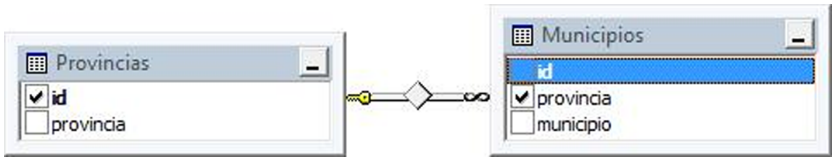
Segon exemple de petició GET

Aquest exemple (exemple 4) consisteix en una pàgina HTML amb dues llistes que permeten a l'usuari escollir una província i una localitat, respectivament. Com que les localitats depenen de les províncies, serà necessari actualitzar la segona llista cada vegada que se selecciona una província de la primera.

Exemple 4

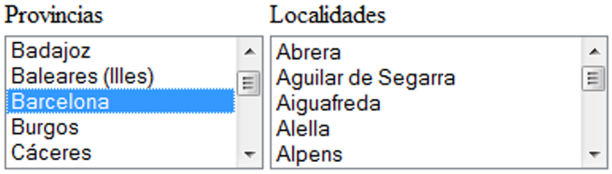
El servidor HTTP disposa d'una base de dades amb dues taules que emmagatzemen el conjunt de províncies i municipis.

Figura 3. Taules de províncies i municipis



AJAX intervé en el moment en què l'usuari selecciona una província de la primera llista, en què s'haurà de fer una petició GET al servidor HTTP afegint a l'URL un paràmetre amb l'identificador de la província seleccionada. El servidor haurà de tornar una pàgina que contingui una llista de les diferents localitats, que posteriorment s'incorporarà en una divisió del marc visible. Si, per exemple, s'ha seleccionat la província "Barcelona", la llista de localitats s'actualitzarà de la manera següent:

Figura 4. Captura de l'exemple



A continuació, es mostra el contingut complet dels diferents fitxers que formen aquest exemple i posteriorment s'explica pas a pas com s'ha d'actualitzar la llista de municipis.

- ***index.html***:

```
<html>
<frameset rows="100%,0" frameborder="no">
     <frame name="visibleFrame" src="ProvincesCities.php" noresize="noresize" />
     <frame name="hiddenFrame" src="about:blank" noresize="noresize" />
</frameset>
</html>
```

S'estableixen dos marcs: un de visible i un altre d'ocult.

- ***provincesCities.php***:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">

/* Funció que obté la llista de ciutats en funció de la província
 * seleccionada. La informació s'allotjarà en el marc ocult */

function requestCities(){
   //S'obté l'índex de la província seleccionada
   var index = document.getElementById('selectProvinces').selectedIndex;
   //S'obté el valor de l'ítem de la província seleccionada
   var id = document.getElementById('selectProvinces').options[index].value;
   //Es redirecciona el marc ocult a la pàgina getCities.php
```

```
   //afegint el paràmetre id a l'URL
   top.frames['hiddenFrame'].location = 'getCities.php?id=' + id;
}

/* Funció a la qual es cridarà des del marc ocult un cop es disposi
 * d'una nova llista de ciutats. Permet establir el codi HTML
 * passat per paràmetre a una divisió en el marc visible. */

function updateNewContent(sDataText){
   var division = document.getElementById('divUpdatable');
   division.innerHTML = sDataText;
}

</script>
<title>Marcos ocultos GET</title>
</head>
<body>
<h3>Selecciona una provincia y una localidad:</h3>
<table>
   <tr><td>Provincias</td><td>Municipios</td></tr>
   <tr><td>
      <select id="selectProvinces" size="15" onclick="requestCities()">
           <?php
           //Es prepara la consulta SQL
           $db_nombre = 'ajax';
           $link = mysql_connect('localhost','ajax','J&J007')
              or die('Could not connect ' . mysql_errno());
           mysql_select_db($db_nombre ,$link) or die("Error selecting database.");
           $sqlQuery = 'Select id, provincia from provincias';
           //Si la consulta ha retornat algun resultat...
           if($result = mysql_query($sqlQuery))
           {
              //Per cada registre retornat s'afegeix un <option> a la llista de provincíes
              while($row = mysql_fetch_assoc($result))
              {?>
                 <option value="<?php echo $row["id"]?>">
                   <?php echo $row["provincia"]?>
                 </option>
              <?php }
           }
           mysql_close($link);
           ?>
      </select>
   </td>
   <td>
       <div id="divUpdatable"></div>
   </td>
```

```
    </tr>
</table>
</body>
</html>
```

Aquesta és la pàgina principal allotjada en el marc visible, amb la que l'usuari interactuarà. Conté les parts següents:

a) El motor AJAX amb dues funcions similars a les de l'exemple anterior:

- `requestCities()`, que inicia una petició HTTP asíncrona que modifica la propietat `location` del marc ocult.
- `updateNewContent(sDataText)`, que mostra el text HTML passat per paràmetre en la divisió `divUpdatable`.

b) Un *script* PHP, que genera la llista HTML de províncies obtingudes de la base de dades.

c) Una divisió amb l'identificador `divUpdatable`, que contindrà la segona llista amb els municipis de la província que se seleccioni a cada moment.

- ***getCities.php***:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">

      /* La funció següent s'executarà en carregar-se aquesta pàgina en el marc
       * ocult. Enviarà part del contingut de la pàgina al motor AJAX
       * perquè el renderitzi en el marc visible. */
      window.onload =
        function()
        {
           var divCities = document.getElementById('divisionCities');
           top.frames['visibleFrame'].updateNewContent(divCities.innerHTML);
        };
</script>
</head>
<body>
<div id="divisionCities">
    <select size="15">
       <?php
           // Es crea la sentència SQL per a obtenir les ciutats a partir de
           // l'identificador de província passat per paràmetre en l'URL
           $db_nombre = 'ajax';
           $idProvincia = $_GET["id"];
```

```
            $link = mysql_connect('localhost','ajax','J&J007')
            or die('Could not connect ' . mysql_errno());
            mysql_select_db($db_nombre ,$link) or die("Error seleccionando la base de datos.");
            $sqlQuery = "Select municipio from municipios Where provincia=\"" .
            $idProvincia."\"";
            //Si hi ha registres...
            if($result = mysql_query($sqlQuery))
            {
               //Per cada ciutat obtinguda, s'afegeix un <option> dins de la llista
               while($row = mysql_fetch_assoc($result))
               {?>
                   <option><?php echo $row["municipio"]?></option>
               <?php }
            }
            mysql_close($link);
       ?>
    </select>
</div>
</body>
</html>
```

Aquesta pàgina serà sol·licitada per a obtenir els municipis d'una província. Executa les accions següents:

a) Recull l'identificador de província passat per paràmetre en l'URL de la petició.

b) Obté de la base de dades la llista de municipis que pertanyen a la província.

c) Genera el codi HTML amb la llista de municipis obtinguts.

A continuació, es mostra pas a pas com s'obtenen i es mostren les localitats.

1) Càrrega inicial de províncies

Es carrega en el marc visible la pàgina `provincesCities.php`, que obté la llista de totes les províncies de la base de dades. Aquest procés es fa en el costat del servidor amb un *script* PHP:

```
<select id="selectProvinces" size="15" onclick="requestCities()">
    ...
    <option value="<?php echo $row["id"]?>"><?php echo $row["provincia"]?></option>
    ...
</select>
```

Una vegada mostrada aquesta pàgina, l'usuari pot seleccionar una província.

2) Selecció d'una província

L'usuari ha seleccionat una província. La propietat `onclic` de `<select>` fa que en seleccionar una província es cridi la funció `requestCities()` per a iniciar la comunicació amb el servidor:

```
<select id="selectProvinces" size="5" onclick="requestCities()">
```

3) Iniciar la comunicació amb el servidor

La funció `requestCities()` modifica la propietat `location` del marc ocult:

```
function requestCities(){
   var index = document.getElementById("selectProvinces").selectedIndex;
   var id = document.getElementById("selectProvinces").options[index].value;
   top.frames["hiddenFrame"].location = "getCities.php?id=" + id;
}
```

Les dues primeres línies obtenen l'identificador de província a partir de l'índex de l'element `<option>` seleccionat de la llista de províncies. L'última estableix, en la propietat `location` del marc ocult, l'URL compost per la pàgina `getCities.php` i el paràmetre `"id"` amb l'identificador de província. En modificar aquesta propietat, automàticament es genera una petició GET en l'URL establert.

4) Generació de pàgina i resposta del servidor

El servidor rep la petició i processa la pàgina `getCities.php`. Inicialment recupera el paràmetre `"id"` de l'URL per a generar la cadena SQL que permetrà obtenir tots els municipis que pertanyen a la província indicada. Per cada municipi obtingut, es crea un `<option>` amb el nom del municipi:

```
...
<div id="divisionCities">
    <select size="15">
        <?php
            $db_nombre = 'ajax';
            $idProvincia = $_GET["id"];
            $sqlQuery = "Select municipio from municipios Where provincia=\""
            . $idProvincia."\"";
            $link = mysql_connect('localhost','ajax','J&J007')
               or die('Could not connect ' . mysql_errno());
            mysql_select_db($db_nombre ,$link)
               or die("Error seleccionando la base de datos.");
            if($result = mysql_query($sqlQuery))
            {
                while($row = mysql_fetch_assoc($result))
```

```
                {?>
                    <option><?php echo $row["municipio"]?></option>
                <?php }
            }
            mysql_close($link);
        ?>
     </select>
</div>
...
```

Tota la llista està continguda dins d'una divisió, que servirà per a manipular-la una vegada estigui continguda dins del marc ocult.

Com en l'exemple anterior, caldrà moure part de la informació del marc ocult cap a la part visible:

```
window.onload =
function()
    {
    var divCities = document.getElementById('divisionCities');
    top.frames['visibleFrame'].updateNewContent(divCities.innerHTML);
};
```

En carregar-se la pàgina al marc ocult, aquesta funció cridarà la funció `updateNewContent()` situada en el marc visible. Com a paràmetre es passarà el contingut de la divisió `divisionCities` amb la llista de municipis.

5) Actualització del contingut

La funció `updateNewContent()` actualitza la divisió `divUpdatabl` amb la nova llista de municipis passada per paràmetre:

```
function updateCities(sDataText){
    var divisionCities = document.getElementById("divUpdatableCities");
    divisionCities.innerHTML = sDataText;
}
```

Una vegada cridada aquesta funció, el navegador mostra la nova llista de ciutats que pertanyen a la província seleccionada.

1.2.2. Peticions POST

En alguns casos cal fer peticions POST en lloc de GET. Un exemple d'això seria la tramesa de múltiples paràmetres o paràmetres de mida considerable. La longitud de l'URL en les peticions està limitada i depèn de cada navegador, per la qual cosa seria inviable l'ús de GET. En canvi, POST permet la tramesa d'una gran quantitat d'informació dins del cos de la petició.

Vegeu també

Vegeu els mètodes HTTP GET i POST en el mòdul "Introducció a AJAX" d'aquesta assignatura.

Els marcs HTML només permeten enviar peticions GET, per la qual cosa aquests només seran d'utilitat per a rebre el contingut. En el seu lloc, s'utilitzen els formularis HTML. Mitjançant formularis HTML és possible especificar el mètode de la petició.

Formularis HTML

Els formularis poden contenir HTML normal i elements especials anomenats *controls*. En enviar un formulari, els controls s'adjunten automàticament a la petició.

Les propietats més importants que s'han de definir en el formulari són les següents:

- `method`: mètode de la petició. S'ha d'establir "post".
- `action`: contindrà l'URL amb el recurs sol·licitat.
- `target`: estableix el marc destinatari de la resposta.

Marc destinatari

Si no s'estableix el marc ocult com a destinació d'una petició, la resposta es rebrà en la mateixa pàgina en què hi ha el formulari, la qual cosa fa que es recarregui la pàgina completa.

La comunicació s'inicia en cridar el mètode `submit()` del formulari. Tots els camps en el seu interior seran enviats d'una manera automàtica dins del cos de la petició. Si el mètode establert en el formulari és GET, els camps del formulari s'enviaran dins de l'URL.

La recepció i manipulació de la resposta funciona de la mateixa manera que les peticions mitjançant el mètode GET: el recurs enviat al client haurà de disposar d'una funció en l'esdeveniment `onload` per a notificar al motor AJAX que les dades sol·licitades estan en el marc ocult.

Exemple de petició POST

Aquest exemple (exemple 5) és una adaptació de l'anterior, excepte que el mètode de les peticions serà mitjançant POST. Cada província seleccionada serà enviada dins del cos de la petició com a paràmetre, i no dins de l'URL com en l'exemple anterior.

Aquest exemple té els mateixos fitxers que l'anterior. Els canvis són en els fitxers `provincesCities.php` i `getCities.php` i aquí s'han marcat amb el comentari *nou*.

- ***provincesCities.php***:

```
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">
/*
 * Fa l'enviament del formulari al servidor mitjançant POST
 */
function requestCities(){          //nou
    document.myForm.submit();      //nou
}                                  //nou

/* Funció a la qual es cridarà des del marc ocult un cop es disposi
 * d'una nova llista de ciutats. Permet establir el codi HTML
 * passat per paràmetre a una divisió en el marc visible.*/
function updateNewContent(sDataText){
    var divisionCities = document.getElementById('divUpdatableCities');
    divisionCities.innerHTML = sDataText;
}

</script>
<title>Marcos ocultos POST</title>
</head>
<body>
<h3>Selecciona una provincia y una localidad:</h3>
<table>
    <tr><td>Provincias</td><td>Municipios</td></tr>
    <tr><td>
        <form name="myForm" method="post" action="getCities.php" target="hiddenFrame">  //nou
            <select name="selectProvinces" size="5" onclick=" requestCities()">
                <?php
                 //S'obté la llista de províncies
                 $db_nombre = 'ajax';
                 $link = mysql_connect('localhost','ajax','
                 J&J007') or die('Could not connect ' . mysql_errno());
                 mysql_select_db($db_nombre ,$link) or die("Error selecting database.");
                 $sqlQuery = 'Select id, provincia from provincias';
                 //Si hi ha resultats...
                 if($result = mysql_query($sqlQuery))
                 {
                  //Cada província obtinguda s'afegeix a la llista HTML
                  while($row = mysql_fetch_assoc($result))
                    {?>
                        <option value="<?php echo $row["id"]?>">
                            <?php echo $row["provincia"]?>
                        </option>
                    <?php }
                 }
                 mysql_close($link);
                 ?>
```

```
            </select>
        </form>                                          //nou
    </td>
    <td><div id="divUpdatableCities"></div></td>
    </tr>
</table>
</body>
</html>
```

El formulari engloba la llista de províncies. Això farà que la província seleccionada s'enviï automàticament dins del cos de la petició en forma de paràmetre. També s'ha establert com a mètode de sol·licitud `"post"`, i com a destinació de la navegació el marc ocult. L'URL s'indica en l'atribut `action`, que sol·licita la pàgina `getCities.php`.

En fer clic sobre una província, es crida la funció `requestCities()`. Ara, aquesta funció es limita simplement a cridar el mètode `submit()` per a iniciar la comunicació:

```
function requestCities(){
    document.myForm.submit();
}
```

La recepció i gestió del contingut nou es fa de la mateixa manera: la funció `updateCities()` obté i mostra la llista nova. Aquesta serà cridada per la pàgina nova una vegada es carregui per complet en el marc ocult.

- ***getCities.php***

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">
    /* La funció següent s'executarà en carregar-se aquesta pàgina en el marc
    * ocult. Enviarà part del contingut de la pàgina al motor AJAX
    * perquè el renderitzi en el marc visible. */
    window.onload =
        function()
        {
            var divCities = document.getElementById('divisionCities');
            parent.frames['visibleFrame'].updateNewContent(divCities.innerHTML);
        };
</script>
</head>
<body>
```

```
<div id="divisionCities">
    <select size="5">
        <?php
           //S'obté la llista de ciutats de la província a partir del seu identificador
           $db_nombre = 'ajax';
           //L'identificador de província s'obté a partir del vector _POST
           $idProvincia = $_POST["selectProvinces"];
           $sqlQuery = "Select municipio from municipios Where provincia=\"" .
           $idProvincia."\"";
           $link = mysql_connect('localhost','ajax','J&J007')
           or die('Could not connect ' . mysql_errno());
           mysql_select_db($db_nombre ,$link) or die("Error seleccionando la base de datos.");
           //Si hi ha hagut resultats...
           if($result = mysql_query($sqlQuery))
           {
               //Per cada ciutat, s'afegeix un <option> a la sortida
               while($row = mysql_fetch_assoc($result))
               {?>
                   <option><?php echo $row["municipio"]?></option>
               <?php }
           }
           mysql_close($link);
        ?>
    </select>
</div>
</body>
</html>
```

Els únics canvis fets sobre aquest fitxer són el tractament de paràmetres. En l'exemple anterior s'obtenien a partir de l'URL de la petició. En aquest cas, són al cos de la petició:

Vector $_POST

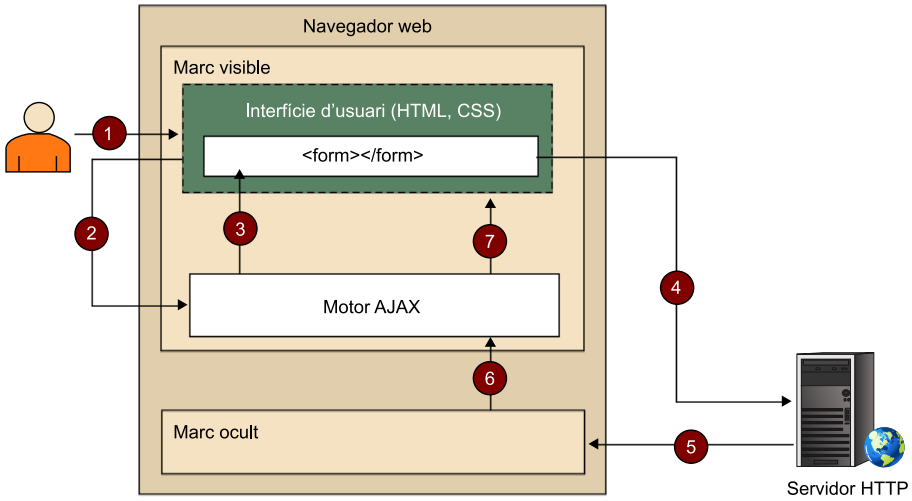
El vector `$_POST` conté totes les variables passades pel mètode HTTP POST.

```
<?php
    ...
    $idProvincia = $_POST["selectProvincias"];
    ...
?>
```

Igual que la tècnica anterior, cal que la pàgina retornada al client contingui una funció JavaScript en l'esdeveniment `window.onload` de la pàgina per a forçar l'actualització de la part visible.

La figura 5 mostra pas a pas cadascuna de les accions dutes a terme en aquest exemple.

Figura 5. Comunicació POST amb un formulari



1) L'usuari fa clic sobre una província.

2) L'esdeveniment associat a la llista crida a la funció `requestCities()` del motor AJAX.

3) El motor AJAX crida el mètode `submit()` del formulari per a iniciar la comunicació.

4) El formulari envia una petició POST al servidor amb l'element seleccionat de la llista.

5) El servidor obté l'identificador de província seleccionat a partir del paràmetre de la petició i genera una pàgina que inclou la llista de municipis.

6) El marc ocult rep la pàgina nova. En renderitzar-la crida a la funció assignada a l'esdeveniment `window.onload`, que crida a la funció `updateCities()` i li passa per paràmetre la llista de municipis.

7) La funció `updateCities()` actualitza la interfície d'usuari amb la llista nova.

1.3. Marcs flotants

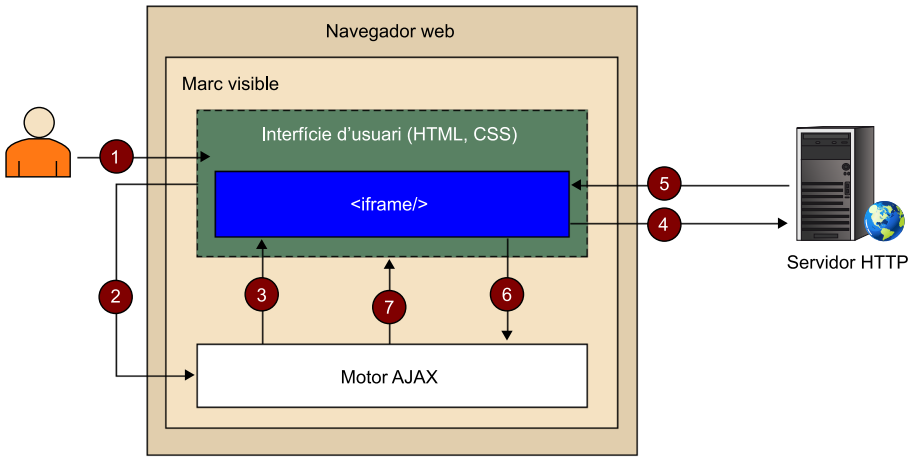
Els marcs flotants són elements HTML que, igual que els marcs tradicionals, poden contenir una pàgina. Aquests són més versàtils i presenten alguns avantatges sobre els marcs estàtics.

Marcs flotants

Els marcs flotants es van introduir en la versió 4.0 d'HTML i corresponen a l'etiqueta `<iframe>`. Es poden situar en qualsevol posició dins de l'àrea de client del navegador i definir la seva posició i dimensió dinàmicament.

La tècnica amb marcs ocults requereix una pàgina dividida en marcs per a disposar d'un d'ocult que permeti la comunicació asíncrona. En canvi, els marcs flotants es poden declarar en la mateixa pàgina que requereixi l'ús d'AJAX. L'arquitectura d'una aplicació AJAX que fa ús de marcs flotants es mostra en la figura 6.

Figura 6. Comunicació amb marcs flotants



Aquest nou model d'aplicació no requereix una divisió prèvia de marcs, sinó que es declaren en el mateix document que requereixi l'ús d'AJAX. Els marcs flotants també es poden declarar d'una manera dinàmica mitjançant el DOM sense requerir una definició prèvia.

Els passos duts a terme des que l'usuari fa una acció fins que s'actualitza el document són:

1) L'usuari interactua amb la interfície fent accions que requereixen dades noves.

2) Es genera un esdeveniment invocant una funció JavaScript del motor AJAX.

3) El motor AJAX modifica la propietat `location` del marc flotant per a iniciar la comunicació.

4) El marc flotant envia una petició GET al servidor.

5) El servidor processa la petició i envia una resposta al marc flotant.

6) Una vegada rebuda la pàgina dins del marc flotant, aquest notifica aquest esdeveniment al motor AJAX i envia el contingut nou.

7) El motor AJAX processa el contingut i actualitza la interfície d'usuari.

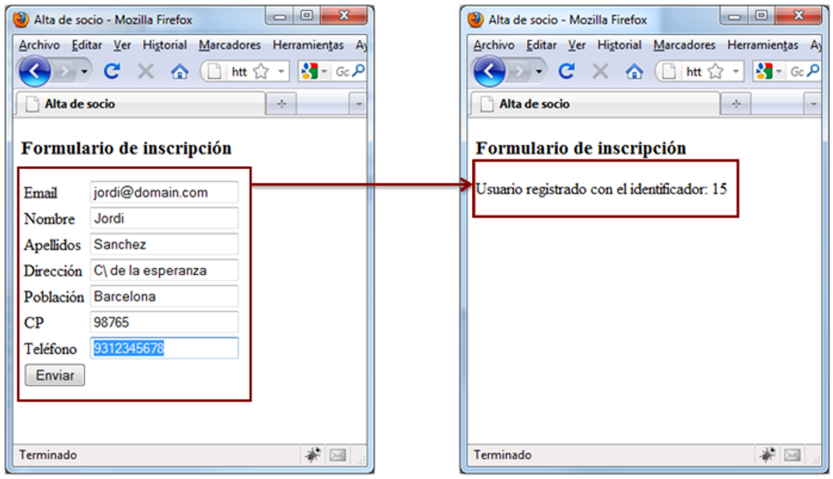
1.3.1. Exemple de marc flotant

Aquest exemple (exemple 6) consisteix en un formulari d'alta d'un usuari amb un conjunt de camps i un botó per a enviar-los. Una vegada que l'usuari l'emplena i fa clic sobre el botó "Enviar", s'envien les dades al servidor. El servidor introdueix les dades en la taula "Members" de la base de dades i torna un missatge amb el resultat de l'operació.

Exemple 6

La figura 7 mostra el resultat d'enviar el contingut del formulari.

Figura 7. Captura de l'exemple 5



El mètode HTTP utilitzat en la petició és POST i permetrà enviar de manera automàtica tots els camps del formulari en forma de paràmetres.

Els diferents fitxers que formen l'exemple són els següents:

- ***index.html***

```
<html>
<head>
<title>Alta de socio</title>
<script type="text/javascript">
  function newUser(){
    document.myForm.submit();
  }

  function showResult(sResult){
    var div = document.getElementById('divForm');
    div.innerHTML = sResult;
  }
</script>
</head>
<body>
<h3>Formulario de inscripción</h3>
<iframe  name="hiddenFrame" width="0" height="0" frameborder="0"></iframe>
```

```
<div id="divForm">
<form name="myForm" method="post" action="saveData.php" target="hiddenFrame">
  <table>
    <tr><td>Email</td><td><input name="email" width="20" /></td></tr>
    <tr><td>Nombre</td><td><input name="name"/></td></tr>
    <tr><td>Apellidos</td><td><input name="surname"/></td></tr>
    <tr><td>Dirección</td><td><input name="address"/></td></tr>
    <tr><td>Población</td><td><input name="city"/></td></tr>
    <tr><td>CP</td><td><input name="postCode"/></td></tr>
    <tr><td>Teléfono</td><td><input name="phone"/></td></tr>
    <tr><td rowspan="3"><button onclick="newUser()">Enviar</button>
    </td></tr>
  </table>
</form>
</div>
</body>
</html>
```

En aquest document hi ha el motor AJAX amb dues funcions JavaScript: `newUser()` per a fer la petició i `showResult(sResult)` per a la visualització del contingut nou. La divisió `divForm` que engloba el formulari serà utilitzada per a mostrar el resultat de l'operació substituint el formulari per la resposta del servidor. També es declara el marc flotant amb el nom `"hiddenFrame"` encarregat de la comunicació asíncrona.

- ***saveData.php***

```
<html>
<head>
<script type="text/javascript">
    window.onload =
       function()
       {
          var div = document.getElementById("divResponse");
          parent.showResult(div.innerHTML);
       };
</script>
</head>
<body>
   <div id="divResponse">
       <?php
           $db_nombre = 'ajax';
           $sEmail = $_POST["email"];
           $sName = $_POST["name"];
           $sSurname = $_POST["surname"];
           $sAddress = $_POST["address"];
           $sCity = $_POST["city"];
```

```
            $sPC = $_POST["postCode"];
            $sPhone = $_POST["phone"];
            $sqlQuery = "insert into members"
            . " (Email,Name,Surnames,Address,City,PC,Phone) Values ('$sEmail',"
            . " '$sName','$sSurname','$sAddress','$sCity','$sPC','$sPhone')";
            $link = mysql_connect('localhost','ajax','J&J007')
            or die('Could not connect ' . mysql_errno());
            mysql_select_db($db_nombre ,$link)
            or die("Error seleccionando la base de datos.");
            if($oResult = mysql_query($sqlQuery))
            {
                $sResult = "Usuario registrado con el identificador: ". mysql_insert_id();
            }
            else
            {
                $sResult = "No se ha podido registrar el usuario";
            }
            echo $sResult;
            mysql_close($link);
        ?>
    </div>
</body>
</html>
```

Aquest document conté l'*script* PHP per a fer l'alta de l'usuari en la base de dades. Tots els camps que es registren s'obtenen dels paràmetres del cos de la petició accessible des del vector `$_POST`. També disposa d'una funció JavaScript assignada a l'esdeveniment `window.onload` per a notificar al motor AJAX la recepció d'aquest contingut en el marc flotant ocult.

Els canvis més significatius amb relació a la tècnica dels marc ocults són dos: el primer és la definició del marc ocult. El marc ocult flotant es pot declarar en el mateix document en què es troben la interfície i el motor AJAX. El segon canvi és l'accés al document que conté la funció d'actualització del contingut des del marc ocult. En un marc tradicional ocult s'havia d'accedir al nivell superior de l'objecte del document i a partir d'allà buscar el marc visible de la manera següent:

```
parent.frames["visibleFrame"].javaScriptFunction();
```

En aquest cas, el marc flotant es troba en el mateix document en què hi ha la funció `showResult()`. N'hi ha prou d'accedir al document amb l'objecte `parent` i accedir aleshores a la funció:

```
parent.showResult(div.innerHTML);
```

1.4. XMLHttpRequest

XMLHttpRequest

XMLHttpRequest va ser ideat inicialment per Microsoft i es va incloure en el seu navegador, l'Internet Explorer 5.

Encara que el terme AJAX no va ser introduït fins a l'any 2005, ja es feia ús de les tècniques de marcs ocults per a obtenir informació del servidor d'una manera asíncrona. Paral·lelament, els navegadors van començar a incorporar un objecte anomenat XMLHttpRequest que permetia fer les mateixes tasques d'una manera més eficient.

XMLHttpRequest és una interfície accessible des de JavaScript que està implementada per la majoria dels navegadors, que permet fer peticions HTTP. Disposa d'un conjunt de mecanismes per a la tramesa i recepció de peticions síncrones i asíncrones. XMLHttpRequest va ser ideat per a intercanviar informació en format XML[4], però actualment suporta diferents formats com ara text ASCII, HTML i JSON[5].

(4)*XML* és la sigla d'*extensible markup language*.

(5)*JSON* és la sigla de *JavaScript object notation*.

Navegadors i XML

La majoria dels navegadors actuals ofereix una implementació nadiua d'aquest objecte, com per exemple Mozilla Firefox, Opera, Google Chrome, Konkeror, Safari i Microsoft Internet Explorer a partir de la versió 7.

La interfície XMLHttpRequest està sent regulada pel W3C amb especificacions definides en dos nivells:

1) XMLHttpRequest Level 1: el primer esborrany va ser presentat l'abril de 2006 amb l'especificació estàndard.

2) XMLHttpRequest Level 2: la primera versió de l'esborrany d'especificació va ser presentada el febrer de 2008 i inclou noves funcions: esdeveniments de progrés de peticions, peticions entre diferents dominis i maneig de flux de bytes per a la transmissió i recepció de dades.

Entorns de treball

Hi ha entorns de treball (*frameworks*), usualment fitxers JavaScript, que s'executen en el costat del client i que faciliten el desenvolupament web ampliant el DOM, oferint funcionalitats AJAX, etc.

En no haver-hi una definició definitiva, els desenvolupadors han de tenir en compte les diferents implementacions d'aquesta interfície a cada navegador. Aquestes diferències poden ser abstretes utilitzant entorns de treball[5] que ofereixen una capa d'abstracció de les diferents implementacions.

1.4.1. Propietats

Propietats d'XMLHttpRequest

La propietat *responseBody* va ser afegida en l'especificació XMLHttpRequest Level 2; la resta es va definir en l'especificació de nivell 1.

La taula següent mostra les propietats d'XMLHttpRequest:

Propietats	Descripció
readyState	Indica l'estat de l'objecte.
responseText	Retorna la resposta del servidor en una cadena de text.
responseXML	Retorna la resposta obtinguda en un objecte DOM XML.
responseBody	Retorna la resposta del servidor en un vector de bytes.
status	Proporciona el codi d'estat de la resposta.
statusText	Retorna l'estat de la resposta amb una explicació textual.

Una vegada enumerades les propietats d'XMLHttpRequest, explicarem en detall cadascuna d'elles.

- ***readyState***

Un objecte XMLHttpRequest passa per diferents estats durant la creació i l'intercanvi de dades amb el servidor HTTP. Aquesta propietat proporciona l'estat de l'objecte en cada moment. Els possibles estats són els següents:

Valor	Estat	Descripció
0	UNSENT	Indica que l'objecte XMLHttpRequest ha estat creat però no ha estat inicialitzat.
1	OPENED	Indica que l'objecte està inicialitzat i preparat per a enviar una petició.
2	HEADERS_RECEIVED	Totes les capçaleres de la resposta ja han estat rebudes i alguns membres de l'objecte ja estan disponibles.
3	LOADING	S'està rebent el cos de la resposta.
4	DONE	La transacció s'ha completat.

Les funcions JavaScript que controlen la transmissió i recepció de dades poden comprovar aquest valor per a conèixer l'estat d'una petició.

- ***responseText***

Retorna el contingut de la resposta HTTP en format de text sempre que el valor de la propietat `readyState` sigui 4 `(DONE)` o 3 `(LOADING)`. En el primer cas s'obtindrà tota la resposta i en el segon, el contingut rebut fins al moment.

- ***responseXML***

Proporciona un objecte DOM del document XML obtingut. Al contrari que la propietat `responseText`, només es podrà obtenir el resultat fins a rebre la resposta completa: quan la propietat `readyState` valgui 4 `(DONE)`.

- ***responseBody***

Proporciona la resposta en una matriu[6] de bytes sense signe. Pot ser útil per a obtenir dades binàries en una resposta.

(6)En anglès, *array*.

- ***status***

Obté el codi d'estat de la resposta HTTP. Si l'operació és correcta, retornarà el codi 200; si la pàgina no existeix, retornarà 404, etc.

Vegeu també

Vegeu els codis de respostes HTTP més comuns en el mòdul "Introducció a AJAX" d'aquesta assignatura.

- ***statusText***

Retorna el codi de resposta en format textual. Per exemple, si el codi de resposta és 200, el text serà "OK".

1.4.2. Mètodes

Els diferents mètodes de l'objecte XMLHttpRequest es mostren en la taula següent:

Mètode	Descripció
open	Prepara l'objecte per a una petició HTTP.
send	Envia la petició al servidor.
abort	Cancel·la la petició en curs.
setRequestHeader	Permet afegir un encapçalament a la petició HTTP.
getAllResponseHeaders	Obté tots els encapçalaments HTTP de l'última petició en una cadena de caràcters.
getResponseHeader	Obté la capçalera HTTP que s'especifica per paràmetre.

A continuació, s'explica cadascun dels mètodes enumerats.

- ***open(mètode, url, async, usuari, contrasenya)***

Inicialitza l'objecte XMLHttpRequest i el prepara per a enviar una petició HTTP. Els paràmetres són:

- ***mètode***: Requerit. Cadena (*string*) que indica el mètode de la petició HTTP (GET, POST, etc.).
- ***url***: Requerit. Cadena que indica l'URL al qual es dirigeix la petició.
- ***async***: Opcional. Booleà que indica si la petició és asíncrona o no. Per defecte, la petició es fa d'una manera asíncrona (*true*).
- ***usuari*** **i** ***contrasenya***: Opcionals. Cadenes que indiquen l'usuari i la contrasenya en cas que el servidor requereixi autenticació.

Si la crida té èxit, l'estat de l'objecte passa ser 1 `(OPENED)`.

- ***send([body])***

Envia la petició prèviament preparada pel mètode `open()`. L'estat de l'objecte abans de cridar a la funció `send()` ha de ser 1 `(OPENED)`. En el cas contrari, llançarà una excepció.

Si en el mètode `open()` es va establir el paràmetre `async` com a vertader, aquesta crida retornarà el control tot seguit. En cas contrari, el procés es bloquejarà fins a obtenir una resposta.

El paràmetre `body` permet afegir informació al cos de la petició. Si la petició no requereix cap cos, s'especificarà `null` o s'ometrà el paràmetre.

- ***abort()***

Cancel·la una petició en curs i estableix l'estat de l'objecte a 0 `(UNSENT)`.

- ***setRequestHeader(header, value)***

Qualsevol petició HTTP incorpora un conjunt de línies opcionals que aporten informació addicional al destinatari, ja sigui el client o servidor. Aquestes línies es denominen capçaleres[7] i doten el protocol HTTP d'una gran flexibilitat per a intercanviar dades sobre la transacció: informació sobre el tipus de contingut, hora i data de la petició, informació sobre el navegador del client, etc. Els encapçalaments són compostos per un nom, seguit de dos punts (:) i el valor.

[7] En anglès, *headers*.

Vegeu també

Per a més informació sobre el protocol HTTP, vegeu el subapartat 2.1 del mòdul "Introducció a AJAX" d'aquesta assignatura.

La línia següent defineix una capçalera de resposta d'un servidor que indica el tipus de contingut retornat al client:

```
Content-Type: text/html; charset=ISO-8859-1
```

Aquesta capçalera indica el tipus de contingut, que en aquest cas es tracta d'una pàgina HTML i la seva corresponent codificació de caràcters.

El mètode `setRequestHeader(header, value)` permet establir en un objecte XMLHttpRequest un encapçalament i el seu valor. Aquest encapçalament s'incorporarà en la petició següent en cridar a `send()`. Per a afegir capçaleres, l'estat de l'objecte ha de ser 1 `(OPENED)`.

- ***getAllResponseHeaders()***

Obté tot els encapçalaments de la resposta HTTP en una cadena. Si l'objecte no està en estat 3 `(LOADING)` o 4 `(DONE)`, retorna `null`.

- ***getResponseHeader(header)***

Retorna una cadena amb el contingut de l'encapçalament passat per paràmetre. Si l'objecte no es troba en estat 3 `(LOADING)` o 4 `(DONE)`, retorna `null`.

1.4.3. Esdeveniments

La taula següent mostra els esdeveniments que permeten notificar els canvis d'estat i els esdeveniments que ocorren en una transacció HTTP:

Esdeveniment	Esdeveniment llançat quan...
onreadystatechange	... la propietat *readyState* canvia de valor.
onloadstart	... la petició s'inicia.
onprogress	... mentre es carreguen i s'envien dades en el transcurs d'un diàleg.
onabort	... la petició es cancel·la amormalment.
onerror	... la petició ha fallat.
onload	... la petició s'ha completat satisfactòriament.

Esdeveniments

Tots els esdeveniments van ser introduïts en l'especificació **XMLHttpRequest Level 2**, llevat `onreadystatechange` que es va definir en la primera especificació.

Esdeveniment `onreadystatechange`

L'esdeveniment `onreadystatechange` permet capturar tots els canvis d'estat de l'objecte XMLHttpRequest. La resta d'esdeveniments té un caràcter més específic.

Abans de fer la transmissió d'una petició, s'hauran d'assignar les diferents funcions JavaScript encarregades de gestionar els successos als esdeveniments que es vulgui capturar.

1.4.4. Ús de l'objecte XMLHttpRequest

Per a fer una petició asíncrona amb el servidor utilitzant la classe XMLHttpRequest s'han de dur a terme els passos següents:

1) Creació d'una instància d'XMLHttpRequest.

2) Inicialització.

3) Assignació d'una funció per a gestionar els canvis d'estat de la instància d'XMLHttpRequest.

4) Configuració de l'encapçalament de la petició.

5) Enviament de la petició.

6) Recepció i gestió del contingut nou.

A continuació, es descriu en detall cadascun d'aquests passos.

Creació d'una instància d'XMLHttpRequest

El primer pas és crear una instància d'XMLHttpRequest. Segons el navegador, la creació pot variar: Internet Explorer proporciona l'objecte XMLHttpRequest com un control ActiveX fins a la versió 6 del navegador. En canvi, Firefox, Chrome i Safary ofereixen suport nadiu. Microsoft® Internet Explorer ofereix de manera nadiua l'objecte XMLHttpRequest a partir de la versió 7.

Exemple 7

La funció següent (exemple 7) crea i torna una instància d'XMLHttpRequest vàlida per a múltiples navegadors:

```
function createXMLHttpRequestObject() {
    /*Aquesta condició determina si el navegador té suport
    nadiu per a XMLHttpRequest*/
    if(window.XMLHttpRequest){
        return new XMLHttpRequest();
    }
    /*No hi ha suport nadiu, s'intenta obtenir com un objete ActiveX
    per a versions anteriors d'Internet Explorer 7*/
    else if(window.ActiveXObject){
        try{
            return new ActiveXObject('Msxml2.XMLHTTP');
        }
        catch(e){
            try{
                return new ActiveXObject('Microsoft.XMLHTTP');
            }
             catch(E){}
        }
    }
    alert('No s'ha pogut crear una instància d'XMLHttpRequest');
}
```

La primera condició avalua si existeix una implementació nadiua d'XMLHttpRequest; si és així, crea una instància. Si no existeix una implementació nadiua, es comprova que existeixi la classe `ActiveXObject` per a crear una instància d'XMLHttpRequest per a versions de l'Internet Explorer inferiors a la 7.

En els exemples següents, totes les instàncies d'XMLHttpRequest partiran d'aquesta funció.

Inicialització

Una vegada creada una instància d'XMLHttpRequest, cal inicialitzar-la. Per a això, es cridarà el mètode `open` i s'indicarà el mètode HTTP que s'emprarà (GET o POST) i l'URL que indica el recurs que s'ha d'obtenir:

```
var url = "urlToResource";
oXMLHttpObject = createXMLHttpRequestObject();
oXMLHttpRequest.open('GET', url);
```

Opcionalment es pot indicar un tercer paràmetre booleà que indiqui si es vol una comunicació asíncrona o síncrona. Els paràmetres quart i cinquè, també opcionals, permeten establir el nom d'usuari i la contrasenya en cas que el servidor HTTP requereixi autenticació.

Tercer paràmetre

Si s'estableix *false* en el tercer paràmetre, el mètode `send` bloqueja l'execució fins a finalitzar la transacció.

Si l'operació s'ha fet de manera correcta, l'objecte està inicialitzat i passa a ser en l'estat 1 `(OPENED)`.

Assignació d'una funció per a gestionar els canvis d'estat

Per a gestionar la transacció HTTP s'ha d'assignar una funció JavaScript als diferents esdeveniments. La propietat `onreadystatechange` permet capturar tots els canvis d'estat.

La funció següent permet gestionar una resposta una vegada que s'ha completat amb èxit:

```
oXMLHttpObject.onreadystatechange = function(){
    if(oXMLHttpObject.readyState == 4){
        if(oXMLHttpObject.status == 200){
            //gestió de la resposta
        }
        else{
            //error
        }
    }
}
```

Per a saber si les dades han estat rebudes, la funció comprova el valor de la propietat `readyState`. Si el seu valor és 4 `(DONE)` implicarà que la transacció ha finalitzat. Per a saber si ha tingut èxit, la funció consulta en una segona condició si el codi de l'estat de la resposta és 200 `(OK)`. En aquest cas, es farà la gestió del contingut.

També es poden assignar funcions a esdeveniments per a controlar estats i esdeveniments més concrets:

```
oXMLHttpObject.onload = function(){
    if(oXMLHttpObject.status == 200){
        //gestió de la resposta
    }
};

oXMLHttpObject.onerror = function(){
    alert("XMLHttpRequest: Error en la petició HTTP.");
};
```

L'exemple anterior assigna una funció a l'esdeveniment `onload` i `onerror`. La primera serà invocada si la resposta es rep correctament. La segona, en el cas contrari.

Configuració de l'encapçalament de la petició

Abans de la transmissió de la petició s'han d'establir els encapçalaments addicionals si fos necessari. Per exemple, les línies següents modifiquen l'encapçalament `User-Agent` que proporciona al servidor informació sobre el client:

```
oXMLHttp.setRequestHeader('User-Agent', 'AJAX');
```

Tramesa de la petició

L'objecte XMLHttpRequest està preparat per a fer la comunicació, i per a iniciar-la, es crida al mètode `send`:

```
oXMLHttpObject.send(null);
```

Recepció i gestió del contingut nou

Els programes controladors d'esdeveniments gestionaran la resposta una vegada enviada la petició. En cas que la recepció s'hagi completat amb èxit, es pot accedir a les propietats `responseText` o `responseXML` per a accedir a les dades rebudes.

Programes controladors d'esdeveniments

Els programes controladors d'esdeveniments són les funcions JavaScript assignades als diferents esdeveniments. Aquests es van establir en el tercer pas.

Exemple mitjançant peticions GET

L'exemple 8 permet consultar els usuaris registrats amb l'exemple anterior de marcs flotants.

Consta d'una pàgina que permet consultar la informació d'un soci a partir del seu número d'abonat. En introduir el número d'abonat en un quadre de text i fer clic en un botó, se sol·licitarà la informació de l'abonat mitjançant una petició asíncrona i s'actualitzarà la pàgina mostrant una taula amb les dades. Els arxius d'aquest exemple són els següents:

- ***index.html***

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Search member form</title>
<script type="text/javascript" src="../script/Ajax.js"></script>
<script type="text/javascript">

    function searchMember(){
        //S'obté l'objete XMLHttpRequest
        var oXMLHttpRequest = createXMLHttpRequestObject();
        //S'obté l'identificador de l'usuari
        var id = document.getElementById("memberNumber").value;
```

```
        var div = document.getElementById("divRequest");
        //Es construeix l'URL amb un paràmetre indicant l'identificador
        var url = "getMember.php?id=" + id;
        //S'estableix el mètode GET i l'URL anterior
        oXMLHttpRequest.open('GET', url);
        //S'assigna la funció que gestionarà els canvis d'estat
        oXMLHttpRequest.onreadystatechange = function(){
            if(oXMLHttpRequest.readyState == 4){
                if(oXMLHttpRequest.status == 200){
                    div.innerHTML = oXMLHttpRequest.responseText;
                }
                else{
                    alert("There was a problem with the request.");
                }
            }
        }
        //S'envia la petició
        oXMLHttpRequest.send(null);
    }
</script>
</head>
<body>
        N° Socio <input id="memberNumber" width="20" />
        <button onclick="searchMember()">Search</button><br><br>
        <div id="divRequest"></div>
</body>
</html>
```

Aquesta és la pàgina principal de l'exemple amb la interfície d'usuari i el motor AJAX. La interfície consta d'un quadre de text i un botó per a consultar el número d'un soci.

El motor AJAX és compost per la funció `searchMember()` que crea i prepara una instància d'XMLHttpRequest.

- ***getMember.php***

```
<?php
    header('Content-Type: text/html; charset=ISO-8859-1');
    $db_nombre = 'ajax';
    //S'obté l'identificador de l'usuari del paràmetre en l'URL
    $idMember = $_GET["id"];
    $sqlQuery = "Select * from members Where nMember=\"" .
    $idMember."\"";
    $link = mysql_connect('localhost','ajax','J&J007')
    or die('Could not connect ' . mysql_errno());
    mysql_select_db($db_nombre ,$link) or die("Error selecting db.");
```

```
    //Si hi ha resultats en la consulta SQL...
    if($result = mysql_query($sqlQuery))
    {
        $total_rows = mysql_num_rows($result);
        //es torna la informació de l'usuari en una taula
        if($total_rows >= 1)
        {
            $row = mysql_fetch_assoc($result);?>
            <table border="1">
                <tr><td>Email:</td>
                <td><?php echo $row["Email"]?></td></tr>
                <tr><td>Name:</td>
                <td><?php echo $row["Name"]?></td></tr>
                <tr><td>Surname:</td>
                <td><?php echo $row["Surnames"]?></td></tr>
                <tr><td>Address:</td>
                <td><?php echo $row["Address"]?></td></tr>
                <tr><td>City:</td>
                <td><?php echo $row["City"]?></td></tr>
                <tr><td>CP:</td>
                <td><?php echo $row["PC"]?></td></tr>
                <tr><td>Phone:</td>
                <td><?php echo $row["Phone"]?></td></tr>
            </table><?php
        }
        else
        {
            echo "Member not found";

        }
    }
    mysql_close($link);
?>
```

Aquesta pàgina consulta el soci de la base de dades a partir de l'identificador passat per l'URL. Una vegada obtingut, construeix una taula que serà retornada al client.

A continuació, es detalla pas a pas el procés d'obtenir informació d'un soci.

1) Càrrega de la pàgina inicial *index.html*

La pàgina mostra el quadre de text i el botó on el client podrà indicar el número de soci i fer clic per enviar la consulta. L'atribut `onclick` s'estableix a la funció `searchMember()`, que iniciarà la petició. Per a representar les dades obtingudes, es declara una divisió amb l'identificador `divRequest`:

```
<body>
    N° Socio
    <input id="memberNumber" width="20" />
    <button onclick="searchMember()">Search</button><br><br>
    <div id="divRequest"></div>
</body>
```

2) Transmissió de la petició GET

L'usuari ha introduït un número de soci i ha fet clic, la qual cosa ha comportat que es cridi la funció `searchMember()`. Aquesta funció crea l'objecte XMLHttpRequest i prepara l'URL format per la pàgina `getMember.php` i el paràmetre `id` que s'obté del quadre de text. A continuació, crida el mètode `open()` i indica que la comunicació es farà amb el mètode GET i assigna l'URL. Abans d'enviar la petició, s'estableix a la propietat `onreadystatechange` una funció JavaScript per a gestionar els canvis d'estat de l'objecte.

Finalment, s'inicia la comunicació cridant el mètode `send()`:

```
var id = document.getElementById("memberNumber").value;
var div = document.getElementById("divRequest");
var url = "getMember.php?id=" + id;
oXMLHttpRequest.open('GET', url);
oXMLHttpRequest.onreadystatechange = function(){
        ...
}
oXMLHttpRequest.send(null);
```

3) Generació de pàgina i resposta del servidor

El servidor rep la petició i construeix la pàgina `getMember.php`. Inicialment s'estableix el tipus de contingut del document que cal enviar, que serà de tipus text (`text/html`) mitjançant la funció php `header()`. A continuació, s'obté l'identificador del soci de l'URL i es fa la consulta a la base de dades. Si el soci existeix, es retorna una taula amb la seva informació; en el cas contrari, es retorna un missatge d'error.

4) Recepció i actualització de pàgina

Cada vegada que l'objecte XMLHttpRequest canviï d'estat, s'executarà la funció assignada a la propietat `onreadystatechange`. Quan s'obtingui la resposta del servidor, aquesta funció actualitzarà la divisió sempre que l'objecte es trobi en l'estat 4 (`loaded`) i l'estat de la resposta sigui 200 (`OK`). En aquest cas, s'assignarà a la divisió el contingut de la resposta obtingut a partir de la propietat `responseText` i mostrarà així el resultat de la consulta:

```
oXMLHttpRequest.onreadystatechange = function(){
```

```
    if(oXMLHttpRequest.readyState == 4){
       if(oXMLHttpRequest.status == 200){
          div.innerHTML = oXMLHttpRequest.responseText;
       }
       else{
          alert("There was a problem with the request.");
       }
    }
 }
```

Exemple mitjançant el mètode POST

Aquest exemple (exemple 9) fa una alta d'usuari igual que en l'exemple anterior de marcs ocults flotants. L'intercanvi d'informació s'efectua amb POST usant XMLHttpRequest.

Les peticions POST que contenen camps dins d'un formulari s'han de tractar d'una manera especial. En cridar el mètode `submit` del formulari, aquest fa la petició d'una manera automàtica. En aquest cas, és l'objecte XMLHttpRequest que ha de fer la petició. Per a això, s'haurà d'enviar la petició afegint els camps del formulari dins del cos d'aquesta.

Bàsicament, l'exemple consisteix en una pàgina HTML que mostrarà un formulari per a donar d'alta un soci nou. El formulari és compost per diferents quadres de text i un botó per a iniciar la tramesa d'informació.

Els fitxers són els següents:

- ***index.html***

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Add member form</title>
<script type="text/javascript" src="../script/AJAX.js"></script>
<script type="text/javascript">

function newUser(){
    var oXMLHttpObject = createXMLHttpRequestObject();
    var form = document.forms[0];
    var div = document.getElementById("divRequest");
    oXMLHttpObject.open('post', form.action);
    oXMLHttpObject.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    oXMLHttpObject.onreadystatechange = function(){
        if(oXMLHttpObject.readyState == 4){
            if(oXMLHttpObject.status == 200){
                div.innerHTML = oXMLHttpObject.responseText;
```

```
            }
            else{
                alert("There was a problem with the request.");
            }
        }
    }
    oXMLHttpObject.send(getFormParams(form));
}
function getFormParams(form){
    var params = new Array();
    for(var i = 0; i < form.elements.length; i++){
        var param = encodeURIComponent(form.elements[i].name);
        param += "=";
        param += encodeURIComponent(form.elements[i].value);
        params.push(param);
    }
    return params.join("&");
}
</script>
</head>
<body>
    <form method="post" action="saveData.php" onsubmit="newUser(); return false">
    <table>
        <tr><td>Email</td><td><input name="email" width="20" /></td></tr>
        <tr><td>Nombre</td><td><input name="name"/>
        </td></tr>
        <tr><td>Apellidos</td><td><input name="surname"/></td></tr>
        <tr><td>Dirección</td><td><input name="address"/></td></tr>
        <tr><td>Población</td><td><input name="city"/></td></tr>
        <tr><td>CP</td><td><input name="postCode"/></td></tr>
        <tr><td>Teléfono</td><td><input name="phone"/></td></tr>
    </table>
    <input type="submit" value="Save member" /><br>
    </form>

    <div id="divRequest"></div>
</body>
</html>
```

Pàgina principal amb un formulari i un conjunt de camps per emplenar.

- ***saveData.php***

```
<?php
    header('Content-Type: text/html; charset=ISO-8859-1');
    $db_nombre = 'ajax';
    $sEmail = $_POST["email"];
```

```
    $sName = $_POST["name"];
    $sSurname = $_POST["surname"];
    $sAddress = $_POST["address"];
    $sCity = $_POST["city"];
    $sPC = $_POST["postCode"];
    $sPhone = $_POST["phone"];
    $sqlQuery = "Insert into members"
    . " (Email,Name,Surnames,Address,City,PC,Phone) Values('$sEmail',"
    . " '$sName','$sSurname','$sAddress','$sCity','$sPC','$sPhone')";
    $link = mysql_connect('localhost','ajax','J&J007')
    or die('Could not connect ' . mysql_errno());
    mysql_select_db($db_nombre ,$link) or die("Error selecting db.");
    if($oResult = mysql_query($sqlQuery))
    {
        $sResult = "Usuari registrat amb l'identificador: "
    . mysql_insert_id();
    }
    else
    {
        $sResult = "No s'ha pogut registrar l'usuari";
    }
    echo $sResult;
    mysql_close($link);
?>
```

Obté els diferents paràmetres del formulari i intenta crear un soci nou, tot retornant el resultat de l'operació al client.

Una vegada presentats els fitxers que formen l'exemple, s'explicarà cadascun dels passos involucrats en l'alta d'un usuari.

1) Càrrega inicial de la pàgina *index.html*

L'usuari carrega la pàgina *index.html* en què es mostra un formulari per a introduir tota la informació d'un soci nou:

```
<form method="post" action="saveData.php" onsubmit="newUser(); return false">
<table>
    <tr><td>Email</td><td><input name="email" width="20" /></td></tr>
    <tr><td>Nombre</td><td><input name="name"/>
    </td></tr>
    <tr><td>Apellidos</td><td><input name="surname"/>
    </td></tr>
    <tr><td>Dirección</td><td><input name="address"/>
    </td></tr>
    <tr><td>Población</td><td><input name="city"/>
    </td></tr>
```

```
    <tr><td>CP</td><td><input name="postCode"/>
    </td></tr>
    <tr><td>Teléfono</td><td><input name="phone"/>
    </td></tr>
</table>
<input type="submit" value="Save member" /><br>
</form>
<div id="divRequest"></div>
```

En el formulari s'estableix la propietat `action` cap a la pàgina `saveData.php`. L'esdeveniment `onsubmit` cridarà a la funció `newUser()` per a iniciar la comunicació i tot seguit retornarà `false` per a evitar que el formulari enviï la petició, ja que és l'objecte XMLHttpRequest, qui l'ha de fer.

Per a representar la resposta del servidor, es declara una divisió amb l'identificador `divRequest` després del formulari.

2) Transmissió de la petició POST

L'usuari ha introduït les dades al formulari i ha fet clic sobre "Save member", cosa que ha provocat una crida a la funció `newUser()`. La segona línia obté l'objecte del formulari en la variable `form`:

```
var form = document.forms[0];
```

Posteriorment, es prepara l'objecte `oXMLHttpRequest` per a fer la petició cridant al mètode `open()`:

```
oXMLHttpObject.open('post', form.action);
```

Com a mètode s'estableix `post`, i l'URL s'obté de l'atribut `action` del formulari, que en aquest cas és `saveData.php`.

En la línia següent s'indica el tipus de contingut usat per a enviar el formulari al servidor com a `application/x-www-form-urlencoded`:

```
oXMLHttpObject.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

Posteriorment, s'assigna una funció en la propietat `onreadystatechange` per a gestionar la recepció de la resposta.

En fer la transmissió de la petició, s'han d'enviar al cos tots els camps del formulari. La funció `getFormParams(form)` retorna una cadena amb els camps i valors del formulari passats per paràmetre:

```
function getFormParams(form){
    var params = new Array();
```

```
    for(var i = 0; i < form.elements.length; i++){
        var param = encodeURIComponent(form.elements[i].name);
        param += "=";
        param += encodeURIComponent(form.elements[i].value);
        params.push(param);
    }
    return params.join("&");
}
```

Aquesta funció recorre tots els camps del formulari i afegeix en una matriu el nom del camp seguit del seu valor amb el format següent:

```
NomCamp=ValorCamp.
```

Finalment, s'utilitza la funció `join` que retorna una cadena amb tots els elements de la matriu concatenats utilitzant el caràcter passat per paràmetre. Així, doncs, el conjunt de paràmetres quedaria de la manera següent:

```
NomCamp1=Valor1&NomCamp2=Valor2&...
```

Finalment, es crida al mètode `send()` passant per paràmetre la cadena de caràcters que representa el conjunt de camps del formulari que s'envia al cos de la petició.

```
oXMLHttpObject.send(getFormParams(form));
```

3) Generació de pàgina i resposta del servidor

En rebre la petició, el servidor HTTP executa la pàgina `addMember.php`. Aquesta obté els diferents paràmetres i els insereix en la base de dades i retorna al client el resultat de l'operació.

4) Recepció i actualització de pàgina

Igual que en l'exemple anterior, la funció assignada a la propietat `onreadystatechange` comprova que s'hagin rebut correctament les dades i les trasllada a la divisió reservada per a això.

1.4.5. La política del mateix origen

La política del mateix origen és una mesura de seguretat sobre JavaScript que tenen els navegadors i que determina els URL vàlids a què pot accedir una aplicació web. Aquesta política prevé que una aplicació carregada des d'un domini pugui obtenir dades situades en altres dominis.

Política del mateix origen

Tots els navegadors, inclosos Firefox, Chrome i Internet Explorer, disposen d'aquesta mesura de seguretat.

Això evita que una pàgina web amb codi maliciós pugui fer atacs o comprometre la confidencialitat d'una altra pàgina web.

Zones de seguretat

Internet Explorer utilitza una altra mesura anomenada *zones de seguretat* en què un conjunt de llocs que compleixen certes normes són exempts de la política del mateix origen.

Els navegadors basats en Mozilla consideren que dues pàgines tenen un mateix origen quan el protocol, el port i el domini són iguals. La taula següent mostra una sèrie d'URL i si són accessibles des de l'origen http://multimedia.uoc.edu/AJAX/index.html:

URL	Resultat	Raó
https://multimedia.uoc.edu/index.html	Error	Protocol diferent
http://multimedia.uoc.edu:8080/index.html	Error	Port diferent
http://www.otrodominio.com	Error	Domini diferent
http://multimedia.uoc.edu/index.html	Correcte	
http://multimedia.uoc.edu/info/index.html	Correcte	

La política del mateix origen també afecta les peticions fetes des d'un objecte XMLHttpRequest, en què els URL es limiten cap al mateix domini del qual prové l'aplicació. Aquesta restricció és un gran inconvenient per a molts desenvolupadors que necessiten obtenir dades des de dominis creuats.

Una possible solució és la d'utilitzar servidors intermediaris (*proxys*) en el costat del servidor, els quals redirigeixen les peticions a servidors en altres dominis i envien de tornada la resposta a l'aplicació.

El W3C va publicar una especificació anomenada CORS que consisteix en l'intercanvi d'encapçalaments específics per a informar el navegador dels diferents dominis i mètodes HTTP amb què pot accedir.

CORS

CORS és la sigla de *cros-origin resource sharing*. L'especificació CORS es pot consultar per Internet a:

- Cross-Origin Resource Sharing

El Firefox 3.5 i el Safary 4 implementen l'especificació CORS. L'Internet Explorer 8 n'implementa una part.

1.5. Avantatges i desavantatges de les diferents tècniques

Els navegadors guarden en l'historial de navegació les diferents peticions que es van fent durant la navegació. D'aquesta manera, l'usuari pot retrocedir a URL ja visitats, i també avançar-hi. Els URL generats des de marcs ocults per a comunicacions asíncrones també s'emmagatzemen en l'historial, de manera que l'usuari pot tornar enrere després de fer-se una recàrrega parcial.

Gmail

Gmail utilitza la tècnica dels marcs ocults, de manera que es pot navegar cap enrere i cap endavant per les diferents accions executades.

Un inconvenient de l'ús d'XMLHttpRequest és que les peticions generades des d'una instància no s'emmagatzemaran en l'historial i, en conseqüència, no és possible navegar entre les diferents peticions fetes. Si un requisit important és la navegació entre peticions, s'haurà d'optar per l'ús de marcs ocults.

Un altre inconvenient són les diferències entre les implementacions d'aquesta interfície pels diversos navegadors. Crear una aplicació web compatible amb els navegadors més populars requereix més lògica, inconvenient que pot ser contrarestat amb l'ús d'entorns de treball.

Entorns de treball

Un entorn de treball (en anglès, *framework*) és un conjunt de components de programari utilitzats per a facilitar el desenvolupament de certes funcions. En aquest cas, els entorns de treball AJAX es poden utilitzar en un projecte web per a afegir funcionalitats AJAX.

D'altra banda, XMLHttpRequest permet gestionar tots els aspectes de la comunicació amb un únic objectiu: canvi de capçaleres i mètode de la petició, seguiment dels canvis d'estat de la comunicació, codi de resposta del servidor, etc. XMLHttpRequest està pensat per a aquesta tasca i disposa d'un ampli conjunt de propietats, mètodes i esdeveniments que no ofereixen els marcs ocults.

1.6. AJAX accessible

L'accessibilitat permet l'accés al Web i als seus continguts a persones amb discapacitat visual, motriu, auditiva, cognitiva o neuronal. La WAI és una iniciativa del W3C que s'encarrega d'establir les especificacions i tecnologies necessàries per a una accessibilitat correcta del Web a persones amb discapacitat i gent d'edat avançada. Aquesta iniciativa proveeix d'estratègies, guies i recursos per a la creació d'aplicacions web accessibles.

WAI

WAI és la sigla de *Web accessibility initiative*, 'Iniciativa per a l'accessibilitat web'. Es pot consultar per Internet a:

- Web Accessibility Initiative (WAI)
- XML Path Language (XPath)

La majoria de llocs web no s'ajusta a les mesures de la iniciativa, cosa que dificulta l'accés a la majoria de gent amb discapacitats. A mesura que el Web creix ràpidament, més essencial es fa que sigui accessible, de manera que hi hagi una igualtat en l'accés i les oportunitats per a tothom. Per això, l'accessibilitat no depèn només de les tecnologies de suport com ara programes de lectura de pantalla, sintetitzadors de veu, línies Braille, etc. La responsabilitat més gran rau en els desenvolupadors web i el programari utilitzat per a produir i avaluar l'accessibilitat.

Tecnologies de suport

Les tecnologies de suport o *assistive technology* és un terme que inclou dispositius, instruments, tecnologies o programari que s'utilitza per a incrementar les capacitats funcionals de persones amb discapacitat.

Un dels objectius de la iniciativa WAI és el desenvolupament de guies i tècniques que descriguin solucions d'accessibilitat per a aplicacions web. Aquestes guies es consideren un estàndard internacional de l'accessibilitat web.

Enllaços d'interès

Aquest enllaç mostra els passos bàsics per a dur a terme projectes web amb accessibilitat:

• Implementation Plan for Web Accessibility

Aquest altre enllaç mostra informació detallada per a desenvolupadors:

• Web Content Accessibility Guidelines (WCAG) Overview

Per a avaluar l'accessibilitat i detectar problemes relacionats, hi ha eines que ajuden en l'avaluació d'una aplicació web. Tot i així, l'avaluació humana és imprescindible.

WAI-ARIA també defineix la manera de fer més accessibles les aplicacions web riques que disposen de continguts dinàmics i una interfície d'usuari avançada amb controls desenvolupats amb AJAX i tecnologies relacionades. Aquests controls han d'interactuar amb les tecnologies de suport que utilitzen les persones amb discapacitats. Per exemple, l'actualització dinàmica de contingut en una pàgina pot no ser detectada per un lector de pantalla que utilitza una persona invident.

WAI-ARIA

WAI-ARIA és la sigla de *Web accessibility initiative – access rich Internet applications*. S'encarrega de la iniciativa per a l'accessibilitat d'aplicacions RIA.

1.6.1. Exemple d'AJAX accessible

WAI-ARIA defineix el terme regions actives[8] AJAX, que permeten a les tecnologies de suport ser informades d'actualitzacions de continguts mitjançant atributs HTML.

(8)En anglès, *live regions*.

Algunes d'aquestes propietats són **`aria-live`** i **`aria-atomic`**.

1) La propietat `aria-live`

Aquesta propietat indica la prioritat amb la qual una tecnologia de suport ha de tractar les actualitzacions en regions actives. Els valors possibles són:

- `off`: és el valor per defecte i indica que la regió no és activa.
- `polite`: indica a la tecnologia de suport que es tracta d'una actualització normal i que no cal llegir-la fins que l'usuari completi la seva activitat actual.
- `assertive`: aquest valor indica una prioritat alta però no cal interrompre l'activitat de l'usuari.
- `rude`: aquest és el valor més alt i indica que l'usuari ha de ser interromput en actualitzar-se'n el contingut.

Valor `rude`

El valor `rude` no és recomanable si no és extremament necessari, ja que pot desorientar l'usuari si està duent a terme una altra tasca.

El fragment de document següent mostra dues divisions amb comportaments diferents en ser actualitzades:

```
<div id="NoticiaUltimaHora" aria-live="rude">

</div>

<div id="chat" aria-live="polite">

</div>
```

La primera divisió interromprà l'usuari en actualitzar-se. La segona divisió representa un xat i tots els elements que es vagin afegint, –per exemple paràgrafs–, es comunicaran d'una manera normal quan l'usuari acabi la seva activitat actual.

2) La propietat `aria-atomic`

Aquesta propietat indica a les tecnologies de suport si en actualitzar-se part d'una regió activa s'ha de presentar tot el contingut en el seu interior, o només l'actualitzat. Si el valor de la propietat és *true*, es presentarà tota l'àrea per complet. Si, al contrari, és *false*, es presentaran només els elements que s'han actualitzat.

El codi següent disposa de dues divisions incrustades que estableixen l'atribut `aria-atomic`:

```
<div id="LlistatNoticies" aria-atomic="false">
    <p>Llistat de notícies</p>
    <div id="destacades" aria-atomic="true">
        <p>Notícia 1</p>
    <p>Notícia 2</p>
    </div>
</div>
```

La divisió `"destacades"` defineix l'atribut `aria-atomic` a `true`. D'aquesta manera, si es modifiqués algun dels dos paràgrafs que conté, el dispositiu mostraria tot el seu contingut llegint, per exemple, tots els títols. En canvi, no es mostraria el paràgraf que es troba fora, ja que està contingut per una altra divisió que declara aquest atribut a `true`.

2. Intercanvi i gestió de documents XML

El format predilecte per a l'intercanvi de dades, i per al qual els navegadors ofereixen un ampli suport, és l'XML. En aquest punt, s'estudiarà com es poden obtenir, manipular i mostrar dades en format XML. Inicialment s'estudiarà l'estructura lògica d'un document XML a partir del DOM. Posteriorment, s'estudiarà com es poden obtenir documents XML d'una manera asíncrona en diferents navegadors i també mitjançant l'objecte XMLHttpRequest.

Una vegada obtingut un document, s'aprendrà a recórrer i seleccionar parts de la seva estructura mitjançant el DOM i XPath. També es podran mostrar documents o fragments XML mitjançant transformacions al format HTML amb plantilles.

2.1. Estructura lògica d'objectes DOM, XML i XHTML

El DOM proporciona una estructura lògica dels documents com un conjunt de nodes relacionats jeràrquicament entre ells en forma d'arbre. Cada node és representat per un objecte amb un conjunt de mètodes i propietats que permeten la manipulació i el recorregut del document.

La taula següent enumera els tipus de nodes més rellevants:

Tipus de node	Descripció
Element	Representa un element. Per exemple, `<html>`, `<p>`, etc.
Attribute	Representa un atribut d'un element.
Text	Contingut textual dins d'un element o atribut.
CDATASection	Representa una secció CDATA: delimita seqüències d'escapada en blocs de text a fi que no s'interpreti com a llenguatge marcat.
Comment	Representa un comentari.
Document	Node contenidor de tot el document. És l'element arrel de l'arbre de nodes.
DocumentType	Representa un node `<!DOCTYPE...>`.
DocumentFragment	Representa un objecte de document lleuger que pot contenir fragments XML.

Les constants i el valor numèric de cadascun dels tipus de nodes exposats en la taula anterior són els següents:

Valor numèric	Nom de la constant
1	*NODE_ELEMENT*
2	*NODE_ATTRIBUTE*
3	*NODE_TEXT*
4	*NODE_CDATA_SECTION*
8	*NODE_COMMENT*
9	*NODE_DOCUMENT*
10	*NODE_DOCUMENT_TYPE*
11	*NODE_DOCUMENT_FRAGMENT*

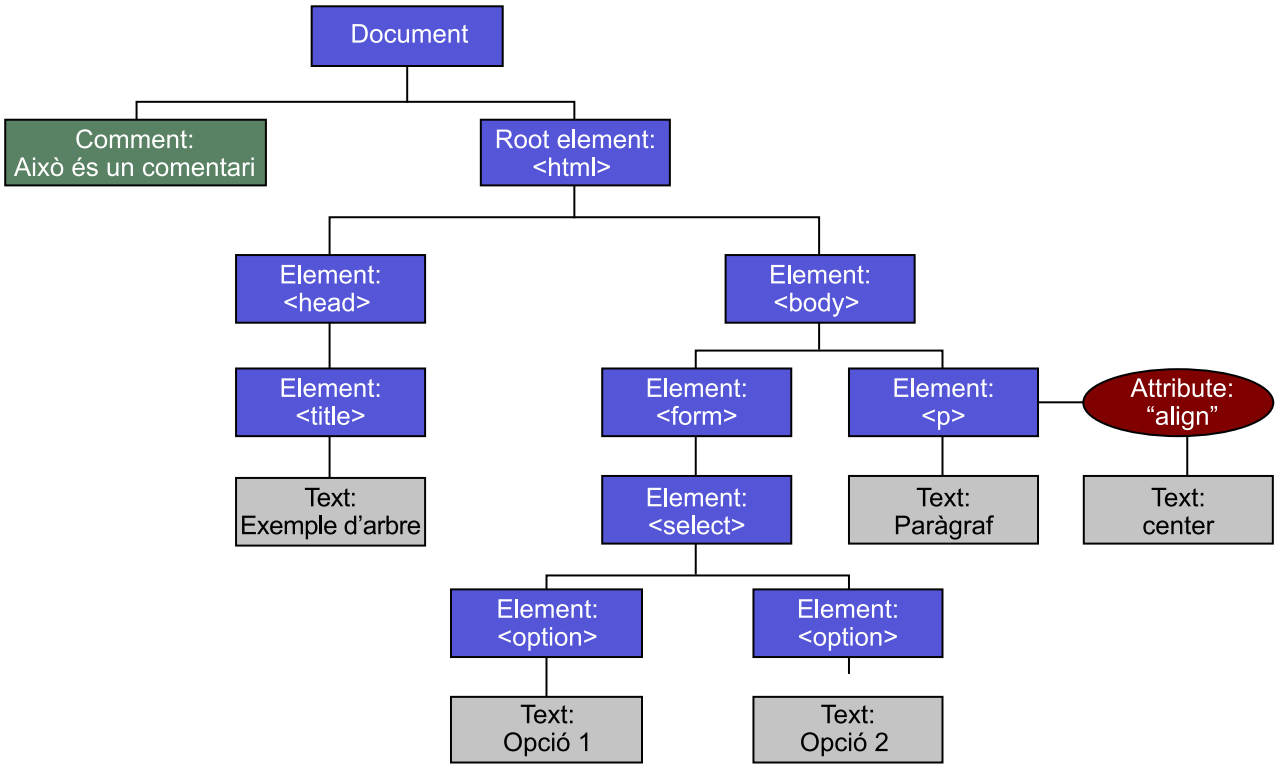
2.1.1. Exemple d'estructura lògica d'un document

A partir del document HTML següent, se'n mostrarà l'arbre lògic:

```
<!-Això és un comentari-->
<html>
<head>
    <title>Exemple d'arbre</title>
</head>
    <body>
        <form name="userForm">
            <select>
                <option>Opció 1</option>
                <option>Opció 2</option>
            </select>
        </form>
        <p align="center">Paràgraf</p>
    </body>
</html>
```

L'arbre resultant d'un objecte DOM del document HTML anterior és el que es mostra en la figura 8.

Figura 8. Exemple d'arbre HTML



Objecte de document

Un objecte d'un document és una instància del DOM del document.

El primer node de la jerarquia del document és de tipus `Document`. Qualsevol objecte de document conté aquest tipus de node i és el que es proporciona en crear un DOM. A partir d'aquest node es té accés a tot el document.

Els diferents tipus de nodes que apareixen són `Document`, `Element`, `Attribute`, `Text` i `Comment`. Els nodes de tipus `Node_Element` corresponen a totes les etiquetes HTML, `<html>`, `<head>`, `<title>`, `<body>`, `<form>`, `<p>`, `<select>` i `<option>`. L'el·lipse que depèn del node `<p>` representa un node de tipus `Attribute`. Els textos del títol del document, de l'atribut `"Align"` i textos de les opcions `<option>` són nodes de tipus `Text`.

2.2. Càrrega de documents XML

El primer pas per a tractar un document és carregar-lo per a disposar d'una instància del DOM. Hi ha diferents maneres d'obtenir un document XML, segons cada navegador. L'objecte DOM obtingut en carregar un document és de tipus `Document`.

En aquest subapartat s'estudiarà com s'ha de carregar un document XML amb els navegadors Internet Explorer i Mozilla Firefox. Posteriorment, s'explicarà com es pot obtenir un objecte de document fent ús d'XMLHttpRequest per a múltiples navegadors.

2.2.1. Càrrega d'un document XML en l'Internet Explorer

MSXML[9] és un conjunt de biblioteques COM de Microsoft per al processament d'XML. MSXML ofereix una implementació de DOM a partir del qual es poden crear i manipular documents.

(9)*MSXML* és la sigla de *Microsoft XML core services.*

Per a crear un objecte DOM i carregar un document XML des de l'Internet Explorer s'ha d'instanciar MSXML com un objecte ActiveX:

```
var oXmlDom = new ActiveXObject("Microsoft.XmlDom");
```

Com que hi ha diferents versions de MSXML, sempre és preferible crear una instància de la versió més recent possible:

```
function createDOMXML(){
    var aVersions = ["MSXML2.DOMDocument.6.0",
                     "MSXML2.DOMDocument.5.0",
                     "MSXML2.DOMDocument.4.0",
                     "MSXML2.DOMDocument.3.0",
                     "MSXML2.DOMDocument'',
                     "Microsoft.XmlDom"];
    for(var i = 0; i < aVersions.length; i++){
        try{
            var oXmlDom = new ActiveXObject(aVersions[i]);
            return oXmlDom;
        } catch(oError){

        }
    }
    throw new Error("Couldn't create a MSXML instance.");
}
oXML = createDOMXML();
```

El vector `aVersions` conté un conjunt de cadenes de versions diferents de DOM MSXML. El bucle recorre des de la versió més recent fins a la primera mentre intenta instanciar l'objecte. Si es produeix una excepció, en la iteració següent s'intentarà amb una versió anterior fins a obtenir una instància.

Una vegada obtingut l'objecte es pot emprar per a obtenir objectes de documents XML a partir de dos mètodes: `load(sURL)` i `loadXML(sXML)`, que s'expliquen a continuació:

- ***load(sURL)***

Aquest mètode inicialitza l'objecte a partir d'un document situat en l'URL passat per paràmetre:

```
oXML.load("/XML/xmlFile.xml");
```

Igual que l'objecte XMLHttpRequest, és possible indicar si la comunicació s'ha de fer en manera asíncrona o síncrona amb la propietat `async`, i s'ha d'establir `true` o `false`, respectivament. Per defecte, la comunicació es duu a terme en manera asíncrona.

Per a controlar l'estat de la comunicació, MSXML també disposa de les propietats `readyState` i `onreadystatechange`:

```
oXML = createDOMXML();
oXML.onreadystatechange = function(){
        if(oXML.readyState == 4)
            alert("Document loaded.");
    };
oXML.load("/XML/xmlFile.xml");
```

En el codi anterior, s'estableix en la propietat `onreadystatechange` una funció JavaScript que en cas que l'estat del document sigui 4 (`loaded`), es mostrarà un missatge que indiqui que les dades s'han carregat correctament.

- ***loadXML(sXMLString)***

Un objecte DOM de MSXML també proporciona el mètode `loadXML(XMLString)` que inicialitza l'objecte a partir d'una cadena que conté el document XML. La crida és síncrona, ja que la lectura és immediata:

```
var sXML =
" <message>
      <from>abc@uoc.edu</from>
      <to>jsirvent123@uoc.edu</to>
   </message>
";
oXML.loadXML(sXML);
```

La variable `oXML` conté l'objecte DOM construït a partir de la cadena de caràcters.

2.2.2. Càrrega d'un document XML en el Mozilla Firefox

El Firefox ofereix una interfície per a obtenir documents DOM buits, és a dir, que no representen cap document existent. Es tracta del mètode `createDocument()` de la interfície `DOMImplementation`.

```
function getNewDocument(){
    if(document.implementation &&
            document.implementation.createDocument){
```

```
        var domXML = document.implementation.createDocument("", "", null);
        return domXML;
    }
}
```

La funció anterior comprova inicialment que el navegador disposi de l'objecte `document.implementation`, que torna una instància de la interfície `DOMImplementation`. Si és així, crida al mètode `createDocument()` per a obtenir un document buit passant tres arguments: el primer especifica un espai de noms per al document. El segon paràmetre accepta una cadena que especificarà el nom del node arrel del document. En el tercer paràmetre s'especifica el tipus de document; per a tenir més compatibilitat, és recomanable passar-lo com a `null`.

Una vegada creat l'objecte, es pot sol·licitar un document XML a partir del seu URL amb el mètode `load`:

```
var oDOM = getNewDocument ();
oDOM.onload = function(){
    alert("XML Document loaded");
}
oDOM.load("lista.xml");
```

En l'exemple anterior, s'assigna una funció en l'esdeveniment `onload`, que serà invocada quan el document estigui carregat.

Càrrega del document

La càrrega es fa per defecte d'una manera asíncrona. Això es pot modificar mitjançant la propietat `async` del document, establint `true` o `false`.

A diferència de MSXML, l'objecte obtingut mitjançant `createDocument()` no disposa de les propietats `readyState` i del programa controlador `onreadystatechange`. Per a obtenir un document a partir d'una cadena, es pot fer ús del mètode `parseFromString` de la classe `DOMParser`. Aquest torna un objecte DOM d'XML a partir d'una cadena de caràcters passada per paràmetre:

```
var sXML =
"<message>
     <from>abc@uoc.edu</from>
     <to>asanchez123@uoc.edu</to>
 </message>";
var oParser = new DOMParser();
var oDOM = oParser.parseFromString(sXML,"text/xml");
```

Com a primer paràmetre es passa la cadena que conté el document. El segon paràmetre permet indicar el tipus de contingut, en aquest cas XML.

2.2.3. Carregar un document XML des d'un URL per a múltiples navegadors

Tant l'Internet Explorer com el Firefox ofereixen suport per a XML. Ambdues implementacions són diferents i incompatibles, per la qual cosa fer ús d'una d'elles descartaria els clients de la resta de navegadors.

La interfície XMLHttpRequest permet obtenir documents XML i proporcionar objectes de documents XML mitjançant la propietat `responseXML`.

L'exemple següent recupera un document XML des del servidor i torna un objecte DOM. Es tindrà en compte que l'objecte XMLHttpRequest està creat i representat mitjançant la variable `oXMLHttpObject`:

```
function getXMLDOMObject(xmlFileName){
    oXMLHttpObject.open('GET', xmlFileName);
    oXMLHttpObject.onreadystatechange = function(){
        if(oXMLHttpObject.readyState == 4){
            if(XMLHttpObject.status == 200){
                oXMLDOMObject = oXMLHttpObject.responseXML;
            }
            else{
                alert("There was a problem with the request.");
            }
        }
    XMLHttpObject.send(null);
    }
}
```

La funció `getXMLDOMObject(xmlFileName)` utilitza XMLHttpRequest per a sol·licitar al servidor el fitxer passat per paràmetre. Una vegada rebut correctament el document, s'obté l'objecte DOM amb la propietat `responseXML`. Aquest objecte creat dependrà de la implementació del navegador i això s'ha de tenir en compte a l'hora d'usar-ne les propietats i els mètodes.

2.3. Recorregut d'un document mitjançant el DOM

La interfície `Node` implementada pels diferents tipus de nodes ofereix un conjunt de propietats i mètodes per a l'accés i recorregut del document. Les propietats més importants s'enumeren en la taula següent.

Interfície `Node`

La interfície `Node` està definida en l'especificació Document Object Model Level 1 pel W3C.

Propietat	Descripció
attributes	Proporciona un vector amb els atributs del node actual. Només aplicable a nodes de tipus `Element`.
childNodes	Retorna un vector amb els nodes fill.

Propietat	Descripció
documentElement	Retorna el node que representa el fill directe del node arrel `Document`.
firstChild	Obté el primer node fill. Si no existeix, retornarà `null`.
lastChild	Obté l'últim node fill.
nextSibling	Obté el node següent dins de la llista de nodes fill a què pertany un node.
nodeName	Obté el nom del node.
nodeType	Retorna el tipus de node.
nodeValue	Retorna el text contingut en l'etiqueta.
parentNode	Fa referència al pare d'un node.
previousSibling	Obté el node anterior dins de la llista de nodes fill a què pertany un node.
tagName	Retorna el nom de l'etiqueta.

Els mètodes següents pertanyen als tipus de node `Element` i `Document`:

Nom	Retorn
appendChild(appendedNode)	Afegeix un nou node fill al final de la llista de nodes.
attributes	Propietat. Proporciona un vector amb els atributs d'un node. Només aplicable a nodes de tipus `Element`.
cloneNode(deep)	Retorna una còpia del node. Si el paràmetre booleà `deep` és `true`, també seran clonats els nodes fill del node.
getAttribute(sName)	Retorna el valor de l'atribut amb el nom passat per paràmetre.
getAttributeNode(sName)	Retorna el node que representa l'atribut amb el nom passat per paràmetre.
getElementsByTagName(sName)	Retorna un conjunt de nodes `Element` amb el nom passat per paràmetre que descendeixen d'un node.
hasAttributes()	Retorna un booleà que indica si el node conté atributs.
hasChildNodes()	Retorna un booleà que indica si el node té nodes fill.
insertBefore(insNode, node)	Insereix un node fill abans que el segon node.
removeChild(node)	Elimina un node fill.
replaceChild(insNode, replNode)	Reemplaça un node fill per un altre passat per paràmetre.

2.3.1. Manipular els espais en blanc en el Mozilla

Els navegadors basats en Mozilla tracten els espais en blanc i els salts de línia d'un document XML com a nodes de tipus text. En carregar un DOM XML s'obté un document l'arbre del qual conté més nodes del que s'espera. Això afecta el recorregut i la interpretació de l'arbre, ja que es recorren nodes innecessaris.

La funció JavaScript següent elimina nodes de text buits d'un objecte DOM passat per paràmetre:

```
function removeWhiteSpace(xmlDOM){
    var notWhitespace = /\s/;
    var i;
    for (i = 0; i < xmlDOM.childNodes.length; i++){
        var currentNode = xmlDOM.childNodes[i];
        if(currentNode.nodeType == 1){
            removeWhiteSpace(currentNode);
        }
        if((notWhitespace.test(currentNode.nodeValue))
            &&(currentNode.nodeType == 3)){
        xmlDOM.removeChild(xmlDOM.childNodes[i--]);
        }
    }
    return xmlDOM;
}
```

Aquesta funció recorre tots els nodes de l'arbre d'una manera recursiva. Si un node és de tipus text i conté un espai en blanc, s'elimina de l'arbre resultant.

Vegeu també

Els exemples numerats que es presenten al llarg de tot aquest subapartat estan també disponibles en línia.

L'exemple 10 posa en pràctica algunes propietats per al recorregut de nodes.

Exemple 10

A partir d'un objecte XMLHttpRequest, s'obté el DOM d'un document XML i se'n recorre l'estructura pels diferents nodes. Els fitxers són:

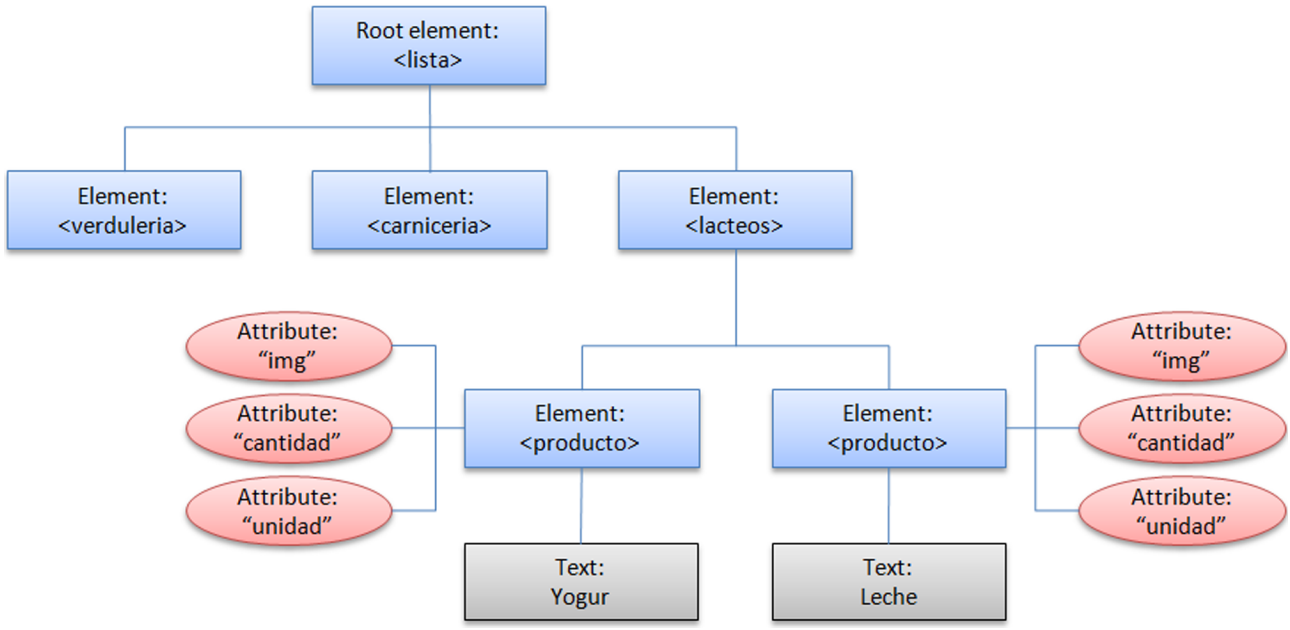
- **lista.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<lista>
  <verduleria>
    <producto imagen="patatas.jpg" cantidad="2" unidad="Kg">
    Patatas</producto>
    <producto imagen="zana.jpg" cantidad="1" unidad="Kg">
    Zanahorias</producto>
    <producto imagen="huevos.jpg" cantidad="1" unidad="Docena">
    Huevos</producto>
  </verduleria>
  <carniceria>
```

```
    <producto cantidad="500" unidad="Gramos">Jamón</producto>
    <producto cantidad="250" unidad="Gramos">Pollo</producto>
  </carniceria>
  <lacteos>
    <producto img="yogur.jpg" cantidad="6">Yogur</producto>
    <producto img="leche.jpg" cantidad="2" unidad="litro">
    Leche</producto>
   </lacteos>
</lista>
```

La vista lògica en forma d'arbre del document anterior es podria representar com es mostra en la figura 9.

Figura 9. Arbre de nodes del document lista.xml



Per a simplificar la figura no es mostren els nodes ni els atributs descendents dels elements <verduleria> i <carniceria>.

- **recorrido.xml**

```
<html>
<head>
<title>Recorrido de un documento XML</title>
<script type="text/javascript" src="../script/AJAX.js"></script>
<script type="text/javascript">
function mostrarLista(){
    var oXMLHttpRequest = createXMLHttpRequestObject();
    oXMLHttpRequest.open('GET', 'lista.xml',false);
    oXMLHttpRequest.send(null);
    if((oXMLHttpRequest.readyState == 4)&&(oXMLHttpRequest.status == 200)){
        var oDomXML = removeWhiteSpace(oXMLHttpRequest.responseXML);
        var oLista = oDomXML.documentElement;
        var oVerduleria = oLista.firstChild;
        var oLacteos = oLista.lastChild;
        var oCarniceria = oVerduleria.nextSibling;
        oCarniceria = oLacteos.previousSibling;
        mostrarProductos(oVerduleria);
        mostrarProductos(oLacteos);
        mostrarProductos(oCarniceria);
```

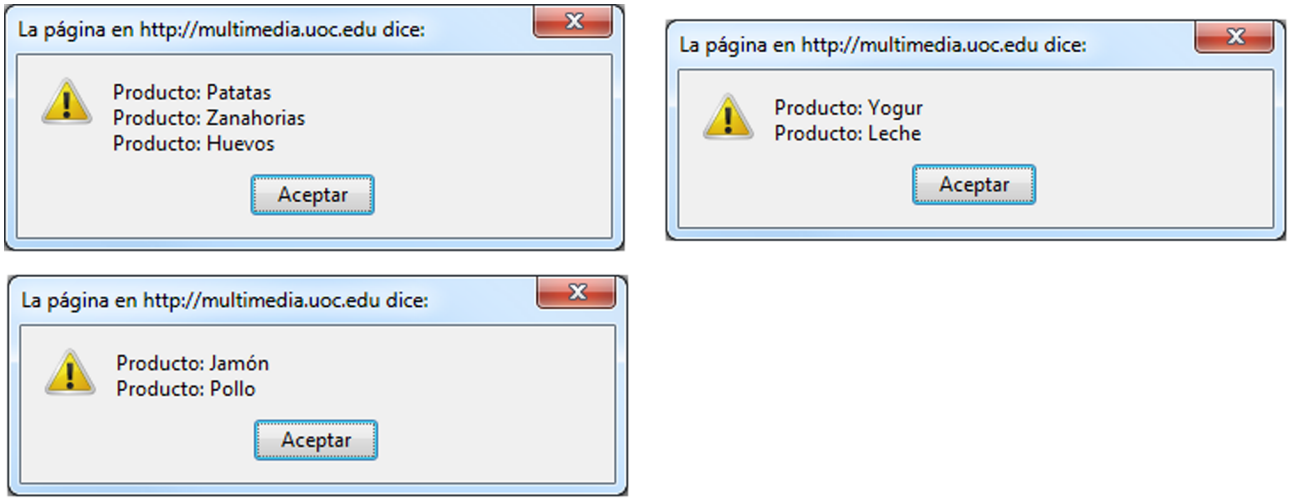
```
        }
        else{
            alert('XMLHttpRequest: error en la petición.');
        }
    }

    function mostrarProductos(oNode){
        var aProducts = oNode.childNodes;
        var sText = '';
        for(var i = 0; i < aProducts.length; i++){
            if(aProducts[i].nodeType == 1){
                if(aProducts[i].firstChild.nodeType == 3){
                    sText += 'Producto: ' + aProducts[i].firstChild.nodeValue + '\n';
                }
            }
        }
        alert(sText);
    }

    </script>
    </head>
    <body>
    <button onclick="mostrarLista()">Mostrar lista</button>
    </body>
    </html>
```

La pàgina `recorrido.html` mostra tots els productes de la llista. Concretament, mostra el text contingut dins de l'etiqueta `<producto>` de cadascuna de les seccions `<verduleria>`, `<carniceria>` i `<lacteos>`. El resultat de l'exemple es mostra en la figura 10.

Figura 10. Resultat de l'exemple



La funció `mostrarLista()` obté el DOM del document `lista.xml` i elimina els nodes de text buits amb la funció `removeWhiteSpace()`. A partir de l'objecte obtingut, es recorre l'arbre per a obtenir els nodes `<verduleria>`, `<carniceria>` i `<lacteos>`:

```
var oDomXML = removeWhiteSpace(oXMLHttpRequest.responseXML);
var oLista = oDomXML.documentElement;
var oVerduleria = oLista.firstChild;
var oLacteos = oLista.lastChild;
var oCarniceria = oVerduleria.nextSibling;
```

La propietat `documentElement` retorna l'element situat en l'arrel, que correspon al node `<lista>`. A partir d'aquest es pot obtenir el primer i l'últim node fill amb les propietats `firstChild` i `lastChild`, que en aquest cas serà `<verduleria>` i `<lacteos>`, respectivament. El node `<carniceria>` és germà de `<verduleria>` i `<lacteos>`. Per a obtenir-lo des de `oVerduleria` es pot utilitzar la propietat `nextSibling` i `previousSibling` des de `oLacteos`. És a dir, les dues instruccions següents retornarien el mateix objecte:

```
var oCarniceria = oVerduleria.nextSibling;
oCarniceria = oLacteos.previousSibling;
```

Posteriorment, es crida tres vegades a la funció `mostrarProductos()` passant per paràmetre cadascun dels nodes obtinguts. `mostrarProductos()` recorre els nodes fill del node passat per paràmetre i mostra el text contingut en l'etiqueta:

```
function mostrarProductos(oNode){
    var aProducts = oNode.childNodes;
    var sText = '';
    for(var i = 0; i < aProducts.length; i++){
        if(aProducts[i].nodeType == 1){
            if(aProducts[i].firstChild.nodeType == 3){
                sText += 'Producto: ' + aProducts[i].firstChild.nodeValue + '\n';
            }
        }
    }
    alert(sText);
}
```

La primera línia obté una matriu amb tots els nodes fill a través de la propietat `childNodes`. Després, recorre tots els nodes de la matriu i per a cadascun comprova que el node sigui de tipus `Element`. Si és així, s'accedirà al primer fill, es comprovarà que sigui de tipus text, i se'n concatenarà el contingut amb la variable `sText`, que acumula el text dels productes. Finalment, es mostrarà la cadena construïda.

2.3.2. Accés a elements XML a partir del nom

En l'exemple anterior s'ha mostrat el text de tots els nodes de tipus `<producto>`. El codi depèn del document i qualsevol canvi que afecti l'estructura de l'arbre podria fer que no funcionés correctament.

Els nodes de tipus `Document` ofereixen el mètode `getElementsByTagName()`, que retorna un vector de nodes corresponents a totes les etiquetes amb el nom passat per paràmetre que depenen del node que fa la crida. Si aquest mètode s'aplica sobre el node arrel, el resultat serien tots els elements `<producto>` que es trobin a tot el document.

Nodes de tipus `Document` i `Element`

Els nodes de tipus `Document` i `Element` disposen del mètode `getElementsByTagName()`.

L'exemple 11 mostra com es poden obtenir tots els productes, representats per l'etiqueta `<product>` sobre el document XML corresponent al fitxer `lista.xml` de l'exemple anterior.

Exemple 11

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Recorrido de un documento XML</title>
<script type="text/javascript" src="../script/Ajax.js"></script>
<script type="text/javascript">
function mostrarLista(){
    var oXMLHttpRequest = createXMLHttpRequestObject();
    oXMLHttpRequest.open('GET', 'lista.xml',false);
    oXMLHttpRequest.send(null);
    //Si s'ha obtingut la llista correctament...
    if((oXMLHttpRequest.readyState == 4)&&
        (oXMLHttpRequest.status == 200)){
        //S'obté l'objecte de document
        var oDomXML = oXMLHttpRequest.responseXML;
        //La variable aProducts obté un vector
        //amb els nodes de tipus producto
        var aProductos = oDomXML.getElementsByTagName('producto');
        var sText = '';
        //S'acumulen els productes en una cadena de caràcters
        for(var i = 0; i < aProductos.length; i++){
            sText += 'Producto: ' +
            aProductos[i].firstChild.nodeValue + '\n';
        }
        //Es mostra la llista de productes
        alert(sText);
    }
    else{
        alert('XMLHttpRequest: error en la petición.');
    }
}
</script>
</head>
<body>
    <button onclick="mostrarLista()">Mostrar lista</button>
</body>
</html>
```

Una vegada obtingut l'objecte del document `lista.xml`, es crida al mètode `getElementsByTagName()` passant per paràmetre `'producto'`. L'objecte retornat és un matriu amb tots els elements `<producto>` de tot el document. La instrucció `for` recorre tots els elements i mostra el text que contenen.

2.3.3. Accés als atributs d'un element i els seus valors

Els nodes de tipus `Element` poden contenir atributs. Per a accedir-hi, es pot utilitzar la propietat `atrributes`, que retorna una matriu amb tots els nodes de tipus `Attribute`. Si es coneix el nom dels atributs a què es vol accedir, es poden utilitzar els mètodes `getAttribute(attributeName)` i `getAttributeNode(attributeName)`: el primer retorna el valor de l'atribut passat per paràmetre, i el segon retorna el node.

L'exemple 12 mostra la llista de la compra i afegeix informació continguda en els atributs que conté l'etiqueta `<producto>`.

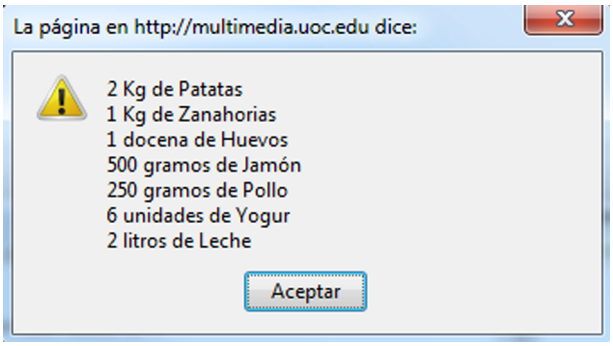
Exemple 12

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Recorrido de un documento XML</title>
<script type="text/javascript" src="../script/Ajax.js"></script>
<script type="text/javascript">
function mostrarLista(){
    var oXMLHttpRequest = createXMLHttpRequestObject();
    oXMLHttpRequest.open('GET', 'lista.xml',false);
    oXMLHttpRequest.send(null);
    //Si s'ha rebut la llista correctament...
    if((oXMLHttpRequest.readyState == 4)&&
           (oXMLHttpRequest.status == 200)){
        //S'obté l'objecte del document lista.xml
        var oDomXML = oXMLHttpRequest.responseXML;
        //S'obté un vector amb tots els nodes <producto>
        var aProductos = oDomXML.getElementsByTagName('producto');
        //Es crea una matriu i en cada posició s'afegeix un producte
        //amb la seva descripció
        var aListaCompra = new Array();
        for(var i = 0; i < aProductos.length; i++){
            var oNodo = aProductos[i];
            var sCantidad = oNodo.getAttribute('cantidad');
            var sUnidad = oNodo.getAttribute('unidad');
            var sProducto = oNodo.firstChild.nodeValue;
            aListaCompra.push(sCantidad + ' ' + sUnidad + ' de ' +
               sProducto);
        }
        //Es concatenen totes les descripcions de productes
        //afegint un salt de línia entre ells
        alert(aListaCompra.join('\n'));
    }
    else{
        alert('XMLHttpRequest: error en la petición.');
    }
}
</script>
</head>
<body>
    <button onclick=''mostrarLista()''>Mostrar lista</button>
</body>
</html>
```

Una vegada obtingut el DOM del document `lista.xml`, s'obtenen tots els nodes `<producto>` mitjançant el mètode `getElementsByTagName()`. Per a cadascun, es crea una cadena que conté els atributs de quantitat, unitat i el text de l'element, en diferents variables. La concatenació de les variables s´hi afegeix en una matriu.

Finalment, es mostra la matriu cridant a la funció `join()`, passant per paràmetre el caràcter `\n`, que representa un salt de línia. El resultat és una cadena amb la concatenació de totes les cadenes contingudes en la matriu amb el caràcter de salt de línia entre cada element:

Figura 11. Llista de productes del fitxer lista.xml



2.4. Gestió d'errors

En carregar un document XML es poden produir errors. Per exemple, el document pot no trobar-se en l'URL especificat, el document pot no estar ben format, etc. La gestió d'errors permet detectar si s'ha produït un error en obtenir un document XML i obtenir informació sobre la incidència.

2.4.1. Gestió d'errors en l'Internet Explorer

Un objecte de document XML creat amb MSXML proporciona l'objecte `parseError`. Una vegada carregat un document XML, podem comprovar el resultat de l'operació mitjançant la propietat `errorCode` de `parseError`. Sempre que `errorCode` sigui diferent de 0, indica que s'ha produït un error.

A continuació, es mostren les diferents propietats de l'objecte `parseError`:

Propietat	Descripció
errorCode	Codi d'error.
filePos	Especifica la posició del document en què s'ha produït l'error.
line	Indica el número de línia del document en què s'ha produït l'error.
linePos	Indica la columna dins de la línia de la propietat anterior en què s'ha produït l'error.
reason	Text descriptiu sobre l'error.
srcText	Text que conté la línia que ha produït l'error.
url	URL del document.

Exemple d'error en la càrrega d'un document

L'exemple següent carrega un document XML a partir d'una cadena. La cadena està mal formada, i provoca un error:

```
function exception(){
    var oDOMXML = createDOMXML();
    var sXML =
      "<message>abc@uoc.edu</from>
         <to>asanchez123@uoc.edu</to>
       </message>";
    oDOMXML.loadXML(sXML);
    if(oDOMXML.parseError.errorCode != 0){
        alert("Couldn't load XML Document: " +
            oDOMXML.parseError.reason);
    }else{
        alert("XML document loaded successfully");
    }
}
```

Després de l'etiqueta `<message>`, faltaria l'obertura de `<from>`. En carregar el document, es comprova l'estat de la propietat `errorCode` i si s'ha produït un error se'n mostra la descripció a l'usuari.

2.4.2. Gestió d'errors en el Firefox

Quan falla el procés de càrrega d'un document, en lloc de llançar una excepció, es retorna un altre document XML amb la descripció de l'error.

La funció següent intenta carregar un objecte DOM d'XML a partir d'una cadena de caràcters. El document està mal format, i es genera un document d'error:

```
function exception(){
    var sXML = "<message>abc@uoc.edu</from><to>asanchez123@uoc.edu</to></message>";
    var oParser = new DOMParser();
    var oDOM = oParser.parseFromString(sXML,"text/xml");
    var oXMLSerializer = new XMLSerializer();
    var sXMLError = oXMLSerializer.serializeToString(oDOM);
    if(oDOM.documentElement.tagName == "parsererror"){
        alert("Couldn't load XML Document: " + sXMLError);
    }else{
        alert("XML document loaded successfully");
    }
}
```

El codi anterior comprova el valor del node arrel i el compara amb la cadena `"parsererror"` per a comprovar si s'ha produït algun error en carregar el document. En aquest cas, el contingut de la variable `sXMLError` és:

```
<?xml version="1.0" encoding="UTF-8"?>
<parsererror xmlns="http://www.mozilla.org/newlayout/xml/parsererror.xml">
Error de lectura XML: etiqueta sin pareja. Se esperaba: <//message>.
Ubicación: file:///C:/xampp/htdocs/DOMXMLIE7/ExceptionFF.html
Número de línea 1, columna 23:
   <sourcetext>
```

```
    <message>abc@uoc.edu</from><to>asanchez123@uoc.edu<>/to<></message>
                     ----------------------^
    </sourcetext>
  </parsererror>
```

La descripció de l'error conté el text descriptiu de l'error i la fracció del codi en què es va originar.

2.5. Selecció de nodes amb XPath

XPath és un llenguatge que permet encaminar i seleccionar parts d'un document XML a partir de rutes de localització i expressions. En aquest punt es farà una introducció breu d'aquest llenguatge.

XPath

XPath és l'acrònim d'*XML path language*, 'llenguatge de rutes XML'. XPath és un llenguatge utilitzat en navegadors i diferents plataformes de desenvolupament. És una recomanació del W3C i es pot consultar per Internet a: *XML Path Language (XPath)*.

Les rutes de localització encaminen un node dins d'un document XML de manera similar a la localització d'arxius i directoris dins d'un sistema de fitxers. Per exemple, si en un sistema UNIX es tingués un directori des de l'arrel anomenat `DirectoriPare` i un subdirectori `DirectoriFill` que contingués un fitxer anomenat `doc.txt`, s'accediria al fitxer a partir de la ruta següent:

```
/DirectoriPare/DirectoriFill/doc.txt
```

En un sistema Windows la ruta podria ser la següent:

```
C:\DirectoriPare\DirectoriFill\doc.txt
```

De manera semblant, la ruta de localització `/Directory/Subdirectory/file` permetria seleccionar el node `<file>` del document XML següent:

```
<Directory>
    <Subdirectory>
        <file name="doc.txt"/>
    </Subdirectory>
</Directory>
```

En XPath, les expressions s'avaluen sobre un node que es coneix com a *node de context* i en ser processades s'obté un objecte. El tipus d'objecte tornat a partir d'una expressió pot ser un dels tipus següents:

- *node-set*, una col·lecció desordenada de nodes sense duplicats;
- un valor booleà;
- un valor numèric;
- una cadena de caràcters.

En aquest mòdul només s'analitzen les expressions que retornen una col·lecció de nodes de tipus *node-set*.

2.5.1. Selecció de nodes

Els símbols i operadors de la taula següent es poden combinar per a construir expressions que retornen un conjunt de nodes en un *node-set*:

Símbol	Descripció
nodename	Seleccionen tots els nodes fill amb el nom especificat.
/	Identifica el node arrel i també permet especificar rutes separant nivells de nodes.
//	Selecciona els nodes per tot l'arbre.
.	Identifica un node en relació amb el node de context.
..	Identifica el node pare d'un node.
|	Fa la unió entre dues expressions que retornen un *node-set*.
@	Selecciona atributs.

Exemple de selecció de nodes

Els exemples d'expressions que s'exposen a continuació parteixen del document XML següent:

```
<menu>
  <breakfast time="7:00 am">
    <food price="1">toasts</food>
    <food price="1">eggs</food>
    <food price="3">beans</food>
    <drink price="1">Orange juice</drink>
    <tea price="1">Tea with milk</tea>
  </breakfast>
  <lunch time="12:00 pm">
    <food price="3">beans</food>
    <food price="1">bread</food>
    <drink price="2">coke</drink>
  </lunch>
  <dinner time="8:00 pm">
    <food price="3">salad</food>
    <drink price="1">water</drink>
  </dinner>
</menu>
```

Les expressions es poden declarar de manera absoluta o relativa al node de context. Si el node de context és `<menu>` i es volen obtenir tots els elements `<food>` dins de `<dinner>`, s'aplicaria l'expressió següent:

```
dinner/food
```

L'expressió anterior és relativa al node de context. Concretament, tornaria el node o conjunt de nodes `<food>` que depenen del node `<dinner>` a partir del node de context. Si el node de context no fos, en aquest cas, `<menu>`, no retornaria cap resultat. L'expressió absoluta equivalent és:

```
/menu/dinner/food
```

En les expressions absolutes, retornen el mateix resultat sigui quin sigui el node de context dins d'un mateix arbre.

La taula següent mostra una sèrie d'exemples d'expressions XPath:

Expressió	Descripció
/	Retorna el node `Document`.
/menu/breakfast	Retorna els nodes `<breakfast>`.
//drink	Retorna tots els nodes de tipus `<drink>` situats en qualsevol part del document.
drink	Retornen tots els nodes fill `<drink>` que siguin fills directes del node de context.
../dinner/food	Descendeix un nivell sobre el node de context i localitza el node `<dinner>`. A partir d'allà, retorna tots els nodes `<food>`.
drink | food	Selecciona tots els elements `<drink>` i `<food>` fills directes del node de context.
/menu/breakfast@time	Retorna el node corresponent a l'atribut `time` de l'element `<breakfast>`.

2.5.2. Ús d'XPath en l'Internet Explorer

Un objecte DOM d'XML en l'Internet Explorer disposa de dos mètodes per a la selecció de nodes mitjançant expressions XPath:

- `selectSingleNode(sExpression)`: torna el primer node trobat després de la selecció. Com a paràmetre sol·licita una cadena amb l'expressió XPath.

- `selectNodes(sExpression)`: torna una col·lecció de nodes a partir de l'expressió passada per paràmetre.

Exemple d'ús d'XPath en l'Internet Explorer

A partir del document `Menu.xml`, s'obtindrà el primer node `<food>` dins de `<breakfast>`. Es considera que la variable `oDOM` representa l'objecte del document `menu.xml`:

```
function getFirstFoodBreakfast(oDOM){
return oDOM.documentElement.selectSingleNode("/menu/breakfast/food");
}
```

La funció anterior executa l'expressió `/menu/breakfast/food`, que retorna tots els nodes `<food>` dins de l'etiqueta `<breakfast>`. El node de context és el node `Element` i l'expressió especifica una ruta absoluta. Encara que dins del node `<breakfast>` hi ha tres nodes `<food>`, la funció `selectSingleNode()` únicament retornarà el primer.

És possible fer la consulta a partir de qualsevol node. La funció següent fa la mateixa acció que l'anterior:

```
function getFirstFoodBreakfast(oDOM){
var oBreakfast = oDOM.documentElement.firstChild;
var oneFood = oBreakFast.selectSingleNode("food");
return oneFood;
}
```

La variable `oBreakfast` representa el node `<breakfast>` del document. L'expressió s'executa a partir d'aquest node, per la qual cosa no cal indicar la ruta absoluta.

La funció següent torna tots els resultats de la mateixa expressió anterior en una matriu de nodes:

```
function getAllFoodBreakfast(oDOM){
var allFood = oDOM.documentElement.selectNodes("/menu/breakfast/food");
return allFood;
}
```

2.5.3. Ús d'XPath en el Firefox

El W3C defineix un conjunt d'interfícies per a l'ús d'XPath en DOM. Entre aquestes es troben `XPathEvaluator` i `XPathResult`, accessibles des de JavaScript en navegadors Mozilla. Una instància d'XPathEvaluator permet fer expressions XPath sobre un objecte DOM d'XML mitjançant el mètode `evaluate()`, tot indicant els paràmetres següents:

Paràmetre	Descripció
xpathExpression	Cadena que conté l'expressió XPath que ha de ser avaluada.
contextNode	Node de l'objecte de document a partir del qual s'avaluarà l'expressió, incloent els nodes fill.
namespaceResolver	Funció encarregada de gestionar els espais de noms.
resultType	Constant que especifica el tipus de resultat que ha de ser retornat en avaluar l'expressió. Aquestes constants són dins de la interfície `XPathResult`.
result	Si s'especifica un objecte existent, aquest serà usat per a retornar els resultats. Especificant `null`, es retornarà un objecte nou `XPathResult`.

El resultat de `evaluate()` és sempre un objecte del tipus indicat pel paràmetre `resultType`. Per a especificar aquest paràmetre, es pot utilitzar una constant definida dins de la interfície `XPathResult`. Per exemple, per a les expressions XPath que retornen un tipus simple, es poden utilitzar les constants següents:

Enllaç d'interès

Es poden consultar tots els tipus de resultats d'una expressió XPath mitjançant XPathEvaluator en aquest enllaç: *Document Object Model XPath.*

- `XPathResult.NUMBER_TYPE`: el tipus retornat per l'expressió és numèric.
- `XPathResult.STRING_TYPE`: cadena de caràcters.
- `XPathResult.BOOLEAN_TYPE`: tipus booleà.

Si, contrariament, l'expressió utilitzada retorna un conjunt de nodes, es pot utilitzar la constant següent:

```
XPathResult.UNORDERED_NODE_ITERATOR_TYPE
```

El tipus retornat és un objecte que permet recórrer els nodes del resultat mitjançant el mètode `iterateNext()`. Aquest mètode retorna el node següent del resultat de l'expressió. En cas que no n'hi hagi més, retornarà `null`.

Excepció

Si mentre es recorre el resultat mitjançant `iterateNext()`, es modifica l'estructura de l'arbre de l'objecte DOM, aquest mètode generarà una excepció.

Exemple d'ús d'XPath en el Firefox

La funció `getFoodArray` torna una matriu amb tots els nodes de tipus `<food>` dins de l'arbre:

```
function getFoodArray(oDOM){
    var oEvaluator = new XPathEvaluator();
    var iterator = oEvaluator.evaluate( sXPath,
       oDOM.documentElement,
       null,
       XPathResult.UNORDERED_NODE_ITERATOR_TYPE,
       null);
    var aFood = new Array;
    try{
        var thisNode;
        while(thisNode = iterator.iterateNext())
        {
            aFood.push(thisNode);
        }
        return aFood;
    }
    catch(e){
        alert('Error: The tree was modified during iteration' + e);
    }
}
```

Per a fer la selecció de nodes, es crea un objecte de la classe `XPathEvaluator`. El mètode `evaluate()` torna un objecte que representa el conjunt de nodes de l'expressió en especificar la constant `UNORDERED_NODE_ITERATOR_TYPE`.

Mitjançant el mètode `iterateNext`, es recorre cadascun dels nodes obtinguts per a afegir-los en la matriu. Tot això està inclòs dins d'un bloc `try` per a capturar una excepció en cas que l'arbre del document sigui modificat en el moment en què s'està accedeix als resultats.

2.6. Exemple general

Fins a aquest punt s'ha estudiat com es pot obtenir un document XML, recórrer-ne l'estructura mitjançant el DOM i seleccionar-ne diferents parts amb XPath. L'exemple 13 posa en pràctica cadascun d'aquests aspectes.

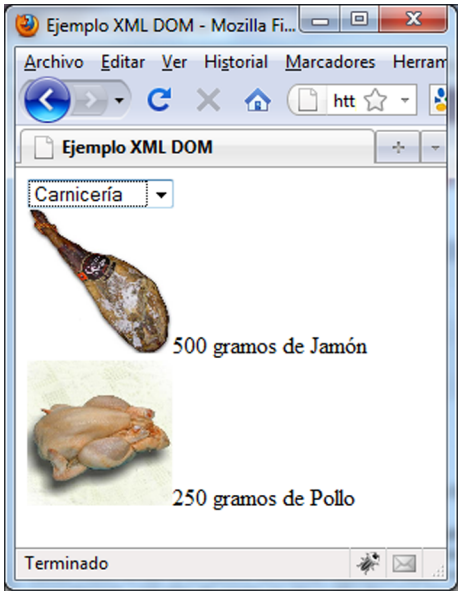
Exemple 13

Aquest exemple consta d'una pàgina en què es mostren els diferents productes del document `lista.xml`. Un combo permet seleccionar una o totes les categories de productes que s'ha de visualitzar. Per a cada producte es mostra la imatge, i un text a partir dels seus atributs mitjançant HTML dinàmic.

Vegeu també

Vegeu el document `lista.xml` en el subapartat 2.3 d'aquest mòdul didàctic.

Figura 12. Captura de l'exemple



L'exemple consta de la pàgina següent:

- **`listaCompra.html`**

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Ejemplo XML DOM</title>
    <script type="text/javascript" src="../script/Ajax.js"></script>
    <script type="text/javascript">
    //S'obté una instància d'XMLHttpRequest
    oXMLHttpRequest = createXMLHttpRequestObject();
    //variable que representarà l'objecte del document lista.xml
    oListaCompra = null;

    //Obté el fitxer lista.xml de forma asíncrona
    function getListaCompra(){
    ...
    }

    //Afegeix al document una divisió amb la llista
    //de productes a partir d'un vector
    function mostrarLista(aProductos){
    ...
    }
```

```
    //Crea una matriu de nodes amb el producte seleccionat del combo
    //i crida a la funció MostrarLista() passant-los per paràmetre
    function cambiarCategoria(){
    ...
    }
    </script>
</head>

<body onload="getListaCompra()">
    <select id="seccionesCompra" disabled="disabled"
    onclick="cambiarCategoria()">
        <option>Toda la lista</option>
        <option>Verdulería</option>
        <option>Carnicería</option>
        <option>Lácteos</option>
    </select>
    <div id="divListaCompra"></div>
</body>
</html>
```

La pàgina consta d'un combo que està inicialment deshabilitat i conté quatre opcions en la llista, tres per a visualitzar els productes de cadascuna de les categories d'aliments i una opció per a mostrar tota la llista. En seleccionar un element del combo es crida la funció `cambiarCategoria()`, que s'encarregarà de visualitzar els productes corresponents dins de la divisió `divListaCompra`.

El motor AJAX disposa de les funcions JavaScript següents:

a) `createXMLHttpRequestObject()`

Crea un objecte XMLHttpRequest tenint en compte el navegador. La funció és dins del fitxer `AJAX.js` inclòs en la pàgina.

b) `getListaCompra()`

```
//Obté el fitxer lista.xml de forma asíncrona
function getListaCompra(){
    oXMLHttpRequest.open('GET', 'lista.xml');
    oXMLHttpRequest.onreadystatechange = function(){
        if(oXMLHttpRequest.readyState == 4){
            if(oXMLHttpRequest.status == 200){
                oListaCompra = oXMLHttpRequest.responseXML;
                if(oListaCompra != null){
                    var oSelect = document.getElementById('seccionesCompra');
                    oSelect.disabled = false;
                    cambiarCategoria();
                }
```

```
            }
            else{
                alert('Error de petición: ' + oXMLHttpRequest.statusText);
            }
        }
    }
    oXMLHttpRequest.send(null);
}
```

Recupera el fitxer `lista.xml` de manera asíncrona. Una vegada obtingut, habilita el *combo box* de la interfície perquè l'usuari pugui seleccionar les diferents categories. Posteriorment, crida al mètode `cambiarCategoria()` per a visualitzar la llista completa de productes. Aquesta funció es descriu a continuació.

c) `mostrarLista(aProductos)`

```
//Afegeix al document una divisió amb la llista de productes a partir d'un vector
function mostrarLista(aProductos){
    //A divLista s'afegirà el detall de cada producte
    var divLista = document.createElement('div');
    /*Es recorren tots els productes passats per paràmetre.
    Per cada un crea una divisió i hi afegeix un conjunt
    d'elements: text, imatge del producte, etc. Finalment
    l'afegeix a divLista*/
    for (var i = 0; i < aProductos.length; i++){
        var sImagen = aProductos[i].getAttribute('imagen');
        var iCantidad = aProductos[i].getAttribute('cantidad');
        var sUnidad = aProductos[i].getAttribute('unidad');
        var sNombre = aProductos[i].firstChild.nodeValue;
        var img = document.createElement('img');
        var div = document.createElement('div');
        img.src = 'images/' + sImagen;
        img.width = 100; img.height = 100;
        div.appendChild(img);
        div.appendChild(document.createTextNode(iCantidad + ' ' + sUnidad +
        ' de ' + sNombre));
        divLista.appendChild(div);
    }
    //Afegeix la divisió amb els productes a la divisió divListaCompra
    var docDiv = document.getElementById('divListaCompra');
    if(docDiv.firstChild != null)
        docDiv.removeChild(docDiv.firstChild);
    docDiv.appendChild(divLista);
}
```

Aquesta funció crea la divisió representada per la variable `divLista` i afegeix text i imatges HTML a partir de la matriu de nodes `<producto>` passada per paràmetre.

El bucle `for` recorre tots els nodes `<producto>` i per cadascun fa el següent:

1) Obté els valors dels atributs `imagen`, `cantidad` i `unidad` del producte en les variables corresponents. També recupera el text de l'etiqueta (*tag*) que descriu el nom del producte mitjançant la propietat `nodeValue`.

2) Crea una divisió en què s'inserirà la imatge del producte i el text.

3) Crea una imatge dinàmicament i hi assigna els atributs `img` amb l'URL i `width` i `height`, que n'estableixen l'amplària i l'alçària. Posteriorment, afegeix la imatge dins de la divisió.

4) Afegeix en la divisió un node de text amb la quantitat, el tipus d'unitat i el nom del producte.

5) Afegeix la divisió del producte dins de la divisió representada per la variable `divLista`.

Després del bucle, la divisió `divLista` contindrà encara una divisió amb una imatge i text per cadascun dels productes de la matriu. Les línies següents després del bucle estableixen la divisió `divLista` en la divisió declarada en el document per a visualitzar-la: si ja s'havia establert una divisió anteriorment, l'elimina.

d) `cambiarCategoria()`

```
function cambiarCategoria(){
    if(oListaCompra == null) getListaCompra();
    var index = document.getElementById('seccionesCompra').selectedIndex;
    var sxPathString;
    switch(index){
        case 0:
            sxPathString = '//producto';
            break;
        case 1:
            sxPathString = '/lista/verduleria//producto';
            break;
        case 2:
            sxPathString = '/lista/carniceria//producto';
            break;
        case 3:
            sxPathString = '/lista/lacteos//producto';
            break;
```

```
    }
    var aProducts = null;
    try{
        var iterator = oListaCompra.evaluate(sxPathString, oListaCompra, null,
        XPathResult.ANY_TYPE, null);
        aProducts = new Array();
        while(thisNode = iterator.iterateNext()){
            aProducts.push(thisNode);
        }
    }catch(error){
        try{
            aProducts = oListaCompra.selectNodes(sxPathString);
        }catch(error){
            alert(''Navegador no compatible amb XPath'');
        }
    }
    mostrarLista(aProducts);
}
```

Aquesta funció crea la matriu de nodes `<producto>`, que es passarà a la funció `mostrarLista()`. Els nodes obtinguts correspondran a la categoria seleccionada en el combo i se seleccionen a partir d'una expressió XPath que simplifica la tasca d'obtenir els nodes corresponents a una categoria de producte.

La primera condició comprova que s'hagi obtingut el DOM del document `lista.xml` i, si no és així, l'obté cridant a la funció `getListaCompra()`. La tasca següent és obtenir l'índex del combo. En un commutador[10], s'avalua l'índex i s'assigna a una variable l'expressió XPath que permetrà obtenir els nodes `<producto>` corresponent a la categoria seleccionada.

[10]En anglès, *switch*.

A continuació, hi ha dos `try...catch` incrustats. Dins d'aquests es processa l'expressió XPath sobre el document `lista.xml`. El primer `try...catch` avalua l'expressió per a navegadors Firefox i d'altres que segueixen l'estàndard definit pel W3C. Després d'executar el mètode `evaluate` sobre el DOM del document XML, s'afegeixen tots els nodes obtinguts sobre la matriu `aProducts`. Si el navegador utilitzat fos l'Internet Explorer, es generaria una excepció i s'executaria la instrucció del segon `try...catch`. L'objecte tornat per la sentència `selectNodes` per a l'Internet Explorer ja retorna una matriu en la qual s'estableix `aProducts`.

Finalment, es crida a la funció `mostrarLista(aProducts)` passant per paràmetre la matriu `aProducts`, que serà representada en la pàgina.

Figura 13. Captura de l'exemple

2.7. Transformacions XSL

XSL és una família de recomanacions de W3C que defineix transformacions i presentacions de documents XML en diferents formats, com l'XHTML. A partir d'un document XML, els dissenyadors utilitzen XSL per a expressar les dades del document en un mitjà de presentació a partir d'un procés de transformació.

Dins d'aquesta família es poden destacar els llenguatges següents:

- XSLT[11]. Aquest llenguatge defineix com s'han de formatar els documents XML en d'altres també basats en XML. Es pot consultar la recomanació per Internet a XSL Transformations (XSLT).

- XSL-FO[12]. Està orientat a la creació de documents de diferents formats no basats en XML. Permet definir altres aspectes de presentació com les característiques d'una pàgina, de paràgrafs, de taules, de vincles, etc.

- XPath: acrònim d'*XML path language*. És un llenguatge que permet l'encaminament i la selecció de diferents parts d'un document.

D'aquesta família es poden destacar l'XSLT i l'XPath, llenguatges que es poden usar conjuntament per a transformar documents XML en XHTML a partir d'una plantilla. L'objectiu d'aquest punt és estudiar com s'ha d'obtenir un document XML, transformar-lo en format XHTML i presentar-lo en una àrea de la interfície del client.

(11) *XSLT* és la sigla d'*extensible stylesheet language transformations*, 'llenguatge de fulls d'estil de transformació'.

(12) *XSL-FO* és la sigla d'*extensible stylesheet language formatting objects*.

Formats no basats en XML

Alguns dels formats de sortida són documents PDF, SVG, RTF, etc.

Per a transformar un document XML caldrà tenir en compte el navegador del client. A continuació, s'explica com s'ha de fer la transformació en navegadors com l'Internet Explorer i el Firefox. La plantilla XSL aplicada per als dos casos és la següent:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" omit-xml-declaration="yes" indent="yes" />
    <xsl:template match="/">
        <div>
        <xsl:for-each select="//producto">
            <div>
                <xsl:variable name="varImagen" select="@imagen" />
                <xsl:variable name="varCantidad" select="@cantidad" />
                <xsl:variable name="varUnidad" select="@unidad"/>
                <xsl:variable name="varProducto" select="."/>
                <img src="./images/{$varImagen}" width="100" height="100" />
                <xsl:value-of select="$varCantidad"/>
                <xsl:value-of select="$varUnidad"/> de
                <xsl:value-of select="$varProducto"/>
            </div>
        </xsl:for-each>
        </div>
    </xsl:template>
</xsl:stylesheet>
```

Aquesta plantilla es pot aplicar sobre el document `lista.xml`. La transformació que fa consta d'un fragment HTML amb la llista de productes. Per cadascun dels productes, mostra la imatge i un text explicatiu. El codi HTML obtingut a partir de la plantilla anterior és el següent:

```
<div>
    <div>
        <img src="./images/patatas.jpg" width="100" height="100">
        2 Kg de patatas
    </div>
    <div>
        <img src="./images/zana.jpg" width="100" height="100">
        1 Kg de zanahorias
    </div>
    <div>
        <img src="./images/huevos.jpg" width="100" height="100">
        1 docena de huevos
    </div>
    <div>
        <img src="./images/jamon.jpg" width="100" height="100">
        500 gramos de jamón
```

```
        </div>
        <div>
            <img src="./images/pollo.jpg" width="100" height="100">
            250 gramos de pollo
        </div>
        <div>
            <img src="./images/yogur.jpg" width="100" height="100">
            6 unidades de yogur
        </div>
        <div>
            <img src="./images/leche.jpg" width="100" height="100">
            2 litros de leche
        </div>
    </div>
```

2.7.1. Transformacions XSL en l'Internet Explorer

El DOM de l'Internet Explorer disposa del mètode `transformNode(oXSL)`. Aquest mètode retorna el document en format de text resultat de transformar el DOM que fa la crida amb la plantilla XSL passada per paràmetre. El mètode es pot cridar des de qualsevol node i el resultat serà la transformació de l'arbre que descendeix a partir d'aquest.

Inicialment caldrà obtenir els DOM dels documents XML i XSL. Per a això es pot utilitzar la propietat `responseXML` d'XMLHttpRequest.

Plantilla de transformació xsl

La plantilla de transformació *lista.xsl* també es pot obtenir mitjançant la propietat responseXML d'XMLHttpRequest, ja que els documents XSL són també XML.

El codi següent obté els objectes de document de *lista.xml* i *lista.xsl* mitjançant la funció `getDocument(sDocURL)`.

```
function getDocument(sDocURL){
    var oXMLHttpRequest = createXMLHttpRequestObject();
    oXMLHttpRequest.open('GET', sDocURL,false);
    oXMLHttpRequest.send(null);
    if(oXMLHttpRequest.readyState == 4){
        if(oXMLHttpRequest.status == 200){
            return oXMLHttpRequest.responseXML;
        }
        else{
            alert('Error de petición: ' + oXMLHttpRequest.statusText);
        }
    }
}
```

```
var oDOMXML = getDocument('lista.xml');
```

```
var oDOMXSL = getDocument('lista.xsl');
```

Una vegada obtinguts els DOM, es pot fer la transformació a partir de l'objecte del document XML amb el mètode `transformNode(oXSL)`, passant per paràmetre el DOM del document XSL:

```
oDOMXML.async = false;
oDOMXSL.async = false;
var sResult = oDOMXML.transformNode(oDOMXSL);
```

La variable `sResult` contindrà el text de les dades transformades. Per a mostrar-les, es pot establir en la propietat `innerHTML` d'una etiqueta `<div>`.

```
var div = document.getElementById('div');
div.innerHTML = sResult;
```

2.7.2. Transformacions XSL en el Firefox

Les transformacions XSL en el Firefox es duen a terme mitjançant la interfície proveïda per la classe `XSLTProcessor`. S'ha de crear una instància i inicialitzar l'objecte amb una plantilla:

```
var oXSLTProcessor = new XSLTProcessor();
oXSLTProcessor.importStylesheet(oDOMXSL);
```

La variable oDOMXSL és un DOM d'un document XSL prèviament carregat.

Per a fer la transformació es pot fer ús d'un dels dos mètodes `XSLTProcessor` que s'enumeren a continuació:

1) `transformToDocument(oXMLDOM)`: requereix un node com a argument. Torna un nou DOM de tipus document amb el resultat de la transformació:

```
var oDocument = XSLTProcessor.transformToDocument(oXMLDOM);
```

2) `transformToFragment(oXMLDOM, ownerDocument)`: torna un fragment de document de tipus DocumentFragment* i requereix dos paràmetres. El primer és el node XML a partir del qual s'aplicarà la transformació. El segon paràmetre és l'objecte de document que tindrà el fragment retornat:

```
var oFrag = oXSLTProcessor.transformToFragment(oListaXML, document);
var div = document.getElementById('division');
div.appendChild(oFrag);
```

Fragments de document

Els fragments de document són porcions lleugeres d'objectes de document. Són útils per a manipular nodes sense haver de processar l'arbre complet. Una vegada processats, es poden afegir al document original.

En el codi anterior s'afegeix el fragment tornat a una part del mateix document que es va passar per paràmetre per fer-ne la transformació.

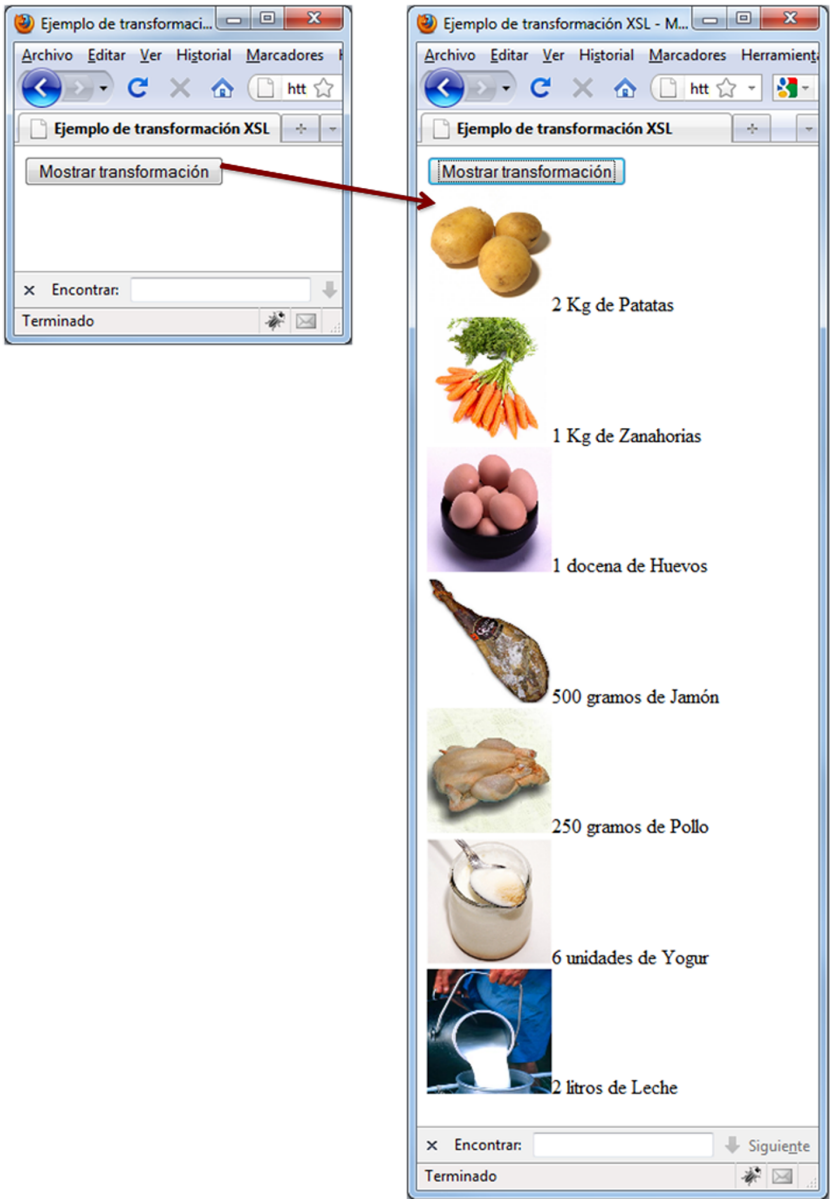
2.7.3. Exemple de transformació XSL per a plataformes creuades

L'exemple 14 és semblant al de l'apartat anterior, en què es mostren productes d'un mateix tipus a partir d'un combo. En aquest cas es mostren tots en fer clic sobre un botó.

Exemple 14

S'utilitza la plantilla `lista.xsl` i fa una transformació obtenint un fragment HTML llest per a ser afegit a qualsevol part del document sense necessitat de crear HTML dinàmic. Es tenen en compte les diferents implementacions dels navegadors Internet Explorer i Mozilla Firefox.

Figura 14. Resultat de l'exemple



El fitxer de l'exemple és el següent:

- ***listaCompraXSL.html***

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Ejemplo de transformación XSL</title>
    <script type="text/javascript" src="../script/Ajax.js"></script>
    <script type="text/javascript">
    oListaXML = null;
    oListaXSL = null;

    //Retorna l'objecte de document a partir de l'URL d'un fitxer XML
    function getDocument(sDocURL){
    ...
```

```
    }

    //Fa la transformació XSL i mostra el resultat a divListaCompra
    function mostrarXSLT(){
    ...
    }

    //Obté els objectes de document de lista.xml i lista.xsl
    function cargaDocumentos() {
    ...
    }

    </script>
</head>

<body onload="cargaDocumentos()">
    <button id="button" disabled="disabled" onclick="mostrarXSLT()">
    Mostrar transformación
    </button>
    <div id="divListaCompra"></div>
</body>
</html>
```

La pàgina conté dins `<body>` un botó i una divisió. El botó, deshabilitat inicialment, cridarà la funció encarregada de fer la transformació XSL i mostrar-la en la divisió. La funció establerta en l'esdeveniment `onload` de l'etiqueta `<body>` carrega els documents *lista.xml* i *lista.xsl*:

```
//Obté els objectes de document de lista.xml i lista.xsl
function cargaDocumentos() {
    oListaXML = getDocument('lista.xml');
    oListaXSL = getDocument('lista.xsl');
    //Si s'han obtingut correctament, s'habilita el botó
    if((oListaXML != null) && (oListaXSL != null)){
        var oButton = document.getElementById('button');
        oButton.disabled = false;
    }
}
```

Les variables globals `oListaXML` i `oListaXSL` contindran un DOM XML a partir dels documents *lista.xml* i *lista.xsl*, respectivament. Per a carregar-los i obtenir-los, s'utilitza la funció `getDocument()` passant com a paràmetre l'URL del document que es vol obtenir. Si s'han obtingut correctament s'habilita el botó, cosa que permet a l'usuari fer clic i mostrar la llista de la compra.

La funció `getDocument` carrega els documents a partir d'un objecte XMLHttpRequest:

```
//Retorna l'objecte de document a partir de l'URL d'un fitxer XML
function getDocument(sDocURL){
    var oXMLHttpRequest = createXMLHttpRequestObject();
    oXMLHttpRequest.open('GET', sDocURL,false);
    oXMLHttpRequest.send(null);
    if(oXMLHttpRequest.readyState == 4){
        if(oXMLHttpRequest.status == 200){
            return oXMLHttpRequest.responseXML;
        }
        else{
            alert('Error de petició: ' + oXMLHttpRequest.statusText);
        }
    }
}
```

La petició es fa de manera síncrona perquè s'ha d'esperar a la resposta per a tornar el DOM XML obtingut.

Una vegada obtinguts els documents i habilitat el botó, l'usuari pot fer clic per llançar la funció `mostarXSLT()`. Aquesta transforma el document `lista.xml` i el visualitza en la divisió:

```
//Fa la transformació XSL i mostra el resultat en divListaCompra
function mostrarXSLT(){
    var div = document.getElementById('divListaCompra');
    try{//Firefox
        var oXSLTProcessor = new XSLTProcessor();
        oXSLTProcessor.importStylesheet(oListaXSL);
        var oFragment = oXSLTProcessor.transformToFragment(oListaXML, document);
        if(div.firstChild != null)
            div.removeChild(div.firstChild);
            div.appendChild(oFragment);
    }catch(error){
        try{//Internet Explorer
            oListaXML.async = false;
            oListaXSL.async = false;
            var sResult = oListaXML.transformNode(oListaXSL);
            div.innerHTML = sResult;
        }catch(error){
            alert(''Navegador no compatible amb transformacions XSL'');
        }
    }
}
```

La transformació es fa en dos blocs `try` incrustats: el primer `try...catch` fa la transformació per a navegadors Firefox i d'altres. En cas que ocorri un error, s'intentarà en el segon `try` per a navegadors Internet Explorer:

1) En el cas de **Firefox**, es crea una instància d'XSLTProcessor i se n'estableix la plantilla amb el mètode `importStylesheet()` passant per paràmetre el DOM de `lista.xsl`. La variable `oFragment` contindrà un fragment d'objecte de document amb el resultat de la transformació del document *lista.xml* després de cridar a `transformToFragment()`. El resultat és un node d'un fragment de document que s'ha d'afegir al node `<div>` de la pàgina. Prèviament, es comprova que la divisió no tingui un node fill que es visualitza; en aquest cas, s'elimina. Finalment, es crida al mètode `appendChild` de la divisió passant per paràmetre el node del fragment obtingut perquè es visualitzi.

2) Per a **Internet Explorer**, el procediment és diferent. Inicialment, s'estableix la propietat `async` de `oListaXML` i `oListaXSL` a `false`. Això evitarà que el mètode `transformNode` bloquegi l'execució fins a fer la transformació; en cas contrari, s'intentaria visualitzar el resultat sense que l'hagi obtingut la transformació.

El mètode `transformNode` retorna el text HTML resultat de la transformació. Una vegada obtingut, s'estableix en la divisió del document la propietat `innerHTML`.

3. Intercanvi de dades amb JSON

JSON[13] és un format de dades lleuger i obert que permet l'intercanvi d'informació estructurada entre diferents plataformes. Es pot usar com una alternativa al format XML quan la mida de les dades que cal transmetre és d'importància cabdal.

(13) *JSON* és la sigla de *JavaScript object notation.*

Enllaç d'interès

Per a més informació, es pot consultar la pàgina json.org.

El model de dades de JSON està basat en la sintaxi de JavaScript per a la notació literal d'objectes, que permet la definició textual de matrius i objectes. L'accés a aquestes dades des de JavaScript es duu a terme sense cap tipus de manipulació, a diferència del DOM d'XML.

3.1. Notació literal d'objectes i matrius en JavaScript

La notació literal d'objectes de JavaScript és un subconjunt del llenguatge que proveeix de mecanismes concisos per a definir objectes a partir de valors literals accessibles per un nom. Es construeixen amb claus, dins de les quals s'especifica un conjunt de parelles de nom i valor separats per comes:

```
var oPersona = {
    "nombre"        : "Albert",
    "edad"          : 25,
    "email"         : albert@dominio.net
};
```

L'exemple anterior crea un objecte en una variable `oPersona`. Els valors de `oPersona` poden ser accedits per les propietats definides pel nom de cada parella nom-valor:

```
alert(oPersona.nombre);
```

També es pot definir una matriu amb notació literal i instanciant un objecte de la classe *Array*. La notació literal d'una matriu s'especifica amb claudàtors, dins dels quals hi ha un conjunt de valors literals separats per comes. Cada valor pot ser d'un tipus diferent: numèric, booleà, cadena de caràcters, un objecte en la seva notació literal, etc.:

```
var aListaLiteral = [5, "cadena de caracteres", false];
```

Els diferents valors continguts en la matriu poden ser accedits per un índex numèric:

```
for(var i = 0; i < aListaLiteral.length; i++){
    alert(aListaLiteral[i]);
}
```

Els objectes i les matrius declarats amb la notació literal es poden combinar per a crear un objecte amb propietats que siguin matrius o una matriu els valors de la qual siguin objectes.

En el cas d'un objecte, qualsevol propietat pot contenir un valor amb una matriu literal:

```
var oPersona = {
    "nombre"   : "Albert",
    "edad"     : 25,
    "email"    : "albert@dominio.net",
    "carnets"  : ["A","B1"]
};
```

La propietat `"carnets"` representa una matriu literal amb dos valors. L'accés al segon valor de carnets es fa de la manera següent:

```
alert(oPersona.carnets[1]);
```

De la mateixa manera, es poden definir matrius amb valors que siguin objectes:

```
var aPersonas = [
    {
        "nombre"         : "Albert",
        "edad"           : 25,
        "email"          : "albert@dominio.net"
    },
    {
        "nombre"         : "Jordi",
        "edad"           : 30,
        "email"          : "jordi@dominio.net"
    }
];
```

La matriu anterior defineix dos objectes amb els seus respectius valors i propietats. L'accés a una propietat d'un objecte es faria indicant l'índex i la propietat:

```
alert(aPersonas[1].email);
```

3.2. Intercanvi d'informació en format JSON

Analitzador sintàctic

Un analitzador sintàctic forma part d'un compilador o intèrpret i permet la transformació d'un text d'entrada en objectes i dades del mateix llenguatge.

Els sistemes que intercanvien dades en JSON han de disposar d'un analitzador sintàctic que permeti codificar els objectes i les matrius representats textualment en objectes i matrius de l'entorn i a l'inrevés.

JavaScript disposa de la funció `eval()` per a analitzar i executar codi JavaScript a partir d'una cadena de caràcters. Si passem una cadena de caràcters que conté un objecte o una matriu en notació literal, `eval()` retornarà una variable que permetrà l'accés a les dades.

L'exemple següent mostra com s'ha d'obtenir un objecte JSON del servidor i crear un objecte JavaScript per a accedir a les seves propietats:

- ***persona.txt***

```
{
   "nombre"    : "Albert",
   "edad"      : 25,
   "email"     : "albert@dominio.net"
}
```

- ***eval.html***

```
...
var oXMLHttpRequest = createXMLHttpRequestObject();
oXMLHttpRequest.open('GET','persona.txt',false);
oXMLHttpRequest.send(null);
var sCadena = oXMLHttpRequest.responseText;
var oPersona = eval('(' + sCadena + ')');
alert(oPersona.nombre);
...
```

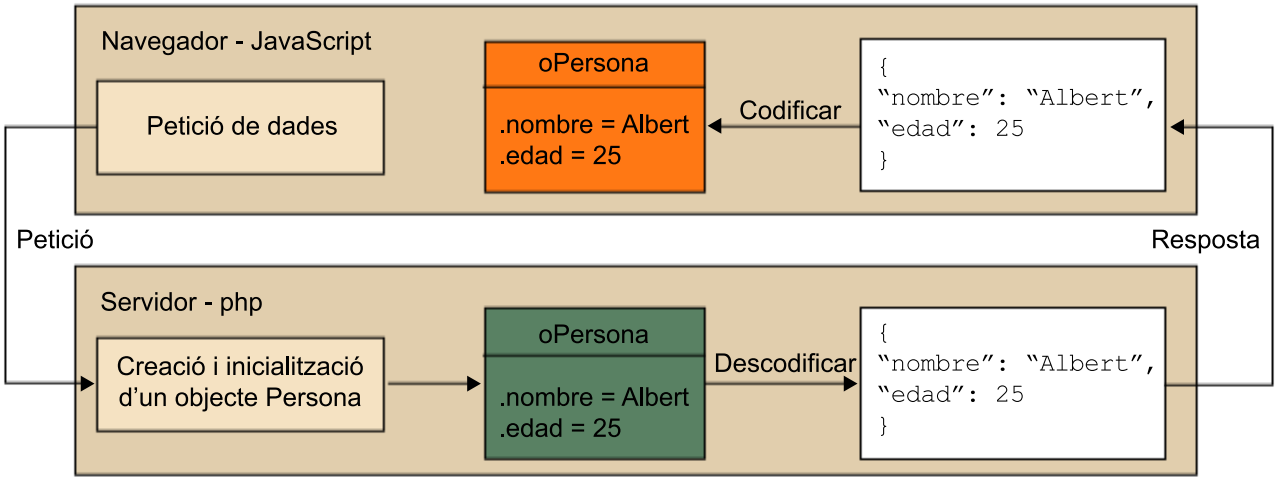
L'objecte XMLHttpRequest de l'exemple recupera el text del fitxer *persona.txt* i s'assigna a la variable `sCadena`. La funció `eval()` transforma el text passat per paràmetre en un objecte. JavaScript no disposa de cap funció per a transformar objectes en la seva notació literal, per la qual cosa serà necessari l'ús de biblioteques externes.

Per raons de seguretat, no es recomana l'ús de la funció `eval()`, sinó un analitzador diferent que ofereixi més seguretat. El motiu és que `eval()` pot avaluar qualsevol codi JavaScript, incloses les funcions. Si en lloc de dades en format JSON s'obtingués codi maliciós, aquest es podria executar i posar en perill el funcionament correcte de l'aplicació.

JSON en el costat del servidor

En l'exemple anterior, s'ha obtingut un objecte procedent d'un fitxer que contenia la informació textual. Però en la majoria de casos interessa tractar les dades des del servidor i convertir-les en objectes JSON de manera dinàmica, tal com mostra la figura 15.

Figura 15. Codificació i descodificació de dades JSON



La figura 15 representa una petició des d'un navegador a un servidor. El servidor genera les dades en objectes propis PHP que posteriorment descodifica per a transformar-los en format JSON. El navegador rep les dades i les codifica per a tractar-les posteriorment.

La majoria d'entorns de treball del costat del servidor disposen d'eines per a descodificar objectes a aquest format.

3.3. Avantatges i desavantatges de JSON sobre XML

Les dades representades en format JSON són més accessibles en JavaScript, ja que utilitzen un subconjunt del llenguatge. La manipulació d'un document XML pot resultar més complexa. A més, les dades en format JSON són més compactes. Per veure'n la diferència, s'estudiarà un exemple amb dos fitxers que contenen la mateixa informació però representada en tots dos formats:

```
<application>
    <users>
        <user>
            <name>Joan Perelló</name>
            <id>34532</id>
            <email>jperello@domain.net</email>
        </user>
        <user>
            <name>Silvia Pons</name>
            <id>12598</id>
            <email>silpons@domain.net</email>
        </user>
        </users>
    <administrators>
        <admin>
            <name>Andrea Álvarez</name>
            <id>43229</id>
```

```
            <email>aalvarez@domain.net</email>
        </admin>
        <admin>
            <name>Enrique Fernandez</name>
            <id>41321</id>
            <email>efernandez@domain.net</email>
        </admin>
    </administrators>
</application>
```

El document anterior representa una llista d'usuaris i administradors que tenen accés a una aplicació. Les etiquetes `<application>`, `<users>`, `<administrators>`, `<user>`, `<admin>`, `<name>`, `<id>` i `<email>` es dupliquen en cada aparició per a identificar un sol element. El document té un total de 430 caràcters, si s'eliminen els espais.

D'altra banda, el fitxer que conté les dades en format JSON és el següent:

```
{
    "application" :
        {
            "users" : [
                {
                    "name"    : "Joan Perelló",
                    "id"      : 34532,
                    "email"   : "jperello@domain.net"
                },
                {
                    "name"    : "Silvia Pons",
                    "id"      : 12598,
                    "email"   : "silpons@domain.net"
                }
            ],
            "administrators" : [
                {
                    "name"    : "Andrea Álvarez",
                    "id"      : 43229,
                    "email"   : "aalvarez@domain.net"
                },
                {
                    "name"    : "Enrique Fernandez",
                    "id"      : 41321,
                    "email"   : "efernandez@domain.net"
                }
            ]
        }
```

```
}
```

Aquest format, encara que pugui semblar més extens pel nombre de línies, conté menys caràcters per a representar la mateixa informació: 317 caràcters. Això significa que les mateixes dades en format JSON ocupen 113 caràcters menys.

Malgrat la seva mida reduïda, JSON pot resultar més intel·ligible per als humans que el format XML. Això pot no ser un problema si les dades intercanviades no requereixen la interpretació humana.

Encara que puguem trobar eines JSON en la majoria d'entorns, XML pot presumir de més suport tant en el costat client com en el de servidor.