Appunti dei corsi di

Programmazione di Rete
Sistemi di elaborazione: Reti II

PROF. G. BONGIOVANNI

3) La programmazione nel Web

Al crescente successo del Web si è accompagnato un continuo lavoro per ampliarne le possibilità di utilizzo e le funzionalità offerte agli utenti.

In particolare, si è presto sentita l'esigenza di fornire un supporto a una qualche forma di interattività superiore a quella offerta dalla navigazione attraverso gli hyperlink, ad esempio per consentire agli utenti di consultare una base di dati remota via Web.

Le principali estensioni del Web da questo punto di vista sono:
- introduzione delle ***form*** per l'invio di dati al server Web;
- introduzione del linguaggio ***LiveScript*** (poi chiamato ***JavaScript***) per la definizione di computazioni, eseguite dal client, direttamente associate a una pagina HTML;
- introduzione del linguaggio ***Java*** per la creazione di file eseguibili, che vengono trasmessi dal server al client e poi vengono eseguiti in totale autonomia dal client;

3. 1) Estensione per mezzo delle form

Finora abbiamo sempre implicitamente considerato una URL come un riferimento ad un oggetto passivo, quale ad esempio una pagina HTML o un'immagine.

Ciò non è vero in generale, infatti una URL può fare riferimento a un qualunque tipo di oggetto, e in particolare anche a un file eseguibile.

Proprio grazie a queste generalità del meccanismo di indirizzamento URL si è definita una potente estensione Web, che lo mette in grado di integrare praticamente ogni tipo di applicazione esterna.

L'estensione di funzionalità di cui stiamo parlando è costituita da due componenti:
- sul client: la possibilità di offrire, nella pagina HTML, dei moduli, detti form, per immissione di dati e un meccanismo per l'invio di tali dati al server;
- sul server: un meccanismo funzionale per consegnare i dati provenienti dalla form ad una applicazione che sa gestirli, e per restituire al client gli eventuali risultati prodotti da tale applicazione.

Idealmente, la situazione è questa:

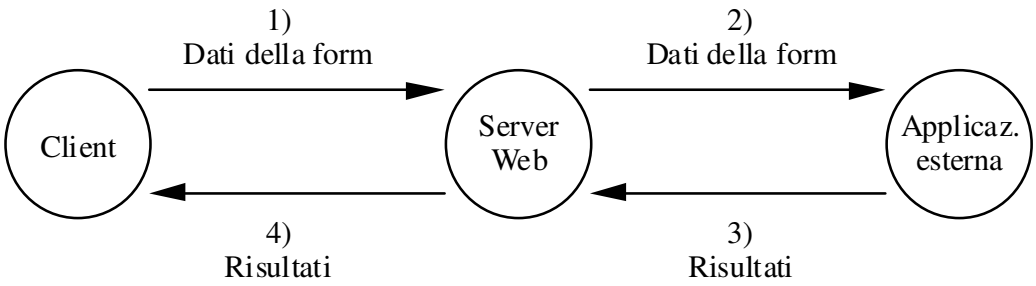

Figura 3-2: Dialogo fra cient, server Web e applicazione esterna

Ad esempio, supponiamo di avere questo scenario (che è anche il più tipico):

- esiste una base dati interessante, ad esempio quella di una biblioteca;
- si desidera consultarla via Web.

Immaginiamo che si voglia presentare all'utente un modulo di ricerca in biblioteca di questo tipo (in una pagina HTML):

Figura 3-3: Modulo per la ricerca in biblioteca

Esistono degli appositi tag di HTML destinati a creare simili moduli. è possibile includere:

- campi testo;
- radio button;
- checkbox;
- menù a tendina.

Nel nostro caso, il codice HTML che genera il modulo sopra visto è il seguente:

```
<HTML>
<HEAD><TITLE> ... </TITLE></HEAD>
</BODY>
<CENTER>
<H1>BIBLIOTECA DEI SOGNI</H1>
<H3>Modulo ricerca dati</H3>
<HR>
<FORM ACTION="http://somewhere.com/scripts/biblio.cgi">
Autore:<INPUT TYPE="text" NAME="autore" MAXLENGTH="64">
<P>
Titolo:<INPUT TYPE="text" NAME="titolo" MAXLENGTH="64">
<P>
<INPUT TYPE="submit"><INPUT TYPE="reset">
</FORM>
<HR>
</BODY>
</HTML>
```

Il tag `<FORM ... >` ha un parametro `ACTION` che specifica l'oggetto (ossia il programma) a cui il server dovrà consegnare i dati immessi dall'utente. In questo caso, vanno consegnati a un programma il cui nome `biblio.cgi` (vedremo poi il perché di questa estensione).

I tag `<INPUT ... >` definiscono le parti della form che gestiscono l'interazione con l'utente. In questo caso abbiamo:

- due campi per l'immissione di testo;
- due pulsanti, rispettivamente per l'invio dei dati e per l'azzeramento dei dati precedentemente immessi.

Quando l'utente preme il bottone "Submit Query" il client provvede a inviare i dati al server, assieme all'indicazione dell'URL alla quale consegnarli.

I dati vengono inviati come coppie:

`NomeDelCampo1=ValoreDelCampo1&NomeDelCampo2=ValoreDelCampo2` ecc.

dove:

- `NomeDelCampo` viene dal parametro `NAME`;

- `ValoreDelCampo` è la stringa immessa dall'utente in quel campo;
- le coppie sono separate dal carattere `&` e sono opportunamente codificate, se ci sono spazi e caratteri speciali.

La forma specifica del messaggio inviato dal client dipende anche da un altro parametro del tag `FORM`: oltre a `ACTION=...` c'è anche `METHOD=...` , che può essere:
- `GET`, assunto implicitamente (come nel nostro esempio);
- `POST`, che deve essere indicato in modo esplicito (questo è praticamente l'unico ambito nel quale è ammesso l'uso di tale comando).

Dunque, possiamo avere:
- `<FORM ACTION=".....">`
- `<FORM ACTION="....." METHOD="get">`
- `<FORM ACTION="....." METHOD="post">`

Nel primo e nel secondo caso, la richiesta inviata dal client consiste nella specifica della URL alla quale vengono "appesi" i dati richiesti (con un punto di domanda):

```
GET /scripts/biblio.cgi?titolo=congo&autore=crichton&submit=submit HTTP/1.0
User-agent: ...
Accept: ...
...
                                                <----- riga vuota
```

Nel caso del metodo `POST`, che in questo contesto è molto diffuso, i dati vengono rinviati nel corpo del messaggio:

```
POST /scripts/biblio.cgi HTTP/1.0
User-agent: ...
Accept: ...
...
Content-type: application/x-www-form-urlencoded
Content-length: 42
                                                <----- riga vuota
titolo=congo&autore=crichton&submit=submit      <----- dati della form
```

Il primo metodo è molto semplice, ma impone un limite al numero di caratteri che possono essere spediti. Il secondo invece non ha alcun limite, ed è da preferire perché probabilmente il primo verrà prima o poi abbandonato.

Quando il server riceve la richiesta, da questa è in grado di capire:
- i valori forniti dall'utente;
- a quale programma deve consegnarli.

Nel nostro esempio (che è un caso tipico) il programma `biblio.cgi` avrà l'incarico di:
- ricevere i dati dal server Web;
- trasformarli in una query da sottomettere al gestore della base dati;

- ricevere da quest'ultimo i risultati della query;
- provvedere a confezionarli sotto la forma di una pagina HTML generata al volo;
- consegnare tale pagina al server che la invia al client.

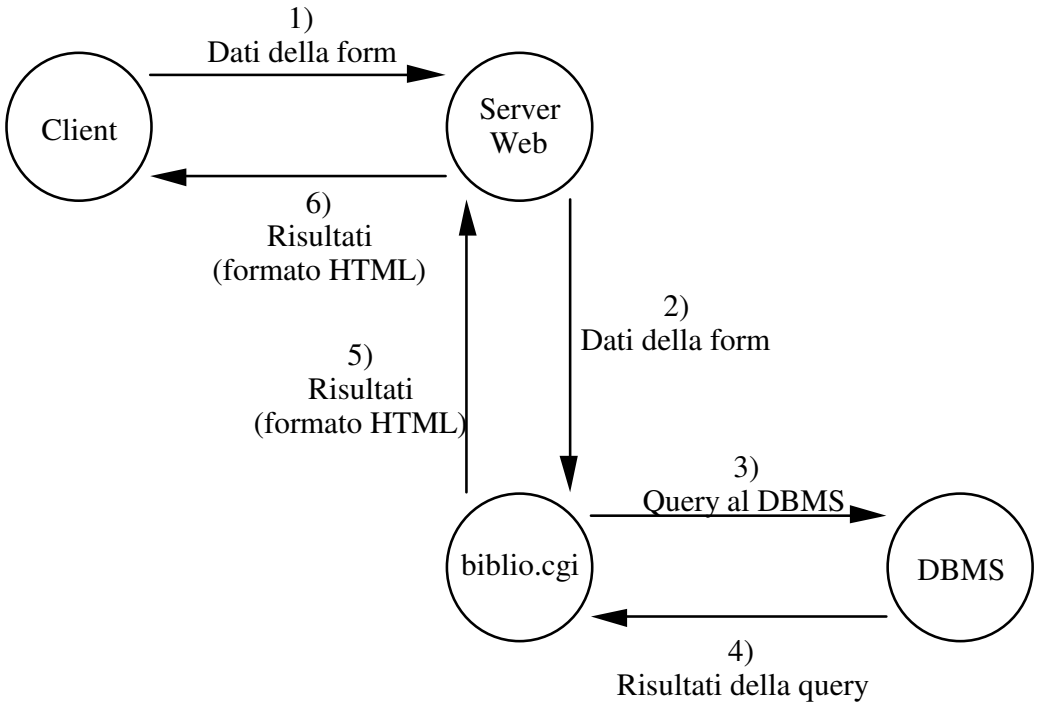

Figura 3-4: Dialogo fra cient, server Web, programma biblio.cgi e gestore della base dati (DBMS) della biblioteca

3. 2) Common Gateway Interface

Il programma `biblio.cgi` del nostro esempio precedente fa da ***gateway*** fra il server Web e l'applicazione esterna che si vuole utilizzare, nel senso che oltre che metterli in comunicazione è anche incaricato di operare le opportune trasformazioni dei dati che gli giungono da entrambe le direzioni.

Da ciò deriva il nome ***CGI*** (***Common Gateway Interface***) per lo standard che definisce alcuni aspetti della comunicazione fra il server Web e il programma gateway, e l'uso dell'estensione .cgi per tali programmi.

Un programma CGI può essere scritto in qualunque linguaggio:
- script di shell (UNIX);
- Perl, TCL;

- C, Pascal, Fortran, Java;
- AppleScript (MacOS).

Si tenga presente che lo standard illustrato qui è di fatto relativo alla piattaforma UNIX. Per altre piattaforme i meccanismi di comunicazione previsti dalla relativa Common Gateway Interface possono essere differenti, fermo restando che le informazioni passate sono essenzialmente le stesse.

Per la comunicazione dal server Web al programma CGI si utilizzano:
- variabili d'ambiente;
- standard input del programma CGI (solo con il metodo `POST`: i dati della form vengono inviati mediante ***pipe***).

Molte variabili d'ambiente vengono impostate dal server prima di lanciare il programma CGI; le principali sono:

QUERY_STRING	La lista di coppie `campo=valore` (usata con il metodo `GET`)
PATH_INFO	Informazioni aggiuntive messe dopo l'URL (usate ad esempio con le ***clickable map***)
CONTENT_TYPE	Usata con il metodo `POST`, indica il tipo MIME delle informazioni che arriveranno (application/x-www-form-urlencoded);
CONTENT_LENGTH	Usata con il metodo `POST`, indica quanti byte arriveranno sullo standard input del programma CGI

In più ci sono altre variabili per sapere chi è il server, chi è il client, per la autenticazione, ecc.

Le comunicazioni dal programma CGI al server Web avvengono grazie al fatto che il programma CGI invia l'output sul suo standard output, che viene collegato tramite pipe allo standard input del server Web.

Ci si aspetta che il programma CGI possa fare solo due cose:
- generare al volo una pagina HTML, che poi sarà restituita dal server al client;
- indicare al server una pagina preesistente da restituire al client (***URL redirection***).

L'output del programma CGI deve iniziare con un breve header contenente delle metainformazioni, che istruisce il server su cosa fare. Poi una riga vuota, quindi l'eventuale pagina HTML.

Le metainformazioni che il server prende in considerazione (tutte le altre vengono passate al client) sono:

Metainformazione	Valori	Significato
Content-type:	Tipo/Sottotipo MIME	Tipo dei dati che seguono
Location:	URL	File da restituire al client
Status:	OK, Error	Esito dell'elaborazione

Ad esempio, supponendo di usare un linguaggio di programmazione nel quale sia definita una ipotetica procedura `println(String s)` che scrive una linea di testo sullo standard output, un programma CGI che genera al volo una semplice pagina HTML potrà contenere al suo interno una sequenza di istruzioni quale:

```
println("Status: 200 OK");
println("Content-type: text/html");
println(""); //linea vuota che delimita la fine delle metainformazioni
println("<HTML>");
println("<HEAD>");
...
println("</HEAD>");
println("<BODY>");
...
println("</BODY>");
println("</HTML>");
```

Viceversa, se il programma CGI vuole effettuare una URL redirection il codice eseguito potrà essere:

```
println("Status: 200 OK");
println("Location: /docs/index.html");
println(""); //linea vuota che delimita la fine delle metainformazioni
```

Per gli altri sistemi operativi i meccanismi di comunicazione possono essere diversi:
- su Windows si usano variabili d'ambiente e file temporanei;
- su MacOS si usa Applescript: le informazioni (inclusa la risposta del programma CGI) vengono passate come stringhe associate a variabili dal nome predefinito.

Dunque, la tipica catena di eventi è la seguente:
- il client invia una richiesta contenente:
  - identità del programma CGI;
  - dati della form;
- il server riceve i dati;
- il server lancia il programma CGI e gli passa i dati;

- il programma CGI estrae dai dati i valori immessi dall'utente;
- il programma CGI usa tali valori per portare avanti il suo compito, che può essere ad esempio:
    - aprire una connessione con un DBMS (anche remoto) ed effettuare una query;
    - lanciare a sua volta un'altra applicazione, chiedendole qualcosa;
- il DBMS (o l'applicazione) restituisce dei risultati al programma CGI;
- il programma CGI utilizza tali risultati per confezionare al volo una pagina HTML, che restituisce al server;
- il server invia tale pagina al client, che la visualizza sullo schermo.

Ulteriori usi del meccanismo CGI si presentano in relazione a:
- server side image map;
- motori di ricerca.

Server side image map
L'idea è di avere sul client un'immagine sensibile al click del mouse che possa fare da "ponte" verso nuove pagine HTML scelte sulla base del punto dell'immagine sul quale si preme il pulsante.

Sul server sono necessari:
- una immagine (ad esempio, image.gif) e una pagina HTML che la contiene;
- un file di testo detto ***mappa*** che definisce le parti dell'immagine sensibili al mouse (ad esempio, image.map);
- un programma CGI di servizio (ad esempio, image.cgi).

Il funzionamento è il seguente:
- l'utente fa click su un punto dell'immagine;
- il client invia le coordinate del punto di click al server;
- il server passa le coordinate al programma CGI;
- il programma GCI esamina la mappa e, sulla base delle coordinate ricevute, restituisce un opportuno documento.

Supponiamo che:
- l'immagine image.gif sia di 100x100 pixel;
- il file image.map contenga le linee:

```
rect /paginaA.html 5,5,95,45
rect /paginaB.html 5,55,95,95
```

Tali linee impongono la seguente struttura (invisibile) sull'immagine image.gif:

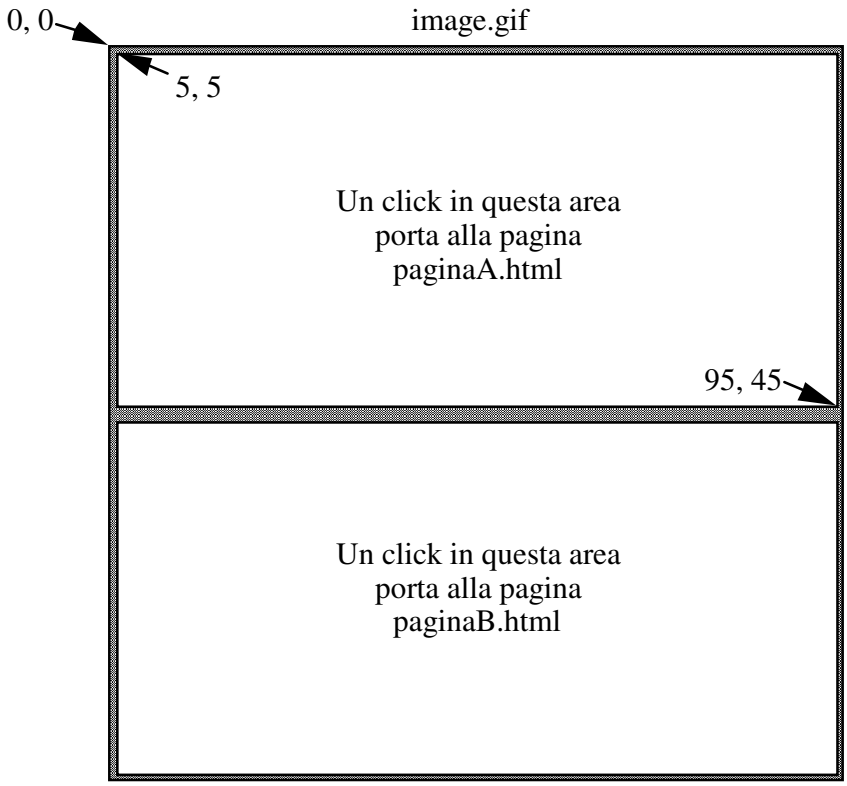

Figura 3-5: Aree sensibili definite da image.map

Nella pagina di partenza l'ancora che contiene l'immagine sensibile è del tipo:

```
<A HREF="image.cgi$image.map"><IMG SRC="image.gif" ISMAP></A>
```

Quando l'utente fa click, ad esempio, nel punto 50, 20 dell'immagine (l'origine è in alto a sinistra), il client invia la richiesta:

```
GET image.cgi$image.map?50,20 HTTP/1.0
```

Il server lancia il programma `image.cgi`, che riceve il nome del file di mappa e le coordinate, e quindi restituisce (tramite URL redirection) il file paginaA.html da mostrare al client.

Client Side Image Map
Recentemente è stato introdotto un nuovo meccanismo per ottenere un comportamento analogo senza bisogno di interagire col server.

Il client fa tutto da solo, poiché trova le informazioni necessarie direttamente nella pagina HTML.

In sostanza, si definisce la mappa in HTML e si dice al client di usare direttamente quella. Il caso precedente diventa:

```
<IMG SRC="image.gif" USEMAP="#localmap">
<MAP NAME="localmap">
<AREA COORDS="5,5,95,45" HREF="paginaA.html">
<AREA COORDS="5,55,95,95" HREF="paginaB.html">
</MAP>
```

Facendo click su una delle due aree, il client invia direttamente la richiesta della pagina A o B.

Per garantire la compatibilità con i client più vecchi, che non gestiscono il parametro `USEMAP`, si possono far coesistere le due possibilità:

```
<A HREF="image.cgi$image.map"><IMG SRC="image.gif" ISMAP USEMAP="#localmap"></A>
<MAP NAME="localmap">
<AREA COORDS="5,5,95,45" HREF="paginaA.html">
<AREA COORDS="5,55,95,95" HREF="paginaB.html">
</MAP>
```

Motori di ricerca
Esistono dei siti che offrono funzioni di ricerca di informazioni sull'intera rete Internet. In generale essi sono costituiti di 3 componenti:

- un ***robot di ricerca***, che è un programma che esplora automaticamente il Web e raccoglie informazioni sui documenti che trova (tipicamente, analizzando TITLE e HEAD). Con tali informazioni alimenta la base di dati di cui al punto successivo;
- una ***base di dati*** che raccoglie i riferimenti alle pagine collezionate, con opportune strutture di indici per velocizzare le ricerche;
- un ***CGI di interfaccia*** fra il Web e il programma di gestione della basi di dati, e una o più form per l'immissione dei criteri di ricerca da parte degli utenti.

3. 3) Linguaggio JavaScript (già LiveScript)

E' una proposta di Netscape. Consiste nella definizione di un completo linguaggio a oggetti, con sintassi analoga a Java, con il quale si possono definire delle computazioni direttamente nel testo della pagina HTML. Tali computazioni vengono poi eseguite dal browser.

Ad esempio, si può usare JavaScript per controllare che i vari campi di una form siano riempiti in modo sintatticamente corretto prima di inviare i dati al server.

Nell'esempio che segue, il bottone "submit" chiama una routine JavaScript di controllo del campo cognome. Se esso è vuoto si avvisa l'utente dell'errore, altrimenti si invia la form.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="LiveScript">
function checkForm (form){
	if (form.cognome.value == "") {
		alert("Errore: il campo \"Cognome\" non puo\' essere vuoto");
		form.cognome.focus();
		form.cognome.select();
	} else {
		form.submit();
	}
}
</SCRIPT>
</HEAD>

<BODY>
<CENTER>
<FORM>
Cognome:<INPUT NAME="cognome" VALUE="">
<P>
<INPUT TYPE="button" VALUE="Submit"
	onClick="checkForm (this.form);"
>
<INPUT TYPE="reset">
</FORM>
</CENTER>
</BODY>
</HTML>
```

L'effetto di un errore di immissione, dal punto di vista dell'utente, è il seguente:

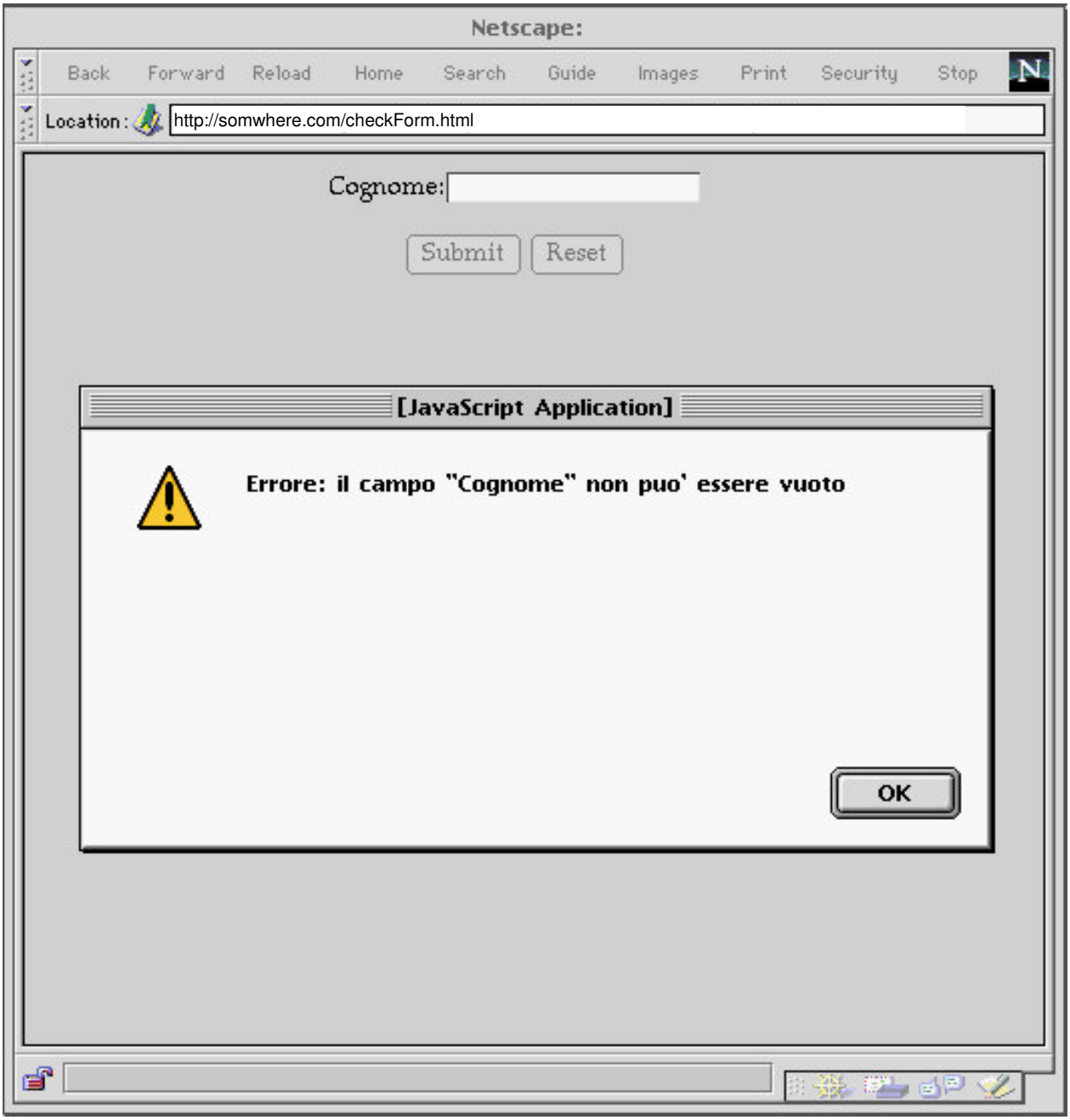

Figura 3-6: Messaggio di errore di JavaScript

Si noti che all'utente il bottone di invio sembra di tipo Submit, ma in realtà è di tipo Button, altrimenti la pressione su di esso causerebbe comunque l'invio dei dati della form.

Il linguaggio consente:
- la definizione di funzioni;
- la definizioni di variabili (non tipate: il tipo si deduce dall'uso);
- la gestione degli eventi, fra cui:
    - onClick;
    - onFocus, onBlur;
    - onChange;
- l'accesso a tutti gli oggetti, automaticamente istanziati, relativi al contenuto della pagina;
- la comunicazione bidirezionale con gli applet Java presenti nella pagina.

Gli oggetti predefiniti sono organizzati in una gerarchia i cui principali elementi sono:
- Document
    - Anchor
    - Applet
    - Form
        - Button
        - Checkbox
        - Radio
        - Submit
    - Textarea
    - Image
    - Link

Ogni singolo oggetto contiene una istanza di ciascun tipo di oggetto in esso contenuto: ad esempio, un oggetto Form contiene le istanze di tutti i campi immissione dati, bottoni ecc. che sono inclusi nella form stessa.

3. 4) Linguaggio Java

Anche disponendo delle form e dei programmi CGI, ciò che rimane impossibile è avere una rapida interazione con la pagina Web.

Inoltre, la vertiginosa apparizione di nuove tipologie di informazioni da gestire e nuovi tipi di servizi da offrire rende problematico l'aggiornamento dei client: troppo spesso si devono ampliare le loro funzionalità o si deve ricorrere a programmi helper e plug-in.

La prima risposta a entrambi questi problemi è venuta da una proposta della Sun che ha immediatamente suscitato un enorme interesse a livello mondiale: il linguaggio Java e il relativo interprete.

Originariamente (nei primi anni '90) il linguaggio si chiamava OAK ed era destinato alla produzione di software per elettrodomestici (orientati all'informazione) connessi in rete. Era progettato per:

- essere indipendente dalla piattaforma di calcolo;
- consentire una esecuzione senza rischi per l'attrezzatura di destinazione.

Successivamente il progetto fu riorientato verso il Web, il linguaggio ha preso il nome di Java ed ha conosciuto un enorme successo.

L'idea di fondo è molto semplice:

- una pagina Web contiene (fra le altre cose) un link a un programma scritto in Java, detto ***applet***;
- quando l'utente carica tale pagina, assieme ad essa viene recuperato il codice eseguibile (***bytecode***) dell'applet;
- dopo essere stato caricato, del tutto o in parte, l'applet viene mandato in esecuzione sul client "dentro" la pagina HTML.

In questo modo si ottiene di fatto la possibilità di estendere senza limiti la funzionalità del Web, infatti:

- se l'applet ha una finalità prettamente locale (gioco, applicazione specifica quale un word processor, tabellone elettronico, ecc.) si offre quella alta interattività che prima mancava al Web;
- viceversa l'applet può costituire una estensione delle funzionalità del browser. Ad esempio, l'inventore di un nuovo formato per la compressione dati (chiamiamolo MPEG-2000) scriverà un applet per la visualizzazione degli stessi, che sarà caricato dai client quando questi dovranno accedere a tali dati.

Sull'elaboratore client è necessario un interprete di bytecode, ossia una ***macchina virtuale Java*** (***Java Virtual Machine***, ***JVM***) che costituisce il cuore di tutto il meccanismo. Essa fornisce un dispositivo di calcolo astratto, capace di eseguire il codice contenuto nei file `.class` e dotato di:

- un set di istruzioni macchina;
- un insieme di registri, un program counter, ecc.

Si noti che attualmente i file `.class` si ottengono a partire da codice Java, ma questo non fa parte delle specifiche della macchina virtuale. Non è escluso che in futuro si possano usare anche altri linguaggi sorgente (con opportuni compilatori).

Naturalmente, per eseguire i bytecode su una macchina reale, bisogna realizzare un ***emulatore*** della macchina virtuale, il cui compito è tradurre i bytecode in istruzioni native della piattaforma HW/SW ospite.

Dunque, sono le varie implementazioni della macchina virtuale che offrono ai bytecode l'indipendenza dalla piattaforma.

Vari modi sono possibili per implementare emulatori della macchina virtuale Java:
- all'interno di un client Web: in questo caso l'emulatore della macchina virtuale risiede all'interno del codice eseguibile del client;
- applicazione a se stante: in questo caso il client viene configurato per usare la JVM come helper esterno (ad esempio, MS Internet Explorer può scegliere quale JVM usare);
- all'interno del S.O. (per ora con Linux, a breve anche con Windows e MacOS).

Nel secondo e terzo caso, la JVM è in grado di eseguire non soltanto gli applet, ma anche applicazioni vere e proprie scritte in Java, cioè programmi del tutto autonomi che possono risiedere permanentemente sulla macchina ospite e possono essere eseguiti senza utilizzare il client Web.

Tali programmi non sono soggetti alle restrizioni imposte (per ragioni di sicurezza, come vedremo poi) agli applet.

Esempio 11

Applet che apre una connessione di rete, funzionalmente identico all'applicazione dell'esempio 8.

Il codice è costituito da due classi. La prima, `BaseAppE8_applet`, è basata su quella dell'esempio 8 e provvede ad istanziare la seconda (inalterata rispetto all'esempio 8) che ha il compito di attivare la connessione e di gestire la comunicazione. Se ne mostrano nel seguito solo le porzioni di codice rilevanti.

La classe `BaseAppE8_applet` estende la classe `Applet`, che a sua volta estende la classe `Panel` che è un `Container` e dunque può contenere i necessari elementi dell'interfaccia utente. Inoltre, la classe `Applet` contiene il metodo `init()` che viene chiamato automaticamente all'avvio, analogamente a quanto accade col metodo `main()` di una applicazione.

```
//Applet multithreaded per l'apertura di una connessione di rete
import java.applet.*;
import java.awt.*;
public class BaseAppE8_applet extends Applet	{
	Label label1, label2, label3, label4;
	TextField textField1, textField2;
	TextArea textArea1, textArea2;
	Button button1, button2, button3, button4, button5, button6;
	BaseTConn baseTConn;
	}
//-------------------------------------------------
	public BaseAppE8_applet() {
		this.setLayout(null);
		//aggiunta elementi interfaccia utente
		resize(600, 460);
		show();
	}
//-------------------------------------------------
	public void init() {
		new BaseAppE8_applet();
	}
//-------------------------------------------------
	public boolean handleEvent(Event event) {
		// gestione eventi, come in Esempio 8
	}
//-------------------------------------------------
	void button1_Clicked(Event event) {
		//come in esempio 8, vale anche per tutti i successivi metodi
	}
}
```

Funzionamento nel Web

In una pagina HTML si fa riferimento all'applet con uno specifico tag:

```
<applet code=BaseAppE8_applet.class width=600 height=460></applet>
```

Quando il client incontra il tag Applet:

- riserva uno spazio di 600x460 pixel nella pagina;
- richiede con una GET il file `BaseAppE8_applet.class`;
- quando riceve il file lo passa alla JVM (interna o esterna) che:
    - effettua gli opportuni controlli (come vedremo in seguito);
    - se è tutto OK, avvia l'esecuzione dell'applet;
- mostra nello spazio riservato nella pagina (600x460 pixel nell'esempio) il contesto di esecuzione dell'applet;
- provvede a richiedere (su indicazione della JVM) gli ulteriori file `.class` che sono necessari all'esecuzione dell'applet.

E' possibile fornire all'applet anche ulteriori parametri, che vengono passati all'applet al momento del lancio, quali ad esempio:

```
<APPLET CODE="game.class" WITDH=200 HEIGHT=200>
<PARAM NAME="level" VALUE="5">
<PARAM NAME="weapons" VALUE="none">
</APPLET>
```

Il metodo:

```
public String getParameter(String paramName)
```

della classe `Applet` restituisce una stringa contenente il valore del parametro il cui nome è `paramName`.

Torna all'indice | Vai avanti