UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI SCIENZE STATISTICHE

Corso di Laurea in Statistica e Tecnologie Informatiche

Ajax: specifiche sull'oggetto XMLHttpRequest

Ajax: XMLHttpRequest object specifics

Relatore:

Prof. Graziano Deambrosis

Laureando:

Davide Marchet

matricola: 537259

ANNO ACCADEMICO
2008-2009

INDICE

INTRODUZIONE

Questa tesi nasce dalla mia passione per il web e nello specifico dalla mia passione per la creazione di siti web.

L'analisi che seguirà è rivolta verso una tecnologia che unisce i fondamenti della rete che sono i metalinguaggi HTML e XML, il linguaggio di programmazione Javascript e linguaggi lato server.

Questa unione ha dato alla luce il concetto sulla quale si basa Ajax ovvero la fruizione di contenuti nella rete in modo asincrono fino alla quasi emulazione di una applicazione desktop.

Il concetto che si concretizza nell'ambiente in cui nasce e nella quale noi siamo partecipi che è il Web 2.0.

Web 2.0 significa oltre che un passaggio tecnologico anche una nuova filosofia nel concepire la rete dove è l'utente che la crea, passando così dal ruolo passivo di ricercatore a quello attivo di promulgatore di informazione.

Nella relazione i concetti che verranno espressi riguardano il cuore della tecnologia Ajax che è l'oggetto XMLHttpRequest.

Dopo una breve introduzione storica verrà illustrata la sua creazione e verranno descritti i suoi metodi e parametri che saranno integrati nell'esempio dimostrativo.

Per finire illustrerò alcune problematiche relative ad Ajax e trarrò le conclusioni personali riguardo il suo utilizzo.

CAPITOLO I

L'OGGETTO XMLHttpRequest

1.1 CENNI STORICI

L'acronimo **Ajax** che significa esattamente *Asynchronous JavaScript And XML* è stato enunciato per la prima volta da Jesse Garrett, nel 18 Febbraio 2005, come titolo di un post all'interno del suo blog.

Non si tratta in senso stretto di una nuova tecnologia né di un'invenzione bensì di **un concetto** utilizzato per sviluppare applicativi avanzati e particolari quali ad esempio Gmail, Google Maps o Google Suggest.

Il concetto è in parte espresso nell'acronimo scelto: un **utilizzo asincrono di Javascript** che attraverso l'interfacciamento con XML (o altri linguaggi server side, nella relazione userò PHP) permette ad un client di richiamare informazioni lato server in modo veloce e trasparente.

Queste applicazioni fino a poco tempo fa erano legate principalmente alle tecnologie *Adobe-Macromedia Flash* o *Java* (*Applet*), entrambe purtroppo non sempre interpretabili dai client degli utenti.

In alternativa a queste tecniche di interazione client-server nel 1996 venne introdotto l'*iframe* di Internet Explorer 3 che permetteva di aggiornare alcune porzioni della pagina emulando quello che oggi è una chiamata asincrona.

Nel 1998 Microsoft cominciò a sviluppare una tecnologia, chiamata *Remote Scripting*, con lo scopo di creare una tecnica più elegante per richiamare contenuti differenti ed è in questo periodo che Ajax venne utilizzato per la

prima volta, per poi evolversi in versioni più mature fino alla creazione dell'oggetto **XMLHttpRequest**.

Simmetricamente alla nascita di Ajax, anche il web stesso ha avuto una grande rivoluzione nel passaggio dal così detto *Web 1.0* all'attuale *Web 2.0*, un nuovo modo di intendere la rete, che pone al centro i **contenuti**, le **informazioni** e l'**interazione**.

Il concetto di Web 2.0 enfatizza le capacità di condivisione dei dati tra le diverse piattaforme tecnologiche sia hardware che software e Ajax, che è una **tecnica multi-piattaforma** utilizzabile su molti sistemi operativi, architetture informatiche e browser web, ha permesso la nascita di vere e proprie applicazioni con funzionalità quasi simili a quelle desktop.

Nasce così il concetto di *wiki*, di *blog* e *social network* dove il **flusso di informazione è totalmente gestito dagli utenti** sia nell'inserimento, nella modifica, nell'utilizzo e condivisione.

Simbolo dell'evoluzione tecnologica e culturale del web è il popolare social network *Facebook*, interamente basato su Ajax, concretizzazione tanto amata quanto criticata della nuova filosofia web, focalizzata sulla fruizione dell'infomazione e sulla sua accessibilità tanto che nel nostro particolare periodo storico è diventato testimone di molte realtà nascoste ai tradizionali mezzi di comunicazione.

1.2 INTRODUZIONE A XMLHttpRequest

Il fondamento che sta alla base della tecnologia Ajax è l'oggetto JavaScript **XMLHttpRequest** il quale definisce un *API* che assegna funzionalità al client per il trasferimento di dati tra client e server.

La caratteristica principale di questo oggetto è di essere in grado di effettuare delle **richieste asincrone** ad un server HTTP, permettendo all'utente di svolgere diversi compiti simultaneamente all'interno di una stessa pagina web senza dover attendere la risposta dal server e i tempi di *refresh*.

Infatti la sequenza classica di una richiesta HTTP consiste nell'invio al server delle informazioni, la loro elaborazione e ricezione tramite un ricaricamento della pagina.

In sostanza con Ajax e quindi tramite l'oggetto XMLHttpRequest (XHR successivamente) si evita tutto questo processo lavorando in back end.

XHR era stato originariamente implementato su Internet Explorer 5 come componente *ActiveX*.

La limitata estensione di ActiveX che non permette l'utilizzo di tutti i metodi di XHR e il fatto che non essendo un oggetto nativo implica una diversa implementazione, hanno evidenziato alcune restrizioni nel suo utilizzo.

Successivamente è stato adottato come standard sui browser Mozzilla e Safari rispettivamente dalle versioni 1.0 e 1.2.

Oggigiorno la maggior parte dei browser quali Internet Explorer dalla versione 7, Mozzilla Firefox, Safari, Opera e Nescape supportano nativamente

l'oggetto XHR, nonostante non sia comunque uno "standard de facto" riconosciuto dal consorzio W3C.

La mancanza di uno standard può influenzare il comportamento *cross-browser* dell'oggetto in questione anche se la maggior parte dei metodi e delle proprietà sono supportate .

1.3 RICHIESTE ASINCRONE

La differenza tra una richiesta **sincrona** e **asincrona** sta nel fatto che nel primo caso l'esecuzione del codice viene interrotta bloccando il browser temporaneamente fino al completamento della richiesta mentre nel caso asincrono non occorre attendere che la richiesta sia ultimata per effettuare altre operazioni, stravolgendo sotto diversi punti di vista il flusso tipico di una applicazione web.

Generalmente infatti il flusso è racchiuso in due passaggi: richiesta dell'utente (*link*, *form* o *refresh*) e risposta da parte del server.

A questo si aggiunge un terzo ed inevitabile incomodo di questo ciclo che è l'attesa che intercorre tra una richiesta dell'utente e la risposta del server.

Con l'aggiunta di Ajax questa linearità viene a meno e quindi mentre l'utente è all'interno della stessa pagina le richieste al server possono essere **numerose**, completamente **indipendenti** e apportate tramite l'utilizzo di JavaScript.

I tempi di attesa praticamente inesistenti di una richiesta asincrona sono dovuti al fatto che i dati restituiti dal server sono inviati attraverso degli array contenuti nei metodi di XHR, che successivamente verranno utilizzati dall'applicazione Javascript del client.

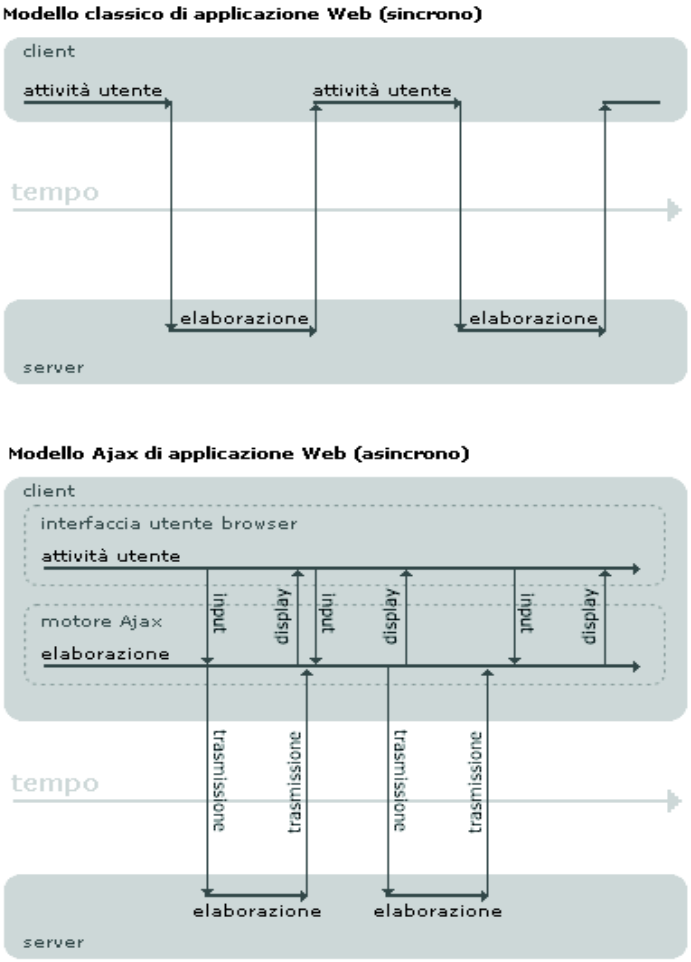


Figura 1 Schema di una chiamata Sincrona e Asincrona

1.4 COSTRUTTORE DI XMLHttpRequest

Il nome **XMLHttpRequest** indica che quest'oggetto Javascript si occupa di realizzare delle richieste compiute dal browser utilizzando il protocollo standard HTTP restituendo l'informazione della risposta in formato **text** o **XML**.

I tre punti fondamentali per il suo utilizzo sono:

1. **instanziare l'oggetto;**
2. **associare la funzione da invocare al termine della richiesta;**
3. **effettuare la richiesta.**

In Javascript un oggetto viene istanziato attraverso l'operatore **`new`** seguito dal costruttore dell'oggetto, dove con costruttore si indica il nome della funzione che lo definisce.

Un esempio di istanza è il seguente:

```
var xmlHttp = new XMLHttpRequest();
```

Questa riga di codice solamente però non ci permette di creare l'oggetto dato che XHR come già detto non è uno standard W3C e il suo funzionamento *cross-browser* è incerto.

Fino alla versione 6 di Internet Explorer l'oggetto è supportato come ActiveX mentre già dalla 7 come oggetto nativo. Gli altri browser lo supportano anche nelle versioni più obsolete.

Alcuni problemi in via teorica si hanno se l'utente naviga con la versione di Internet Explorer 4 (browser deprecato) nella quale lo stesso oggetto ActiveX non è supportato, impedendo l'utilizzo di Ajax.

Il nome del browser su cui navighiamo è ricavabile con il metodo **`navigator.user.Agent`**.

Questa eterogeneità tra browser deve essere considerata con dei controlli all'interno del codice per instanziare l'oggetto .

Di seguito il codice:

```
var xmlHttp;
function createXMLHttpRequest()
{ browserUtente = navigator.userAgent.toUpperCase();
  if(typeof(XMLHttpRequest)        ==        "function"        ||
typeof(XMLHttpRequest) == "object")
      xmlHttp=new XMLHttpRequest();
  else  if(window.ActiveXObject  &&  browserUtente.indexOf("MSIE
4") < 0) {
      if(browserUtente.indexOf("MSIE 5") < 0)
            xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
      else  XHR = new ActiveXObject("Microsoft.XMLHTTP");
      }
}
```

Sezione 1 Codice di instanzia XHR

Come si può vedere la creazione dell'oggetto non è molto complessa.

Il codice contiene una logica *branching* semplice che determina come instanziare XMLHttpRequest a seconda del browser sottostante.

Innanzitutto si crea una variabile di ambito globale chiamata **`xmlHttp`** in cui è contenuto il riferimento all'oggetto.

All'interno della funzione **`assegnaXMLHttpRequest`** che creerà un'istanza di XMLHttprequest, troviamo una variabile locale **`browserUtente`** a cui viene assegnato il valore restituito dalla funzione **`navigator.userAgent.toUpperCase()`** che conterrà la stringa identificativa del browser.

Successivamente vengono effettuati i controlli per creare l'oggetto.

Il primo riguarda i browser che supportano Javascript nativamente e avviene attraverso un'operazione logica che valuta se il browser supporta XMLHttpRequest come funzione o oggetto.

Nel caso il controllo abbia esito positivo viene istanziato l'oggetto **`xmlHttp =new XMLHttprequest()`** e in caso contrario si passa ai controlli successivi.

La condizione per verificare se è un oggetto ActiveX avviene con il controllo **`window.ActiveXObject`** che restituirà un valore positivo in caso affermativo o un valore valore **`null`** in caso contrario.

Un controllo avviene anche sul nome del browser che nel caso fosse “MSIE4” verrebbe filtrato.

L'istanza dell'oggetto su Internet Explorer 6 avviene con il codice **`xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");`** mentre per le versioni 5 e 5.5 è **`xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");.`**

Un'alternativa al controllo sulle stringhe nominative dei browser sarebbe stata quella di utilizzare controlli basati sulle eccezioni attraverso i blocchi **`try catch`** piuttosto che le condizioni **`if`**.

Il codice suggerito risulterebbe più snello e robusto a valori anomali:

```
If(window.ActiveObjext){
    try{xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");}
    catch(e){
        try{xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");}
        catch(e){}
    }
}
```

Sezione 2 Controllo browser con eccezzioni

1.5 METODI DELLA CHIAMATA DI XMLHttpRequest

Definita la creazione dell'oggetto è indispensabile conoscere i metodi che ci permettono di sfruttarlo al meglio a seconda delle necessità.

La lista dei metodi è diversa da browser a browser e per questo motivo vengono usati quelli che fanno capo al browser MAC Safari, il quale supporta XHR nativamente con meno metodi ma comuni a tutti gli altri browser.

La lista di questi metodi viene presentata rispetto il loro ordine di utilizzo:

1. **`open`**
2. **`send / setRequestHeader`**

3. **abort**
4. **setResponseHeader / getAllResponseHeaders**

Metodo `open`

Il primo metodo a essere utilizzato per inizializzare una chiamata asincrona è il metodo **`open`**. E' definito dal W3C per accettare cinque parametri.

La definizione del metodo è la seguente:

```
open(method, url [,async][,user][,password]);
```

Parametro `method`

Il primo parametro è una stringa che indica il metodo di invio dati.

Possono essergli assegnati due valori GET e POST.

La differenza tra queste due assegnazioni è la stessa che intercorre nell'invio dati in un qualunque form HTML.

Scegliendo GET, che in genere viene usato quando più richieste restituiscono lo stesso risultato e non modificano lo stato del server, le variabili verranno trasferite direttamente nell'*url* della pagina come in questo esempio: **`pagina.html?variabile=valore`**.

Il limite principale di una chiamata GET sta nel numero di caratteri che è possibile inviare e quindi nella limitatezza delle richieste. Possono essere trasferiti un massimo di 256 caratteri ma è una cifra soggettiva ai vari browser.

Riferendoci ad Ajax, inviare dati sotto forma di link risulta il modo più comodo e semplice.

Il metodo POST invece invia le informazioni in modo invisibile all'utente senza le restrizioni del metodo GET e viene genericamente usato quando si modifica lo stato del server (es. un'iscrizione ad un form). Il limite della lunghezza della richiesta può essere impostato dal server e di solito pari ad 8 Mb.

Parametro `url`

Il secondo parametro è il nome della pagina da leggere e quindi da dove verranno estrapolati i dati della risposta.

Il collegamento è rappresentato anch'esso da una stringa e può essere espresso come percorso *assoluto* o *relativo*.

Alcune limitazioni inducono il metodo `open` a segnalare un'eccezione se l'*url* della richiesta proviene da un dominio differente rispetto quello della pagina corrente.

Parametro `async`

Il terzo parametro, `async`, è un valore booleano che deve essere impostato a **`true`** per indicare al metodo `open` che la richiesta da effettuare è di tipo asincrono oppure a **`false`** per una normale richiesta sincrona. Impostarlo a *`false`* comunque risulterebbe molto limitativo dato che la possibilità di richieste asincrone è uno dei vantaggi principali dell'utilizzo di Ajax.

Parametri `user password`

I parametri **`user`** e **`password`** servono per l'autenticazione dell'HTTP e possono essere passati facoltativamente.

Metodo `send`

Il secondo metodo da usare è il metodo **`send`** il quale esegue l'effettiva richiesta al server.

La sintassi metodo è:

```
send(data);
```

L'argomento `data` può essere un'istanza di un oggetto *DOM*, un flusso di dati di input o una stringa.

La sua specificazione nel codice varia a seconda del tipo della richiesta da effettuare, GET o POST per l'appunto.

Per una chiamata di tipo GET il parametro inviato in `send` è **`null`** in quanto le informazioni vengono racchiuse nell'*url* del metodo `open`.

```
xmlHttp.open("get","pagina.ext?variabile=valore",true);
xmlHttp.send(null);
```

Sezione 3 Esempio chiamata GET

Nel caso di invio dati con POST, i parametri della richiesta vengono inviati attraverso `send` con la differenza di dover creare degli *headers* cioè una serie di *coppie chiave-valore* specifici per uno scambio dati via HTTP.

Per fare ciò è necessario impostare il metodo **`setRequestHeader`**.

```
xmlHttp.open("post","pagina.ext",true);
xmlHttp.setRequestHeader("content-type","application/x-www-form-urlencoded");
```

Sezione 4 Esempio chiamata POST

Per chiudere la connessione con il server è consigliato impostare un altro *header* dal nome connection e valore `close`.

```
xmlHttp.setRequestHeader("connection","close");
```

Per inviare più variabili è necessario separarle con una **"&"** sia col metodo GET che POST, con l'accortezza di non usare il carattere speciale all'interno del nome o valore della variabile perché il server interpreterebbe il carattere come un separatore.

Per prevenire questo problema si usa la funzione **`escape()`**, esterna al contesto di Ajax, la quale trasforma il contenuto delle variabili in modo da ricevere le giuste coppie *chiave-valore*.

Esempio:

```
variabile = "primo & secondo";
2variabile="valore";
```

questa scrittura, nel passaggio dei parametri attraverso GET per esempio, porterebbe a questo risultato: **`link=pagina.ext?variabile=`**`primo` **`& secondo & 2variabile=`**`valore`.

Invece che due variabili ce ne sono tre, tra le quali "**`secondo`**" non ha nessun valore.

Utilizzando **`escape()`** questo problema viene risolto e la sintassi di utilizzo è la seguente:

```
link=pagina.ext?escape(variabile);
link +="&";
link+= 2variabile;
```

Metodo `abort`

Questo metodo, che non necessita di parametri, viene utilizzato quando abbiamo la necessità di interrompere bruscamente le operazioni di invio o ricezione.

Di solito viene usato dopo **`send()`** e la sua sintassi, priva di parametri è:

```
abort();
```

Metodi `getResponseHeader getAllResponseHeaders`

`getResponseHeader` e **`getAllResponseHeaders`** ci forniscono informazioni (gli *headers HTTP*) relative alle risposte del server.

La differenza tra i due metodi sta nel fatto che il primo restituisce un valore preciso e necessita del passaggio di un parametro, mentre il secondo non necessita di parametri e fornisce la lista completa delle informazioni.

```
xmlHttp.getResponseHeader(Content-Length);

Content-Length: 195

xmlHttp.getAllResponseHeaders();

Content-Length: 195 Content-Type: text/xml

ETag: "b34ba3751251c81:4cd" X-Powered-By: ASP.NET

MicrosoftOfficeWebServer: 5.0_Pub

Last-Modified: Mon, 07 Jan 2008 09:48:30 GMT
```

Sezione 5 esempio di visualizzazione degli header

1.6 PARAMETRI DELLA RISPOSTA

Dopo la definizione dei metodi con i quali viene effettuata una chiamata asincrona al server, ora passiamo a quella dei parametri con cui viene costruita una risposta.

La lista è la seguente:

1. **onreadystatechange**
2. **readyState**
3. **status / Status Text**
4. **responseXML / responseText**

Parametro `Onreadystatechange`

Dopo una richiesta al server ci serve una funzione che possa ricevere i dati di ritorno dalla chiamata.

Il parametro **`onreadystatechange`** memorizza la funzione che elaborerà i dati ricevuti dal server.

Questo non è un metodo perché la funzione memorizzata verrà richiamata automaticamente dall'oggetto XMLHttpRequest.

Il codice di chiamata della proprietà è il seguente:

```
xmlHttp.onreadystatechange=function(){ codice della funzione }
```

Parametro `readyState`

Il parametro **`readyState`** detiene l'esito della risposta del server, la quale al variare di esso, sarà eseguita la proprietà **`onreadystatechange`**.

I valori del parametro sono i seguenti:

- **`0 Uninitialized`** l'oggetto XMLHttpRequest esiste, ma non è stato richiamato alcun metodo per inizializzare una comunicazione;
- **`1 Open`** è stato richiamato il metodo `open()` ed il metodo `send()` non ha ancora effettuato l'invio dati;
- **`2 Sent`** il metodo `send()` è stato eseguito ed ha effettuato la richiesta;
- **`3 Receiving`** i dati in risposta cominciano ad essere letti. Quando assume il valore 3, per tutti i browser tranne le versioni precedenti alla 7 di Internet Explorer, è possibile già sapere il valore di alcuni header della risposta come la lunghezza del testo, il tipo restituito (txt o XML) e quanto testo è stato ricevuto in quel momento grazie alla lunghezza della stringa di `responseText`. Queste informazioni possono risultare utili anche per creare una sorta di *preload* comunicando all'eventuale utente a che punto è la ricezione dati;

- **4 Loaded** l'operazione è stata completata.

Questi stati di `readyState` non sono usufruibili allo stesso modo su tutti i browser e l'unico supportato comunemente è `Loaded`.

La sintassi è la seguente:

```
xmlHttp.onreadystatechange=function(){
      If(xmlHttp.readyState == 4){ codice della risposta  }
}
```

Parametro `Status StatusText`

Il parametro **`status`** contiene esattamente il codice HTTP di risposta restituito dal server.

I valori sono identificati con in codici di stato HTTP quindi **`200`** se l'interazione ha avuto successo, **`404`** se il file non è stato trovato oppure può assumere uno dei valori definiti nella *RFC 261600*.

Il parametro **`statusText`** invece, contiene il responso testuale di status come ad esempio "`OK`" per `200` oppure "`NOT FOUND`" per `404`.

Questo parametro è supportato da quasi tutti i browser ma per avere la certezza che il testo corrisponda esattamente al codice di `status` sarebbe bene creare un apposito oggetto che contenga un insieme di coppie chiave valore che restituiscano il responso numerico e testuale.

Noi per comodità valuteremo subito se lo stato corrisponde al codice `200` dato che se assume un valore diverso non potremmo creare la riposta.

```
xmlHttp.onreadystatechange=function(){
      If(xmlHttp.readyState == 4){
            if(xmlHttp.status == 200){ codice della risposta }
      }
}
```

Parametro `responseXML responseText`

I parametri **`responseXML`** e **`responseText`** saranno i dati restituiti dal server a operazione ultimata.

La differenza principale tra i due sta nel fatto che `responseText` è, ad interazione ultimata, un dato di tipo stringa mentre `responseXML` potrebbe essere un oggetto con valore **`null`** qualora i dati restituiti non dovessero essere un documento XML e quindi un oggetto utilizzabile come tale.

Il vantaggio di `responseText` è quindi quello di poter ricevere comunque informazioni dal server le quali, se non dovessero essere reperibili, verrebbero inviate sottoforma di comunicazione di servizio che informerà l'utente dell'assenza del documento.

Inoltre un vantaggio di `responseText` è anche quello di trasferire documenti XML con la sua rappresentazione testuale, semplificando le operazioni di *parsing* necessarie per l'XML.

```
Function handleStateChange() {

     if(xmlHttp.readyState == 4){

        if(xmlHttp.status == 200){

           document.getElementById("idname").innerHTML=

           xmlHttp.responseText;

     }

}
```

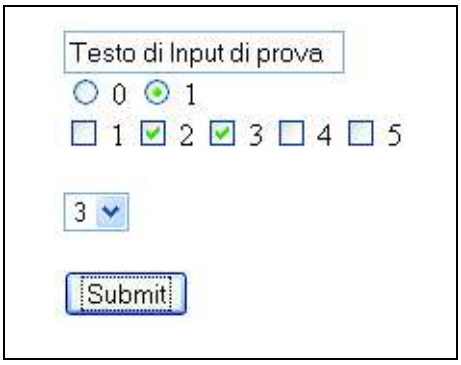
Sezione 6 Codice della risposta asincrona

I dati restituiti dal server attraverso questi parametri sono contenuti in un array.

La forma con cui vengono restituiti è illustrata nel seguente esempio.

Il form sottostante invia i valori a una semplice funzione PHP che ritorna al client le nostre scelte e le visualizza tramite il parametro `responseText`.

La chiamata è attivata attraverso l'evento `onlick` applicato al tasto di input *Submit*.



L'array di ritorno è il seguente:

```
Array ( [field] => Testo di Input di prova [radio] => 1 [check1]
```

```
=> [check2] => 2 [check3] => 3 [check4] => [mycheck5] => [select]
=> 3 )
```

CAPITOLO II

APPLICAZIONE BASATA SU AJAX

2.1 INTRODUZIONE ALL'APPLICAZIONE

Con questo esempio verranno applicati tutti i concetti che sono stati spiegati precedentemente.

L'applicazione illustra dei questionari a due variabili (caricati in modo dinamico) e la visualizzazione dei risultati in modo asincrono.

I dati saranno elaborati da una funzione php che restituirà i valori dell'elaborazione aggiornando i grafici relativi al questionario selezionato.

I diagrammi sono stati ottenuti attraverso il framework Javascript JScharts che è opensource.

L'obiettivo è quello di creare una applicazione dinamica sfruttando il trasferimento asincrono.

L'applicazione è visibile all'indirizzo:

http://proveworksdido.altervista.org/Tesi.

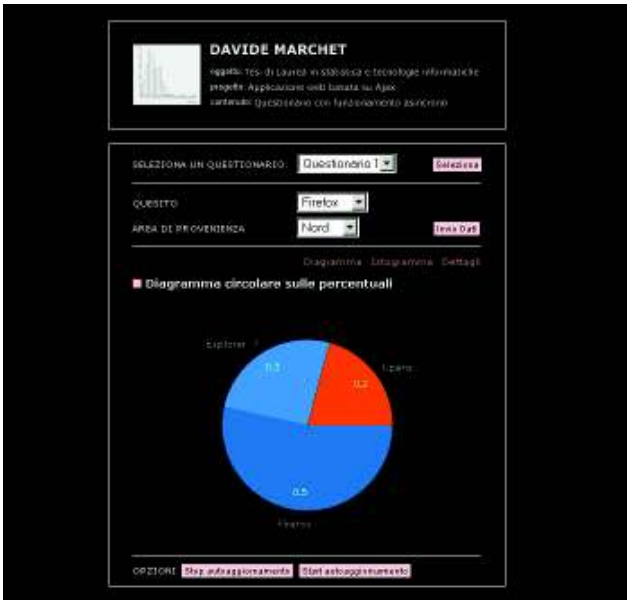


Figura 2 Interfaccia dell'esempio

2.2 EVENTI E PASSAGGIO DEI PARAMETRI

Prima di analizzare il codice della chiamata verranno studiati gli eventi e le funzioni che rispettivamente attivano la chiamata e passano i parametri.
Gli eventi sono posizionati all'interno del tag `<body>`, nel form dei quesiti e nei pulsanti di aggiornamento.
All'interno di `<body>` c'è l'evento `onload` che al caricamento della pagina richiama la funzione **`Startup().`**

```
<body onload="Startup()">
```

`Startup()` è una funzione ausiliaria che ho aggiunto e al suo interno contiene due invocazioni alla funzione **`startRequest()`** che effettuerà la chiamata asincrona.

```
function Startup() {
    startRequest(' ');
    timer = setInterval(function(){startRequest(' ')}, 5000 );
}
```

Le chiamate all'interno di `Startup` sono **`startRequest('')`** e **`timer = setInterval(function(){ startRequest('') }, 5000 )`**. Quest'ultima assegnata alla variabile **`timer`**, effettuerà la chiamata in background ogni 5 secondi.

C’è da notare che nelle funzioni non viene passato nessun parametro. Questo perché in questa fase di caricamento della pagina non abbiamo bisogno di dare un input ma piuttosto di vedere lo stato attuale dell’applicazione.
Lo stesso ragionamento è applicato anche ai pulsanti di aggiornamento.
Il passaggio dei parametri invece, avviene tramite il form *quesiti* opportunamente modificato per il passaggio dei parametri in modo asincrono.
La logica è identica a quella di un form di una qualsiasi applicazione web, solo che il passaggio deve avvenire senza un ricaricamento della pagina che è implicato dall’utilizzo dei metodi GET e POST selezionabili con l’attributo **`method`** del tag `form`.
Occorre quindi modificare la struttura del form in base alla nostra necessità.
Di seguito il codice che verrà analizzato:

```
  <form
action="javascript:get(document.getElementById('quesiti'));"
name="quesiti" id="quesiti" >
    <li class="title">Quale browser utilizzi?</li>
    <li>
    <select name="domanda_A">
    <?php
    foreach($xml->elementi as $elementi)
    {     ?>
    <option value="<?php echo ''.$elementi->id.'';?>"> <?php
echo ''.$elementi->nome.'';?></option>
```

```
    <?php      }
    ?> </select>
    </li>
    <li class="title">Provenienza in Italia</li>
    <li><select name="area">
    <option value="nord">Nord</option>
    <option value="centro">Centro</option>
    <option value="sud">Sud</option>
    </select></li>
    <br />
    <input type="hidden" name="questionario" id="questionario"
value="<?php echo($questionario); ?>" />
    <input type="submit" value="Invio Dati"/>
    </form>
```

La parte evidenziata fa riferimento alla modifica apportata all'invio dei parametri.

Come già detto l'utilizzo dei metodi GET e POST standard implicherebbero il ricaricamento della pagina. La logica adottata affinché questo non avvenga è stata quella di avvalersi della funzione GET di Javascript che applicata nell'attributo **`action`** con parametro **`document.getElementById('quesiti'),`** invia a una funzione ausiliaria di recupero dei parametri (**`get(obj)`**) l'array dei valori selezionati nel form. Questo senza ricaricare la pagina perché è l'applicazione Javascript che gestisce il trasferimento.

La funzione `get(obj)` riceve quindi come parametro **`obj`** che è un vettore contenente i valori del form identificati dal nome del questionario, della domanda e dell' area. `obj` è scomponibile in **`obj.questionario`**, **`obj.domanda_A`** e **`obj.area`**, i tre *id* che ci permetteranno di identificare le tuple del database alle quali vogliamo riferirci.

Il codice della funzione `get(obj)` è il seguente:

```
function get(obj) { // Input ricevuto dal form
        var sel = obj.domanda_A;
      var getstr = "questionario=" + obj.questionario.value;
        getstr=getstr+"&quesito=" +
obj.domanda_A.options[sel.selectedIndex].value;
        sel = obj.area;
        getstr=getstr+"&area=" +
obj.area.options[sel.selectedIndex].value;
      startRequest(getstr);
}
```

Il recupero dei valori una volta ottenuto l'*id* è molto semplice, si tratta di prelevare il valore della *select* del form identificato dal metodo **`selectIndex`**. Una volta prelevati, i valori verranno salvati in una stringa che diventerà il parametro passato alla funzione `startRequest(getstr)` che invocherà la chiamata asincrona completando l'url con i parametri che andranno ad aggiornare il server.

2.3 CODICE DELLA CHIAMATA

Visto il metodo con cui vengono passati i parametri ora verrà analizzata la chiamata asincrona verso il server e la risposta di quest'ultimo.

Le funzioni interessate sono **`startRequest(`**`parametri`**`)`** e **`handleStateChange()`**.

La funzione `startRequest(parametri)` si occupa di effettuare la chiamata asincrona.

```
function startRequest(parametri){
     var url="dati.php" + parametri;
     createXMLHttpRequest();
     xmlHttp.onreadystatechange=handleStateChange;
     xmlHttp.open("GET",url, true);
     xmlHttp.send(null);
}
```

Alla sua invocazione prende come parametro una variabile chiamata **`parametri`**, una stringa che completerà l'url della chiamata asincrona con le variabili utili al server.

L'url completa sarà composta in questo modo: **`dati.php?questionario=Questionario_browser&quesito=Firefox&area=nord`**

Dopo l'assegnazione dell'indirizzo viene istanziato l'oggetto XMLHttpRequest attraverso la funzione **`createXMLHttpRequest();`**, la funzione per ricevere i dati di ritorno **`xmlHttp.onreadystatechange=handleStateChange;`** e infine viene aperta la chiamata asincrona con il metodo **`open`**.

Più complicata è invece la funzione della risposta **`handleStateChange()`** della quale sotto è illustrato il codice opportunamente semplificato dalle righe superflue alla spiegazione.

```
function handleStateChange()
{
      if(xmlHttp.readyState == 4){
            if(xmlHttp.status == 200){
            var returned=xmlHttp.responseText;
            var valori=returned.split(",");
            var myData = new Array();

            var colors =new Array();
            for (i=0; i<valori[0]; i++){
            var myData = myData.concat([[ valori[i+1],
            parseFloat(valori[i+parseInt(valori[0])+1]) ]]);
            colors=colors.concat(colori[i]);
            }

            var myChart = new JSChart('grafico', 'pie');
            myChart.setDataArray(myData);
            myChart.draw();

            var myDataBar=new Array(['Nord',
            parseInt(valori[valori[0]*2+2])], ['Centro',
            parseInt(valori[valori[0]*2+1])], ['Sud',
            parseInt(valori[valori[0]*2+3])]);
```

```
            var myChartBar = new JSChart('graficoBar', 'bar');
            myChartBar.setDataArray(myDataBar);
            myChartBar.draw();
            }
      }
}
```

La funzione inizia con i controlli visti nella sezione dei parametri della risposta affinché sia corretta.

Passati i controlli, viene assegnata alla variabile **`returned`** la stringa che contiene la risposta attraverso **`xmlHttp.responseText;`**.

La stringa è ottenuta dal server se stampiamo l'array di valori ottenuto dopo l'elaborazione in php è una stringa di valori separati da una virgola.

Per visualizzarlo: **`echo implode(",",$valori);`**

Con la variabile di ritorno la nostra visualizzazione dei dati non può ancora avvenire perché si tratta di una stringa e quindi, sfruttando la caratteristica di javascript di essere debolmente tipicizzato, la stringa verrà convertita in un array numerico.

Il metodo di conversione è una semplice assegnazione con una scrematura del separatore dei numeri: **`var valori=returned.split(",");`**

A questo punto non rimane che assegnare i risultati ai grafici.

Dato che possiamo avere diversi questionari con un diverso numero di elementi di ritorno occorre generalizzare la visualizzazione dei risultati.

Con questo scopo attraverso l'utilizzo di cicli `for` e giocando con l'indicizzazione degli array, i grafici che andremo a creare si adatteranno alla quantità di dati di ritorno.

Il metodo di input dei grafici è una funzione contenuta nella libreria JScharts e si chiama `myChart.setDataArray(myData);`.

Il parametro della funzione è `myData`, un ulteriore array bidimensionale che contiene i valori contenuti nella variabile `valori` convertiti a `float` con il comando di *parsing* `parseFloat(valori[i+parseInt(valori[0])+1])]);`. I valori di ritorno verranno affiancati al quesito a cui fanno riferimento.

L'assegnazione di `myData` (per il primo grafico) è la seguente:

```
var myData = myData.concat([[ valori[i+1],
parseFloat(valori[i+parseInt(valori[0])+1]) ]]); .
```

Lo stesso procedimento è applicato al diagramma a barre.

Assegnati i valori ai due grafici e visualizzati i risultati, il processo della chiamata Ajax termina.

2.4 BUG FIXES DI INTERNET EXPLORER

Questo paragrafo è dedicato a come risolvere un bug di Internet Explorer dovuto al passaggio dei parametri GET con Ajax.

Il codice presentato prima funziona correttamente con tutti i tipi di browser tranne che con Internet Explorer (testato sulle versioni 6, 7 e 8).

Il problema consiste nel fatto che in queste versioni di Internet Explorer che supportano Ajax i risultati non vengono aggiornati una volta inviati i nuovi parametri al server e quindi le statistiche rimangono sempre quelle precedenti all'aggiornamento.

Il problema risiede nel *caching* (memorizzazione delle url) delle chiamate HTTP di Internet Explorer che trovandosi davanti sempre la stessa url ripetuta (quella della chiamata asincrona di aggiornamento e quindi senza parametri) non visualizza mai gli stati nuovi del server ma quelli precedentemente contenuti in memoria.

In particolare nel codice questo bug fa riferimento alle chiamate di aggiornamento della pagina quindi quelle contenute nel `body` all'evento `onload` e nei pulsanti.

Per risolvere questo bug quindi bisogna inserire nella url un nuovo parametro che non influenzerà lo stato del server ma "ingannerà" il browser facendogli credere di avere sempre una url diversa.

L'idea è quella di mettere un numero casuale come parametro ad ogni richiesta e quindi nella variabile **`url`** contenuta nel codice della chiamata.

In javascript questo avviene con le funzioni **`escape(Math.random());`** oppure **`new Date().getTime();`** abbreviabile con **`new Date();`**.

La nuova variabile url da inserire sarà:

```
var url="dati.php?" + parametri + "&_=" +-new Date ;
```

Una nota, questo bug è presente anche in versioni passate di Firefox.

CAPITOLO III

SICUREZZA E USABILITA’

3.1 SICUREZZA E DEBUGGING

I limiti di sicurezza di Ajax derivano proprio dalla sua natura asincrona e quindi dal fatto che l'utente spesso è inconsapevole di cosa stia realmente accadendo durante la sua navigazione.

La mancanza del *refresh*, dell'utilizzo dei tasti *avanti* e *indietro* e il reperimento di informazioni senza *reload* crea un senso di disorientamento nell'utilizzatore e favorisce attacchi di tipo Cross Site Scripting (XSS) o SQL Injection che consistono nell'inserimento di codice "*malvagio*" che poi andrà a modificare lo stato del server eludendo validazione degli input o firewall.

Per evitare questi inconvenienti occorre quindi monitorare ciò che avviene durante la navigazione in pagine web che utilizzano Ajax.

A questo scopo ci vengono incontro strumenti di debugging molto efficaci e nel nostro caso cito Firebug, un'estensione del browser Firefox che ci permette una completa analisi del flusso della pagina.

Attraverso questi strumenti non risulta difficile ricostruire il percorso fino all'accesso diretto del servizio tramite parametri sensibili che il programmatore potrebbe non aver adeguatamente protetto.

Se il servizio fosse gestito direttamente da un linguaggio server a discapito della dinamicità queste informazioni risulterebbero molto meglio preservate e di difficile accesso.

Occorre quindi avere un ottima consapevolezza dell'applicazione web in modo da coprire ogni possibile falla.

Quando si sviluppa una applicazione *web 2* basata su Ajax bisogna concentrarsi su tre punti fondamentali.

Questi punti sono:

- La scoperta di chiamate nascoste
- Il crawling dell'applicazione (analisi tramite software)
- Analisi della logica

3.2 LA SCOPERTA DI CHIAMATE NASCOSTE

L'importanza di questo punto è stata sottolineata nel paragrafo precedente.

Strumenti come Firebug ci vengono di aiuto per cogliere le informazioni che ci saranno utili per identificare tutte le risorse dell'applicazione e le possibili vulnerabilità ad esse associate.

3.3 ANALISI TRAMITE CRAWLER

Un importante strumento di riconoscimento utilizzato nelle verifiche di sicurezza e indicizzazione delle pagine web è il **crawler**.

Un web crawler è un software che analizza i contenuti di una rete (o di un database) in modo metodico e automatizzato memorizzando tutte le risorse presenti.

Per fare questo prende in considerazione tutti i link che collegano le varie pagine.

Ajax che si appoggia su eventi per il reperimento delle informazioni e non su link, ha introdotto la necessità di una nuova generazione di crawler che interpretano come collegamento ipertestuale l'evento `onclick`.
Un esempio è il crawler *Chickenfood*, sempre contenuto come estensione di Firefox.

3.4 ANALISI DELLA LOGICA

Analisi della logica significa capire i vari passaggi che la richiesta effettua fino alla sua destinazione, che può essere un servizio.
Per fare questo occorre un'efficace tecnica di *debugging* che possa rilevare eventuali falle del codice che permetterebbero di risalire a punti critici dell'applicazione.
Il fatto che Ajax non abbia ancora uno standard di stesura e che sia strettamente dipendente dal lato client evidenzia la sua insicurezza.
Per questo il trattamento di dati sensibili o l'utilizzo di servizi come B2B, B2C, dove persiste uno scambio di informazioni importanti, devono essere sempre gestiti tramite server.
Diverso invece quando parliamo di forum, blog dove Ajax invece può dare il massimo delle sue potenzialità come efficienza di navigazione verso l'utente.

3.5 USABILITA'

Altri fattori negativi che si riscontrano nell'utilizzo di Ajax risiedono nella sua usabilità.

Usabilità intesa sia dal lato del programmatore che quello dell'utilizzatore.

Innanzitutto parliamo di Javascript e quindi di applicazioni o se generalizziamo di "veri software" che vengono eseguiti lato client. Questo implica un lavoro da parte della CPU che in presenza di tanti moduli Javascript potrebbe risentirne nelle prestazioni. Inoltre durante la fase di progettazione bisogna tenere conto anche della presenza di Javascript nei browser dei navigatori (l'11% degli utilizzatori non lo hanno attivato) e della quantità di informazioni asincrone che vengono trasmesse che potrebbero penalizzare gli utenti con connessioni più lente.

Un altro limite sta nel fatto del saper controllare le chiamate che vengono effettuate.

Saper controllare è inteso come avere coscienza del carico di lavoro assegnato ad ogni richiesta associata ad un evento.

Una chiamata asincrona può generare un gran numero di chiamate al server per ogni minimo cambiamento dello stato della pagina e se moltiplicate per il numero possibile di utenti che fanno uso di quel servizio contemporaneamente è facile trarre le conclusioni sui possibili risvolti nella reperibilità delle informazioni sul server.

Sempre riguardo la trasmissione di dati è anche utile calibrare bene l'evento da associare e dove associarlo. Spiegando con un esempio questa frase immaginiamo un form di assistenza tecnica: nel campo testuale dove scrivo le motivazioni del mio problema, commetto l'errore di mettere un'informazione che per esempio mi annullerebbe la garanzia e commesso questo errore, cancello la frase sbagliata. L'utilizzatore che scrive la ragione del problema non è consapevole del fatto che a quel form potrebbe essere associato un evento `onkeypress` che invierebbe informazioni ad ogni rilascio del tasto con la conseguenza che la frase errata è comunque stata inviata al server e che un destinatario potrebbe benissimo averla letta.

Questo esempio simbolico fa anche capire l'importanza (che risiedeva pure nel problema della sicurezza) del segnalare le richieste e le ricezioni che avvengono in background.

Un altro importante accorgimento è quello di non usare Ajax quando Ajax non serve. Una chiamata asincrona è inutile se viene utilizzata nello stesso modo di una chiamata sincrona. Una chiamata sincrona richiama una pagina e quindi interamente il peso del file da visualizzare. Utilizzare Ajax per richiamare un'intera pagina web è totalmente inutile in quanto non si guadagnerebbe nulla in termini di efficienza di visualizzazione penalizzando inoltre il client che dovrà anche eseguire le funzioni Javascript annesse. Quindi Ajax deve essere usato per il caricamento di porzioni di pagina con un limite di peso delle informazioni, anche se soggettivo al programmatore, pari ai ¾ di quella che si sarebbe trasferita in modo sincrono.

L’asincronicità interferisce anche nella ricerca delle informazioni. Questo problema riguarda i motori di ricerca che non indicizzano queste chiamate e non possono quindi effettuare un crawling all’interno di applicazioni dinamiche come Ajax (un problema riscontrato anche nei siti in flash) evidenziando quindi la necessità di avere anche una componente statica ai fini della reperibilità.

CONCLUSIONI

In questo paragrafo trarrò le conclusioni riguardo questa relazione.

Come anticipato nella parte introduttiva, il mondo del web si sta sempre di più orientando verso l'interattività da parte degli utenti e di conseguenza anche le applicazioni su cui si basa seguono questo percorso.

Flash e Java sono i pionieri di questo processo (in passato e anche oggigiorno) ma con delle limitazioni quali il necessario supporto di applicazioni come la Java Virtual Machine o Flash Player e le difficoltà di indicizzazione nei motori di ricerca.

Ajax invece avvalendosi delle potenzialità di Javascript interpretato dal client parte con il presupposto di compatibilità verso tutti i browser e facile reperimento delle informazioni in quanto visualizzate sempre in formato HTML o XHTML.

Se facciamo una panoramica sul web attuale, vediamo che i maggiori networks della rete si basano su questa tecnologia come ad esempio Youtube, WebMessenger, Google, Facebook e numerosi altri.

Molto banale come esempio, ma che può rendere l'idea di una sua funzionalità la troviamo nei form e nello specifico nel controllo dei campi. La struttura di un campo form viene solitamente analizzata tramite linguaggi lato server oppure tramite funzioni Javascript (dopo l'invio con metodo GET o POST), dove entrambi presuppongono l'invio dei parametri del form. Con Ajax e quindi sfruttando le potenzialità di XMLHttpRequest le analisi

avvengono in modo asincrono, con un risparmio notevole dei tempi standard per queste azioni e con una maggiore praticità.

Un esempio di quanto detto lo possiamo trovare al link: **http://zendold.lojcomm.com.br/fvalidator/exampleB.asp** .

Un altro esempio molto significativo lo possiamo trovare al seguente link **http://demo.tutorialzine.com/2009/09/shopping-cart-php-jquery/demo.php** . La demo rappresenta un carrello di ecommerce realizzato attraverso Ajax e PHP con un effetto drag and drop. Paragonando a ciò che è un classico carrello l'interattività di questa applicazione a mio avviso è veramente degna di nota e sempre come mio parere semplifica molto anche l'usabilità di strumenti di questo tipo.

Entrando nello specifico della relazione vorrei sottolineare il fatto che mi sono appoggiato a materiale acquisito prevalentemente tramite il web dato che il materiale cartaceo su Ajax dà una panoramica semplice sul suo funzionamento ma non evidenzia particolarità o specifiche come la risoluzione di bug oppure le problematiche di sicurezza e usabilità. Le varie tecniche di programmazione cross browser, il bug fix dell'esempio e numerose peculiarità del codice presente in questa relazione, mi sono state illustrate da una fruente community web (un ringraziamento specifico lo dedico a HTML.it e al suo forum) e questo a mio avviso evidenzia l'attualità dell'argomento che trova una sua evoluzione direttamente sulla rete.

Vorrei aprire una parentesi anche sul codice che ho utilizzato in questa tesi. Prendendo come esempio il codice della chiamata di XMLHttpRequest devo

sottolineare il fatto che ho usato una forma didattica che esplicita passo a passo la creazione dell'oggetto. Utilizzando frameworks JavaScript come Jquery, MooTools e altri, la chiamata all'oggetto è molto più semplice perché incorporata appunto in funzioni del frameworks.

Una precisazione rispetto a quanto detto all'inizio è che ora Ajax (XMLHttpRequest) è standard de facto per i browser.

Concludendo quindi vedo in Ajax una rivoluzione del web e senza accorgercene la maggior parte dei siti stanno adottando soluzioni asincrone.

Un futuro per l'interattività che si snoda lungo due grandi filoni paralleli: Ajax e Flash. Con i naturali limiti che un'applicazione basata su JavaScript ha nell'animazione di una pagina (non mi riferisco all'interattività da parte dell'utente ma alla grafica stessa) posso dire che Ajax sia comunque la scelta migliore.

FONTI CONSULTATE

BIBLIOGRAFIA

Professional Ajax
Nicholas C. Zakas, Jeremy Mcpeak, Joe Fawcett.
Wiley Publishing, 2006

Ajax, la grande guida
Ryan Asleson, Nathaniel T. Shutta.
Mondadori Informatica, 2006

Ajax per applicazioni web
Andrea Romagnoli, Pasquale Salerno, Andrea guidi.
Apogeo, 2007

SITOGRAFIA

http://www.adaptivepath.com/ideas/essays/archives/000385.php
Ajax: A New Approach to Web Applications di Jesse James Garrett, 18 febbraio 2005

http://www.securityfocus.com/infocus/1868/
Ajax security basics di Jaswinder S. Hayre e Jayasankar Kelath
SecurityFocus, 22 giugno 2006

http://www.aspitalia.com/articoli/asp.net2/AJAX-web-2.0.aspx
Introduzione ad AJAX: Il web del futuro di Stefano Mostarda, 9 maggio 2006

http://www.w3.org/TR/2009/WD-XMLHttpRequest-20090820/
XMLHttpRequest di Anne van Kesteren (Opera Software ASA) W3C Working, 15 aprile 2008

http://e-articles.info/t/i/732/l/it/
Trasmissione Delle Asincrone di Carol Rudenberg

http://javascript.html.it/guide/leggi/95/guida-ajax/
Guida ad Ajax di Andrea Giammarchi

http://it.wikipedia.org/wiki/AJAX

APPENDICE

CODICE JAVASCRIPT

```
<script type="text/javascript">

// variabile globale dell'oggetto di ritorno, nulla di default
var xmlHttp;

// funzione per assegnare l'oggetto XMLHttpRequest
// compatibile con i browsers più recenti e diffusi
function createXMLHttpRequest() //inizializzo l'istanza
{

      // variabile locale con le informazioni sul nome del browser
    browserUtente = navigator.userAgent.toUpperCase();

      // browser standard con supporto nativo
      // non importa il tipo di browser
      if(typeof(XMLHttpRequest) == "function" ||
typeof(XMLHttpRequest) == "object")
            xmlHttp=new XMLHttpRequest();

      // browser Internet Explorer
      // è necessario filtrare la versione 4
      else if(window.ActiveXObject && browserUtente.indexOf("MSIE
4") < 0) {

            // la versione 6 di IE ha un nome differente
            // per il tipo di oggetto ActiveX
            if(browserUtente.indexOf("MSIE 5") < 0)
                  xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
            // le versioni 5 e 5.5 invece sfruttano lo stesso nome
            else
                  xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
}

function Startup() {
    startRequest('questionario=<?php echo($questionario); ?>');
    timer = setInterval(function(){startRequest('questionario=<?php
echo($questionario); ?>')}, 5000 );

}

function get(obj) { // Input ricevuto dal form
        var sel = obj.domanda_A;
      var getstr = "questionario=" + obj.questionario.value;
```

```
        getstr=getstr+"&quesito=" +
obj.domanda_A.options[sel.selectedIndex].value;
        sel = obj.area;
        getstr=getstr+"&area=" +
obj.area.options[sel.selectedIndex].value;
      startRequest(getstr);
}

function startRequest(parametri) //avvio la richiesta
{

        var url="dati.php?" + parametri + "&_=" +-new Date ;
            createXMLHttpRequest();
            xmlHttp.onreadystatechange=handleStateChange;
            xmlHttp.open("GET",url, true);
            xmlHttp.send(null);

}

function handleStateChange()
{
      //Operazione completata
      if(xmlHttp.readyState == 4){
            if(xmlHttp.status == 200){
                //affiancato alla proprietà innerHTML uso il metodo
responseText per produrre un contenuto HTML
                  var returned=xmlHttp.responseText;
                  var valori=returned.split(",");
                var myData = new Array();
                  var colori = ['#0673B8', '#0091F1', '#F85900',
'#1CC2E6', '#C32121'];
                  var colors =new Array();
                  for (i=0; i<valori[0]; i++){
                var myData = myData.concat([[ valori[i+1],
parseFloat(valori[i+parseInt(valori[0])+1]) ]]);
                  colors=colors.concat(colori[i]);
                  }


                  var myChart = new JSChart('grafico', 'pie');
                  myChart.setDataArray(myData);
                  myChart.colorizePie(colors);
                  myChart.setTitleColor('#000');
                  myChart.setTitleFontSize(11);
                  myChart.setTextPaddingTop(15);
                  myChart.setPieValuesDecimals(1);
                  myChart.setPieUnitsFontSize(9);
                  myChart.setPieValuesFontSize(8);
                  myChart.setPieValuesColor('#fff');
                  myChart.setPieUnitsColor('#969696');
                  myChart.setSize(400, 300);
                  myChart.setPiePosition(0,0);
                  myChart.setPieRadius(95);
                  myChart.setFlagColor('#fff');
```

```
                    myChart.setFlagRadius(4);
                    myChart.setTooltipOpacity(1);
                    myChart.setTooltipBackground('#ddf');
                    myChart.setTooltipPosition('ne');
                    myChart.setTooltipOffset(2);
                    myChart.draw();

                    var myDataBar=new Array(['Nord',
parseInt(valori[valori[0]*2+2])], ['Centro',
parseInt(valori[valori[0]*2+1])], ['Sud',
parseInt(valori[valori[0]*2+3])]);

                    var colorsBar = ['#C32121', '#327AAD', '#009933'];
                    var myChartBar = new JSChart('graficoBar', 'bar');
                    myChartBar.setDataArray(myDataBar);
                    myChartBar.colorizeBars(colorsBar);
                    myChartBar.setTitleColor('#000');
                    myChartBar.setAxisNameX('');
                    myChartBar.setAxisNameY('');
                    myChartBar.setAxisColor('#C4C4C4');
                    myChartBar.setAxisNameFontSize(16);
                    myChartBar.setAxisNameColor('#999');
                    myChartBar.setAxisValuesColor('#777');
                    myChartBar.setAxisColor('#B5B5B5');
                    myChartBar.setAxisWidth(1);
                    myChartBar.setBarValuesColor('#2F6D99');
                    myChartBar.setBarOpacity(0.5);
                    myChartBar.setAxisPaddingTop(60);
                    myChartBar.setAxisPaddingBottom(40);
                    myChartBar.setAxisPaddingLeft(45);
                    myChartBar.setTitleFontSize(11);
                    myChartBar.setBarBorderWidth(0);
                    myChartBar.setBarSpacingRatio(50);
                    myChartBar.setBarOpacity(0.9);
                    myChartBar.setFlagRadius(6);
                    myChartBar.setTooltipPosition('nw');
                    myChartBar.setTooltipOffset(3);
                    myChartBar.setSize(440, 300);
                    myChartBar.draw();

            }
      }
}
</script>
```

CODICE PHP

```
<?php

include("param.inc");

if(isset($_GET['quesito'])){
//ok prende dati e nomi del quest
$questionario=$_GET['questionario'];
$xml = simplexml_load_file( $questionario );
foreach($xml->elementi as $elementi)
{
$etichette=$elementi->nome;
}

$quesito=$_GET['quesito'];
$area=$_GET['area'];

$query="update sondaggio set numero=numero+1 where
quesito='$quesito' and area='$area' and
questionario='$questionario'" ;
mysql_query($query);
}
else {
     $questionario=$_GET['questionario'];
     $xml = simplexml_load_file( $questionario );

foreach($xml->elementi as $elementi)
{
$etichette=$elementi->nome;
}
}


//risolvere quazndo è automatico l'aggiornamento
$valori= array();

$n_domande=count($xml->elementi); //contatore delle opzioni della
domanda
$valori[0]=$n_domande; //primo parametro il numero di domande per
adattare i grafici

$i=1; //contatore per inserire nell'array


$xml = simplexml_load_file( $questionario );
foreach($xml->elementi as $elementi) //assegnazione delle etichette
al vettore
{
$etichette=$elementi->nome;
$valori[$i]=$etichette;
$i++;
}
```

```
$query = "select sum(numero) as totale from sondaggio where
questionario='$questionario'"; //calcolo il numero totale di unità
$result = mysql_query($query);
$records = mysql_fetch_array($result);
$totale=0;
$totale=$records['totale'];

$i++;

$query = "select sum(numero) as conteggio from sondaggio where
questionario='$questionario' group by quesito";
$result = mysql_query($query);
while($records = mysql_fetch_array($result)){
$valori[$i]=$records['conteggio'] / $totale; //calcolo percentuali
$i++;
}

$i++;

$query = "select sum(numero) as conteggio from sondaggio where
questionario='$questionario' group by area";
$result = mysql_query($query);
while($records = mysql_fetch_array($result)){
$valori[$i]=$records['conteggio'];
$i++;
}


echo implode(",",$valori);


?>
```

CODICE DEL FILE INDEX.PHP

```
<?php

if(isset($_GET['questionario']))

$questionario=$_GET['questionario'];
else $questionario="Questionario_browser.xml";

$xml = simplexml_load_file( $questionario );
?>


<body onload="Startup()" style="background-color:#000; padding:0px;
margin:0px">
```

```
<div style="width:440px; height:120px; border:1px solid #EEEEEE;
margin:auto; margin-top:50px;
background:url(immagini/titolo.jpg)"></div>

<div style="width:440px; height:490px; height:auto !important; min-
height:490px; border:1px solid #EEEEEE; background-color:#000;
margin:auto; margin-top:15px;">

    <form id="form1" name="form" method="GET" action="index.php"
style="margin-left: 25px;  margin-top:10px;" >
        <span style="color:#FFCCCC; font-family:Verdana, Arial,
Helvetica, sans-serif; font-size:10px; margin-right:10px;">SELEZIONA
UN QUESTIONARIO</span>

        <select name="questionario" id="questionario">

        <?php include("param.inc");

$query = "select distinct questionario from sondaggio";
             $i=1;


$result = mysql_query($query);


while($records = mysql_fetch_array($result)){
             ?><option value="<?php echo( $records['questionario']
); ?>"><?php echo( "Questionario  $i" ); ?></option><?php
             $i++;
              }
                                                                    ?>
        </select>

        <input type="submit"  style="border:#000000 1px solid;
margin-left:36px;  background:#FFCCCC; font-family:Arial, Helvetica,
sans-serif; font-size:10px; cursor:pointer;" value="Seleziona"/>
    </form>

    <div style="margin-left:25px; background:#EEEEEE; height:1px;
width:388px; margin-top:10px;"></div>

    <form
action="javascript:get(document.getElementById('quesiti'));"
name="quesiti" id="quesiti" style="margin-left: 25px;  margin-
top:10px;">

    <li>
    <span style="color:#FFCCCC; font-family:Verdana, Arial,
Helvetica, sans-serif; font-size:10px; margin-
right:128px;">QUESITO</span>

    <select name="domanda_A">
    <?php
```

```
foreach($xml->elementi as $elementi)
    {
    ?>
    <option value="<?php echo ''.$elementi->id.'';?>"> <?php echo
''.$elementi->nome.'';?></option>
    <?php
    }
    ?>
    </select>

    </li>
    <li class="title">  <span style="color:#FFCCCC; font-
family:Verdana, Arial, Helvetica, sans-serif; font-size:10px;
margin-right:54px;">AREA DI PROVENIENZA</span>

    <select name="area" style="width:70px">
    <option value="nord">Nord</option>
    <option value="centro">Centro</option>
    <option value="sud">Sud</option>
    </select>
    <input type="hidden" name="questionario" id="questionario"
value="<?php echo($questionario); ?>" />
    <input type="submit"  style="border:#000000 1px solid; margin-
top:10px;  background:#FFCCCC; font-family:Arial, Helvetica, sans-
serif; font-size:10px; cursor:    pointer; margin-left:78px;"
value="Invio Dati"/>
    </li>

    </form>
    <div style="margin-left:25px; background:#EEEEEE; height:1px;
width:388px;  margin-top:10px;"></div>

<div id="slider1" class="csw">

<div class="panelContainer">



<div class="panel" title="Diagramma  ">


<div class="wrapper">



<h5><img src="immagini/square.png" style="margin-right:5px"
/>Diagramma circolare sulle percentuali</h5>



<div id="grafico" style="float:left; margin-left:20px; " ></div>


</div>
```

```
</div>

<div class="panel" title="Istogramma  ">

<div class="wrapper">

<h5><img src="immagini/square.png" style="margin-right:5px"
/>Istogramma sulla provenienza dei votanti</h5>

<div id="graficoBar" style="float:left; "  ></div>

</div>

</div>
            <div class="panel" title="Dettagli">

<div class="wrapper">

<h5><img src="immagini/square.png" style="margin-right:5px"
/>Dettagli sull'applicazione</h5>

<div style="width:390px; height:400px;  margin-left:25px; margin-
top:8px; color: #666666">Questa applicazione è basata su Ajax e PHP.
<br />Il suo funzionamento consiste nel leggere automaticamente
all'interno del web server tutti i questionari presenti in formato
XML e di caricarli all'interno degli option box. Una volta caricati
l'utente potrà scegliere quale questionario svolgere (Per semplicità
sono a due variabili). Una volta selezionata l'opzione la scelta del
quesito verrà inviata al server in modo asincrono ed elaborata. I
risultati verranno visualizzati sempre in modo asincrono e
consisteranno nell'aggiornamento dei grafici (ottenuti con la
libreria JSCharts di Js).  </div>

</div>

</div>
```

```
</div>

</div>

<div style="margin-left:25px; background:#EEEEEE; height:1px;
width:388px;  margin-top:10px;"></div>

<span style="color:#FFCCCC; font-family:Verdana, Arial, Helvetica,
sans-serif; font-size:10px; margin-left:25px; margin-top:10px;
margin-right:5px; float:left">OPZIONI</span>
<input type="submit" style="border:#000000 1px solid; margin-
top:8px; background:#FFCCCC; font-family:Arial, Helvetica, sans-
serif; font-size:10px; margin-right:5px; float:left;
cursor:pointer;" value="Stop autoaggiornamento"
onclick="clearInterval(timer);" />
<input type="submit" style="border:#000000 1px solid; margin-
top:8px; background:#FFCCCC; font-family:Arial, Helvetica, sans-
serif; font-size:10px; margin-right:5px; float:left;
cursor:pointer;" value="Start autoaggiornamento"
onclick="Startup();" />
</div>

<div style="width:440px; height:50px; margin:auto;"></div>

</body>
</html>
```

RINGRAZIAMENTI

Ringrazio il professor Graziano Deambrosis per avermi dato la possibilità di sviluppare questo argomento come tesi.
Un altro dovuto ringraziamento a tutta la community web di HTML.it che mi ha supportato durante lo svolgimento della relazione.
Per concludere ringrazio tutte le persone che si sono interessate al mio lavoro, Iulia per i suoi preziosi consigli sullo svolgimento e la mia famiglia per avermi permesso di frequentare questa facoltà.