Esploriamo Javascript![1]

di Ivan Venuti

L'Html è stato pensato (e creato) per presentare contenuti statici che, visualizzati da utenti diversi, diano sempre lo stesso risultato.
Per sopperire a questa limitazione sono possibili tre strade:

1. generare la pagina in modo personalizzato ogni volta che vi si accede. In questo caso la pagina di per sé rimane Html standard. È il Web Server (ovvero il computer che genera la pagina) a generarla personalizzata. Per questo le tecnologie usate sono chiamate "lato Server" (server side). Tipico esempio sono i programmi che permettono di leggere la posta elettronica attraverso il navigatore (browser): il contenuto viene generato in modo diverso per ogni utente e cambia quando arrivano nuovi messaggi.
2. Generare una pagina Html che contenga al suo interno elementi "dinamici", che permettono di personalizzare la pagina dopo che è stata letta dal navigatore (browser). Per farlo si usano le tecnologie Javascript e Css (la cui accoppiata viene indicata come Html dinamico o Dhtml). È una tecnologia che, avvenendo sul navigatore, viene chiamata "lato cliente" (client side). La visualizzazione della pagina viene cambiata senza ricaricare la pagina.
3. Uso di tecnologie sia lato server che lato cliente (combinando le tecnologie appena illustrate).

La programmazione lato server è particolarmente complessa e necessita di nozioni avanzate di programmazione, mentre la programmazione lato cliente è adatta anche a persone con poca esperienza di programmazione. Questo articolo illustrerà piccoli (ma significativi) esempi di codice Javascript per ottenere effetti dinamici nelle pagine Html.

Javascript

Il linguaggio Javascript è un linguaggio di programmazione interpretato e ad oggetti. Interpretato significa che è possibile scrivere le istruzioni in un file di testo (quindi anche in una pagina Html) e che esse, se corrette, vengono eseguite da un programma apposito senza dover essere "tradotte" in un linguaggio comprensibile all'elaboratore. Il programma che esegue la traduzione è lo stesso navigatore.
Il codice Javascript viene quindi inserito in una pagina Html all'interno di appositi tag:

```
<SCRIPT>
 … codice …
</SCRIPT>
```

In questo modo informiamo il navigatore che, quello che incontrerà all'interno del tag `<SCRIPT>`, non è semplice testo da visualizzare, ma è del codice Javascript che va interpretato.
"Ad oggetti" significa che gli elementi sono oggetti, a cui sono associate delle proprietà. La notazione "puntata" permette di associare le proprietà ai loro oggetti.
Per esempio: quadro1.bordo.colore = "rosso", indica che si assegna "rosso" alla proprietà colore dell'oggetto "bordo" dell'oggetto "quadro1".

1 Questo articolo è stato pubblicato su *Inter.Net* N. 71, Settembre 2001 edito dalla Edizioni Systems Comunicazioni S.r.l.; quest'ultima detiene i diritti di pubblicazione e utilizzazione economica di quest'articolo; l'autore è stato autorizzato a renderlo pubblico sul proprio sito Web personale.

Ogni navigatore mette a disposizione oggetti predefiniti che ne modellano alcune caratteristiche. Questo significa anche che navigatori di tipo diverso mettono a disposizione oggetti diversi, creando un problema di compatibilità.
Per esempio possiamo scrivere sulla barra di stato della finestra corrente del navigatore attraverso la proprietà status dell'oggetto window:

```
window.status = "Esempio banale"
```

Il risultato è visibile in **Figura 1**.

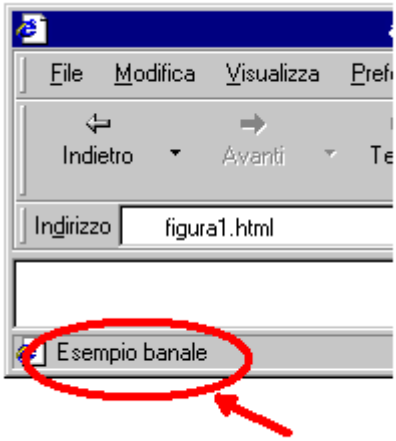

Figura 1: Proprietà status su oggetto window.

Altre volte è necessario accedere ai metodi, cioè alle funzionalità, dell'oggetto. Per farlo si usa la stessa notazione puntata, ma l'ultimo termine contiene delle parentesi tonde, ad indicare che è un metodo e non una proprietà. Dentro tali parentesi si possono scrivere i valori dei parametri del metodo (se esistono). Un esempio è il metodo alert(): visualizza la stringa che gli viene passata. Per cui l'istruzione

```
alert("Avviso importante")
```

dà l'avviso mostrato in **Figura 2**.

Figura 2: Esempio di alert

Esempi di base ed avanzati

Alcuni script più avanzati ci permettono di arricchire le pagine con interessanti effetti grafici. Il primo è il cosiddetto effetto roll-over (rotolamento): quando si passa il mouse sopra una immagine, essa cambia alcune caratteristiche (per esempio colore di fondo).
Il principio è molto semplice: bisogna riconoscere quando il puntatore del mouse è posizionato

sopra l'immagine interessata e far caricare una nuova immagine al posto di quella originale. Per prima cosa è necessario creare le due immagini (**Figura 3**). A questo punto si inserisce l'immagine di base con il solito tag:

```
<img src="esempio1.jpg">
```

Figura 3 Le immagini (esempio1.jpg ed esempio2.jpg) che contribuiscono a creare l'effetto roll-over.

Javascript permette di intercettare degli eventi, come lo spostamento del mouse o il click di esso su un elemento del documento. L'evento che interessa è onMouseOver. Quando questo viene intercettato facciamo cambiare il sorgente dell'immagine, in questo modo:

```
<img src="esempio1.jpg"
 onMouseOver="document.images[0].src='esempio2.jpg'">
```

Testando il risultato, la pagina carica esempio1.jpg di default e, quando ci si posiziona sopra con il mouse, essa cambia e viene caricata esempio2.jpg.
A questo punto si vuole completare l'effetto di "roll-over" ricaricando l'immagine originaria quando il cursore si sposta fuori dall'immagine:

```
<img src="esempio1.jpg"
     onMouseOver="document.images[0].src='esempio2.jpg'"
     onMouseOut="document.images[0].src='esempio1.jpg'">
```

È interessante notare come si accede all'oggetto immagine: attraverso `document.images[0]`. `document` è l'oggetto che rappresenta tutto il contenuto della pagina Html, `images` contiene tutte le immagini del documento: è un vettore, cioè un insieme ordinato di oggetti a cui si accede attraverso un numero (0 è la prima posizione). Si pensi ad un oggetto come a un cassetto dove ci sono delle cose, e un vettore come ad una cassettiera, in cui possiamo riferirci ai vari cassetti con un numero (0 per il primo, 1 per il secondo e così via).
Riferirsi alle immagini in questo modo è scomodo: se si inseriscono altre immagini prima dell'immagine in esame, è necessario modificare gli indici del vettore, in quanto la sua posizione relativa cambia. Per ovviare a questo inconveniente è possibile dare un nome alle varie immagini. Estendiamo l'esempio di partenza in modo da avere un menu con 3 elementi a cui ci si riferisce per nome (**Listato 1**). Il risultato della pagina Html è visibile in **Figura 4**.

Listato 1

```
<html>
<head>
</head>
<body>

<img src="esempio1.jpg" name="esempio"
onMouseOver="document.esempio.src='esempio2.jpg'"
onMouseOut="document.esempio.src='esempio1.jpg'">

<img src="home1.jpg" name="home" onMouseOver="document.home.src='home2.jpg'"
onMouseOut="document.home.src='home1.jpg'">

<img src="link1.jpg" name="link" onMouseOver="document.link.src='link2.jpg'"
onMouseOut="document.link.src='link1.jpg'">

</body>
</html>
```

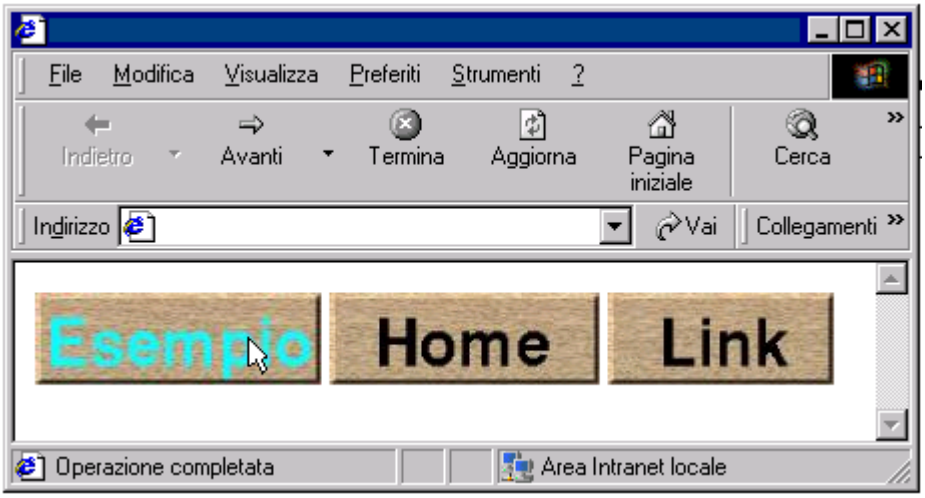

Figura 4

L'esempio di partenza può essere esteso ulteriormente in modo che, oltre all'effetto roll-over sull'immagine, possiamo anche visualizzare un'immagine diversa in un'altra posizione. Quest'ultima potrebbe essere, per esempio, un commento all'immagine su cui è posizionato il cursore del mouse. Il **Listato 2** presenta la realizzazione di questa idea, il cui risultato è visibile nella **Figura 5**.

Listato 2

```
<html>
<head>
<script>
      function cambia(nome, numero) {
            var obj = eval("document."+nome);
            obj.src = nome+numero+".jpg";
            if (numero==2) document.testo.src = "t"+nome+".gif";
                  else document.testo.src="tbianco.gif";
      }
</script>

</head>
<body>

<img src="esempio1.jpg" name="esempio" onMouseOver="cambia('esempio',2)"
onMouseOut="cambia('esempio',1)">
<img src="home1.jpg" name="home" onMouseOver="cambia('home',2)"
onMouseOut="cambia('home',1)">
<img src="link1.jpg" name="link" onMouseOver="cambia('link',2)"
onMouseOut="cambia('link',1)">

<table width="100%">
<tr><td align="center">
      <img  src="tbianco.gif" name="testo">
</td></tr>
</body>
</html>
```

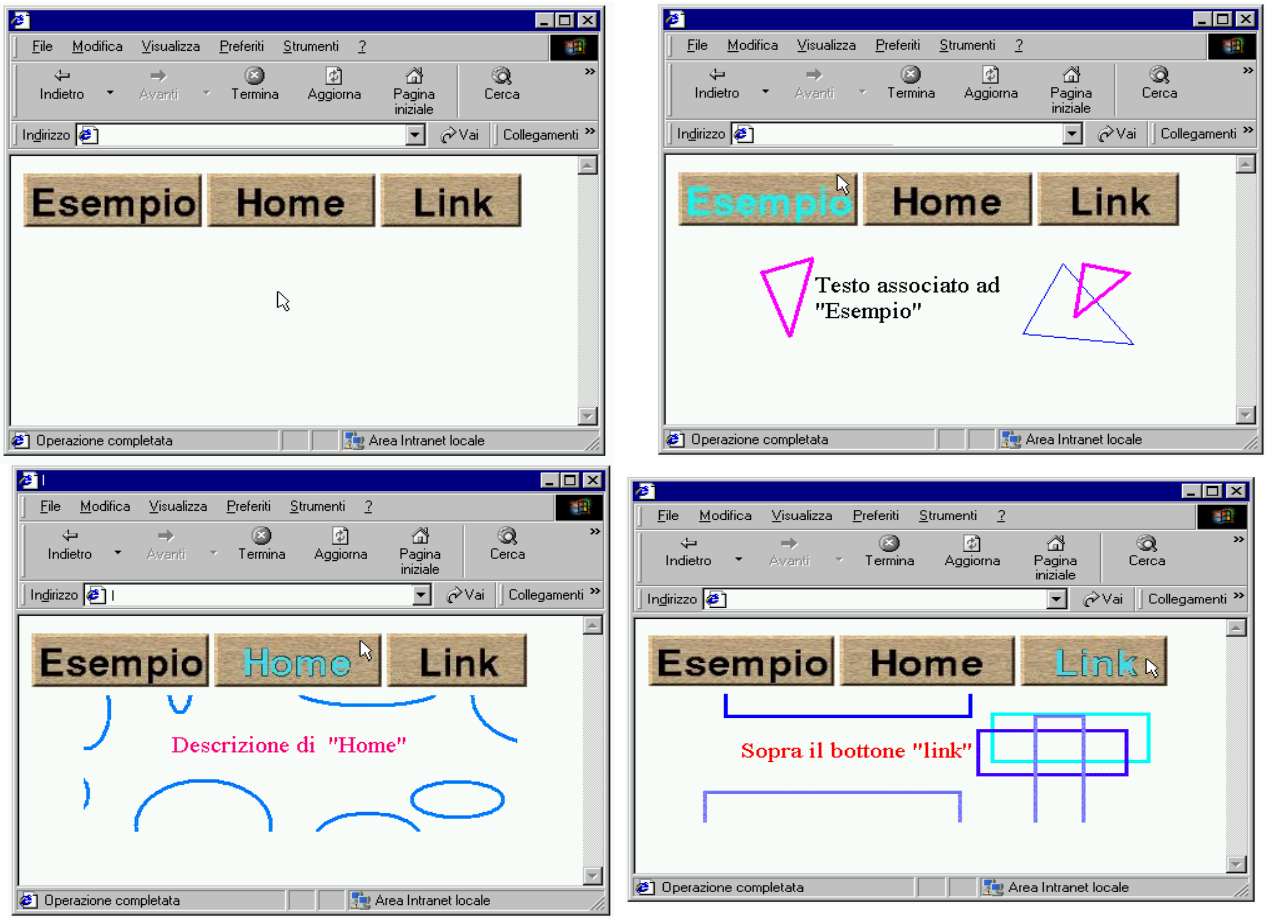

Figura 5 Esempio di roll-over "composto"

Nel **Listato 2** si è fatto uso di una nuova funzione predefinita: `eval()`. Essa ritorna l'oggetto descritto dalla stringa passata come argomento.
Ora cerchiamo di realizzare script che possano essere usati per cambiare i colori del testo di un documento o il colore dello sfondo.
Per farlo si usano due proprietà dell'oggetto `document`: `bgColor` ed `fgColor`.
Se vogliamo cambiare il colore dello sfondo in rosso si può usare l'istruzione:

```
document.bgColor = "Red"
```

o equivalentemente (usando la notazione RGB per i colori)

```
document.bgColor="#FF0000"
```

Da queste istruzioni creiamo uno strumento che permetta di personalizzare il colore dello sfondo o del testo di una qualunque pagina. Questo può essere utile, per esempio, a chi sviluppa pagine Web per testare l'impatto visivo dei diversi colori su una pagina.
Il risultato è riportato in **Figura 6**. Per realizzarlo è necessario modificare il file Html che si vuole personalizzare inserendo, nella sezione di testa (racchiusa tra <head> ed </head>), il seguente script:

```
<script>
     function apri(pagina) {
      nuova = window.open(pagina+".html",pagina,"toolbar=no,"+
          "location=no,directories=no,status=no,menubar=no,"+
          "scrollbars=no,resizable=yes,width=250,height=35")
     }
</script>
```

Tale script apre una nuova finestra con la pagina specificata e con le proprietà indicate.
Inoltre bisogna modificare il tag <body> in questo modo:

```
<body onLoad="apri('fondo'); apri('testo');">
```

Così facendo all'apertura del documento verranno aperte due nuove finestre contenenti ciascuna il codice che, in parte, è presentato nel **Listato 3** (tale codice è identico a meno del titolo del documento e della funzione scegli, che vale `window.opener.document.bgColor = colorName;` nella pagina che modifica il colore dello sfondo, mentre vale `window.opener.document.fgColor = colorName;` nella pagina che modifica il colore del testo). Con `window.opener` rappresenta la finestra che ha aperto il documento corrente.

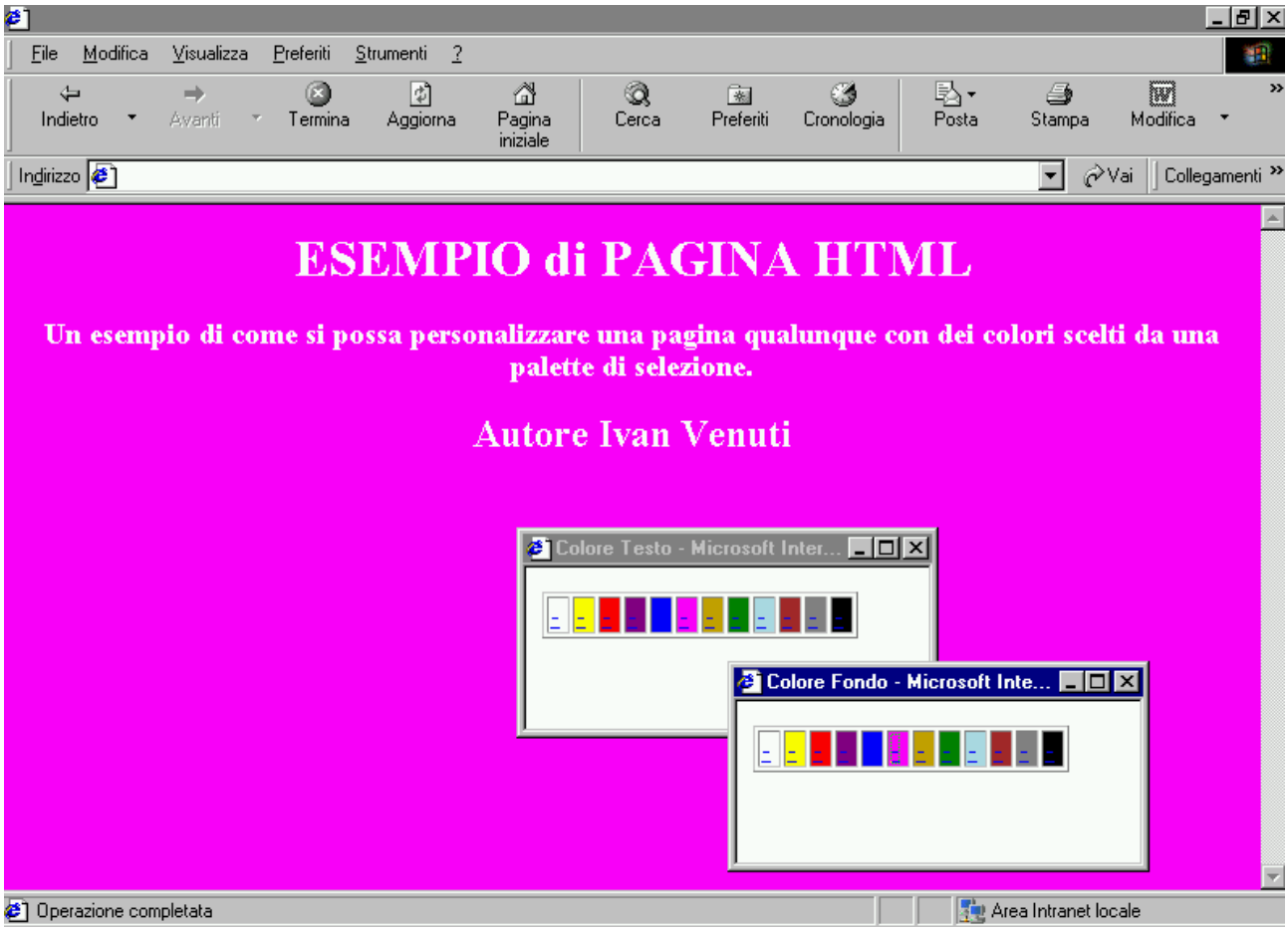

Figura 6

Il **Listato 3** è in gran parte Html standard e le funzioni Javascript usate sono davvero minime. È definita una funzione, `scegli(colorName)`, che riceve come parametro un colore e lo assegna al colore dello sfondo del documento oppure, nel caso del listato del tutto equivalente a meno di questa istruzione, al colore del testo. Questa funzione viene invocata intercettando l'evento `onClick` sulle ancore (link) dei colori.

```
<a href="#" onClick="scegli('White'); return false;">-</a>
```

Come si nota oltre a invocare la funzione da noi definita c'è un `return false` che impedisce al navigatore di seguire effettivamente l'ancora e ricaricare la pagina.
Ma perché inserire l'ancora se poi essa non viene usata? Ciò è necessario perché l'evento onClick non è associabile a tutti gli elementi di una pagina Html, ma solo ad alcuni: l'ancora è uno di questi, mentre non lo sono gli elementi testuali o elementi strutturali di una tabella.
Il problema della compatibilità tra navigatori diversi diventa evidente con l'esempio appena visto. Infatti esso funziona correttamente con Internet Explorer, mentre fallisce nel selezionare il colore del testo con Netscape. Il motivo è che quest'ultimo non riconosce la proprietà `fgColor`.

Listato 3

```
<html>
<head>
<Title>Colore Fondo</title>
<script language="Javascript">

	function scegli(colorName) {
		window.opener.document.bgColor = colorName;
	}

</script>

</head>
<body>
  <table border=1>
    <tr>
	<td width="10" bgcolor="white">
		<a href="#" onClick="scegli('White'); return false;">-</a>
	</td>
	<td width="10" bgcolor="Yellow">
		<a href="#" onClick="scegli('Yellow'); return false;">-</a>
	</td>
	<td width="10" bgcolor="Red">
		<a href="#" onClick="scegli('Red'); return false;">-</a>
	</td>
<!-- in modo simile c'e' il codice per i colori Purple, Blue, Magenta, Cyano, Green, LightBlue, Brown -->

	<td width="10" bgcolor="Gray">
		<a href="#" onClick="scegli('Gray'); return false;">-</a>
	</td>
	<td width="10" bgcolor="Black">
		<a href="#" onClick="scegli('Black'); return false;">-</a>
	</td>
    </tr>
  </table>
</body>
</html>
```

Supponiamo di avere un documento (documento A) che punta ad una pagina con molto testo e con molte ancore a parti di testo locali (documento X). La situazione è quella presentata in **Figura 7**.

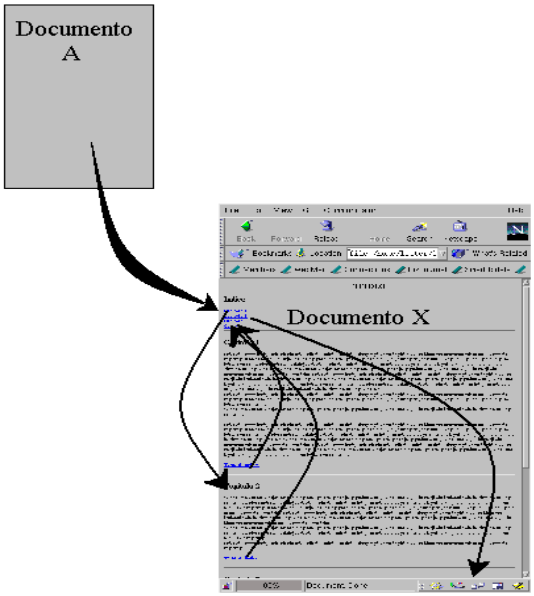

Figura 7

Chi conosce l'Html sa che le ancore locali permettono una facile navigazione all'interno del documento senza doverlo ricaricare ogni volta. Una volta navigato "localmente" si può avere la necessità di ritornare al documento di origine. Dalla situazione presentata basta mettere una ancora che punta al documento A. Nel caso si sia nella situazione presentata in **Figura 8**, non è così semplice.

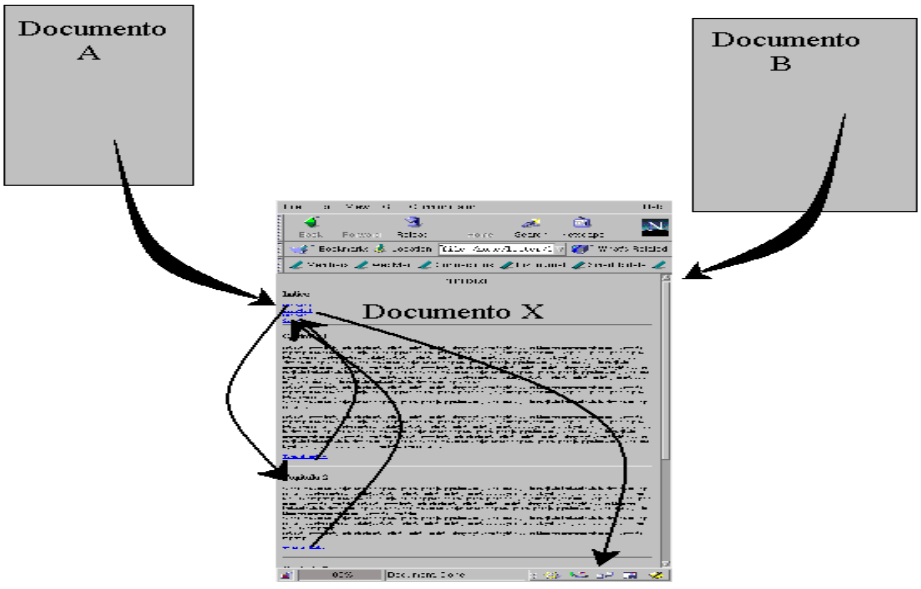

Figura 8

Infatti non sappiamo se siamo arrivati dal documento A o dal documento B. L'unica strada è quella di ritornare al documento di origine con il tasto "Indietro" (back) del navigatore. Ed ecco

l'inconveniente: se abbiamo navigato "localmente", dobbiamo premere il tasto "Indietro" tante volte quanti sono le ancore seguite. Questa situazione, chiaramente poco desiderabile, può essere risolta in Javascript in questo modo: si tiene traccia del numero di volte che abbiamo seguito le ancore "locali" e, quando si vuole ritornare al documento di origine, si esegue esattamente quel numero di salti all'indietro.
Il risultato è riportato nel **Listato 4**.
Come si vede è necessario definire una variabile globale `numero` e una funzione che incrementa tale variabile globale (funzione `incrementa()`). Ogni ancora locale invocherà la funzione incrementa(). Infine, nella ancora al documento di origine, si mette `history.go(-numero)`. L'oggetto `history` memorizza le pagine precedenti, e con la funzione `go(-1)` si simula il comportamento del tasto "Indietro", con `go(-2)` si simula la pressione di due volte il tasto "Indietro", e così via.

Listato 4

```
<HTML>
<HEAD>
…
<script>
        var numero=1;

        function incrementa() {
                numero++;
               return true;
        }
</script>
…
</HEAD>
<BODY>
        <a href="#locale1" onClick="incrementa()">Vai a locale 1</a>
        <a href="#locale2" onClick="incrementa()">Vai a locale 2</a>
        …
        <a href="#" onClick="history.go(-numero); return false">Pagina iniziale</a>
        …
</BODY>
</HTML>
```

Conclusioni

Gli esempi visti vanno a modificare le proprietà dell'Html standard. È possibile andare anche a intervenire sugli stili nel caso in cui la pagina contenga dei Css (Cascade Style Sheets).
Javascript permette di cambiare le proprietà del documento corrente, ma non solo: in presenza di un frameset, si può intervenire anche sui documenti collegati.
Insomma, gli effetti ottenibili sono innumerevoli. Su Internet è possibile trovare ottimi esempi di script pronti per essere usati.

Ivan Venuti. Laureato in Scienze dell'informazione, programmatore e docente di corsi di informatica, si occupa di tecnologie Object Oriented e problematiche legate al Web.