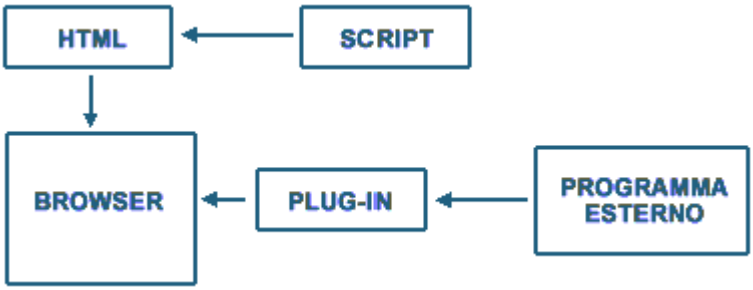
L'elaborazione dei dati su client
Linguaggi di script

Attualmente si tende a scaricare sul *computer* dell'utente piccoli programmi (*Javascript* o ***applet Java***) che svolgano parte dell'elaborazione in locale. Il vantaggio è sia nel tempo di risposta (più veloce perché si attendono meno dati dal ***server***) sia nel risparmio di elaborazione centrale, sia di altro tipo (per esempio può essere utile per criptare dati in modo più sicuro o per altro ancora). In sé, quindi, non è una cattiva scelta, però è più pericolosa, perché si basa sull'attrezzatura *hardware* e *software* dell'utente, che è **variabile**.



Per rendere dinamica una pagina *Web*, si utilizzano i cosiddetti **linguaggi di *scripting***, che sono dei veri e propri linguaggi di programmazione. Si distinguono dagli altri comuni, e più anziani, linguaggi perché non sono compilati ma **interpretati**. A seconda dell'**interprete** si parla di *scripting*:

- **server-side**: l'esecuzione dello *script* avviene sul ***server*** e propaga il risultato sul *client* che ne ha chiesto l'elaborazione;
- **client-side**: l'elaborazione avviene all'interno del *client* che ha caricato il documento html.

Se è il *server-Web (Apache, Internet Information Server)* che elabora uno *script* si parla di *scripting server-side (SSI, server side include)*, viceversa, se è il *browser (Netscape, Internet Explorer)* si parla di *scripting client-side*. Il linguaggio lato *client* per eccellenza, ma anche per anzianità, è sicuramente **JavaScript**.

Quando le operazioni da compiere non sono troppo complesse o troppo pesanti per il *browser*, si adottano gli *script client-side*, per esempio, quando occorre validare il contenuto di un campo di testo o realizzare piccole funzioni che sfruttano le informazioni del *browser*.

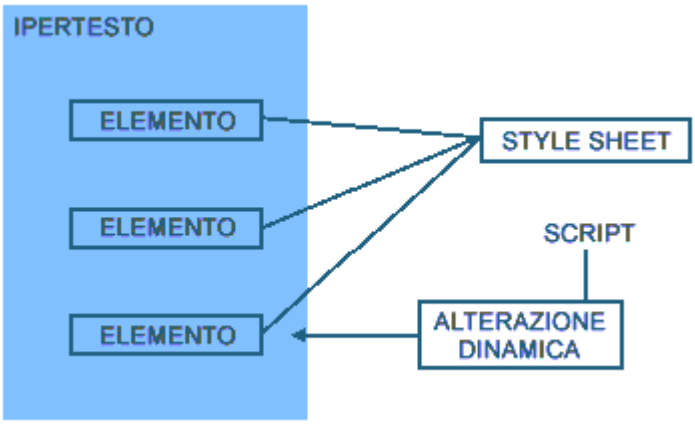
La necessità di uno scambio di informazioni più corposo tra il *client* e il ***server*** (quando, per esempio, si consulta una base dati) richiede l'utilizzo di *script server-side*. Lato ***server*** ci sono due modi per risolvere questa esigenza. Il primo è quello di realizzare *script* ***CGI***, programmi scritti in qualsiasi linguaggio (*C, C++, Visual Basic*) che eseguono delle operazioni e inviano il risultato di queste sotto forma html al *browser*. Il secondo, è quello di utilizzare linguaggi di *scripting* il cui codice è inserito direttamente nella pagine e processato dal ***server*** prima di essere inviato al *client* (ASP, PHP).

Abbiamo così chiarito sommariamente la differenza tra *scripting* ***server-side*** e **client-side**.

È necessario sottolineare l'importanza di ottenere precise garanzie riguardo alla compatibilità di questi pezzetti di *software* che si scaricano sul *computer* dell'utente. Devono funzionare bene sulle diverse piattaforme (PC, *Mac*, e *Unix*) e con diversi *browser* (*Explorer* e *Netscape* delle *release* più diffuse). *Macintosh* e *Netscape* solitamente vengono trascurati perché costituiscono una minoranza, che tuttavia non è trascurabile, soprattutto presso i consumatori e se il sito è destinato a grandi numeri.

Cosa è DHTML

DHTML o *Dynamic* HTML è un'evoluzione di HTML, nato per rendere meno statico il codice HTML delle pagine *standard*: purtroppo allo stato attuale il suo uso è implementato in maniera differente dalle versioni 4.0 dei *Web browser* quali *Netscape Navigator*, MS *Internet Explorer* ed altri, per cui non esiste uno *standard* unico per lo sviluppatore che desideri scrivere codice *cross-browser* ovvero non specifico per un *browser* particolare, come sarebbe invece utile.



HTML dinamico permette, attraverso l'uso congiunto di CSS (*Cascading Style Sheet*), *script* e codice HTML, di creare pagine *Web* interattive mediante i cosiddetti stili dinamici. Nella pratica ciò significa che gli sviluppatori di pagine *Web* hanno, grazie a queste nuove specifiche, la possibilità di modificare l'aspetto di un documento senza che questo venga ricaricato.

Un uso particolarmente interessante degli stili dinamici è quello che consente la creazione di sommari espandibili che, a seconda delle scelte effettuate, viene esposto o nascosto alla vista del *browser*. Uno degli esempi più evidenti di tale struttura è visibile nella *home page* di *Microsoft*.

Il modello di stile dinamico considerato negli esempi di questo approfondimento è compatibile esclusivamente con MS *Internet Explorer* 4, mentre *Netscape* ha percorso una strada diversa adottando il modello JASS (*JavaScript Accessible Style Sheets*). Tale incompatibilità è il freno maggiore allo sviluppo di **DHTML** presso i creatori di pagine *Web*, che poco tollerano l'idea di creare siti dinamici che discriminino l'accesso agli utenti dell'uno o dell'altro *browser*.

Stili dinamici

Gli stili dinamici permettono agli sviluppatori di rendere inizialmente visibile solo una parte del documento, con la possibilità, attraverso la definizione di elementi e classi in **DHTML**, di espanderne la struttura e visualizzarne il contenuto in modo analitico. Il codice che segue è un esempio molto semplice di tale struttura:

```
<HTML>
<HEAD>
<TITLE>Testo espandibile</TITLE>
<STYLE TYPE=text/css>
body {background:white}
.testo {color:#000080; font-size:10pt; FONT-FAMILY: verdana; cursor:help}
.times {color:red; font-size:14pt; FONT-FAMILY: times new roman;}
.vuoto {display:none}
</STYLE>
<SCRIPT LANGUAGE=JavaScript>
function stile(st) {
menu.className=st
}
```

```
</SCRIPT>
</HEAD>
<BODY ONCLICK=outliner();>
<H1 CLASS=fisso child=menu> DHTML </H1>
Questo testo viene formattato automaticamente senza dover ricaricare la pagina.
<p> </p>
<p><input type=radio value=V1 checked name=R1 onclick=stile(times)>times 14
pt<br>
<input type=radio value=V2 name=R1 onclick=stile(testo)>verdana 10 pt<br>
<input type=radio value=V3 name=R1 onclick=stile(vuoto)>nascondi il testo</p>
<p> </p>
<DIV ID=menu CLASS=times>
testo formattato...
</DIV>
```

In questo esempio vengono create tre classi: fisso, espandibile e vuoto. La classe fisso stabilisce gli elementi che, una volta cliccati, cambiano aspetto alla struttura e si associa all'attributo *child*. Quest'ultimo contiene l'ID dei dati da nascondere o visualizzare, e identifica, in modo univoco, gli elementi di questo genere all'interno della struttura. Nel caso specifico la classe fisso determina, oltre al colore blu del testo, il tipo di puntatore che il *mouse* deve assumere quando si trova su un elemento della classe stessa. Quando si clicca sull'elemento fisso, il nome di classe dei dati cambia da vuoto a espandibile, o viceversa, generando l'effetto desiderato.

Gli *SCRIPT* indicano al *browser* come interpretare le informazioni presenti in una pagina *Web*. In questo modo, relativamente alla marcatura presente tra `<SCRIPT> </SCRIPT>`, il *browser* delega all'autore la gestione degli eventi del documento. Nel momento in cui il *browser* legge, attraverso una particolare procedura chiamata di *parser*, la marcatura del documento ed incontra `<SCRIPT>`, ne passa il contenuto all'elaboratore di *script*, per poi continuare l'interpretazione del resto della pagina.

I *browser* che non supportano la gestione di *script* ignorano quanto posto all'interno di <SCRIPT></SCRIPT>.

Plug-in

Esistono attualmente sul mercato diverse soluzioni di distribuzione tramite *browser* dei dati *host* ai *desktop* che si basano sulla tecnologia di *Internet*. Tali soluzioni utilizzano vari metodi: tra i quali ricordiamo le ***applet***, ***ActiveX*** di *Microsoft* e la pubblicazione *host*. In questo documento verrà fornita una descrizione di ognuna di queste soluzioni e dei relativi punti di forza e limiti.

Applet Java* e *ActiveX

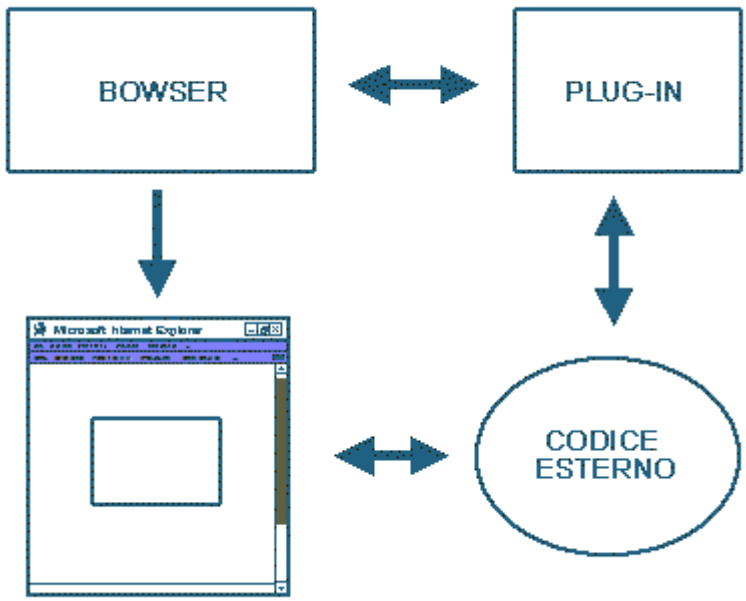
Le ***applet*** scaricabili si avvalgono delle nuove tecnologie distribuite di elaborazione dei dati, come ***Java*** e ***ActiveX***, per implementare applicazioni di connettività pienamente funzionali che possono essere scaricate direttamente nel *desktop*. Queste ***applet*** si integrano totalmente nell'ambiente del *browser* e in genere possono essere automatizzate con strumenti di *scripting* per *client* tipo *VB Script* e *Java Script*. È possibile sviluppare ***applet*** con livelli di funzionalità paragonabili a quelli dei prodotti di connettività *desktop* tradizionali, ma con un inconveniente: le ***applet*** sono sì sempre più funzionali, ma allo stesso tempo diventano più grandi e meno facili da scaricare. La maggior parte delle ***applet*** disponibili sul mercato offre meno funzionalità rispetto ai tradizionali prodotti *desktop*. È però probabile che, una volta acquisita maggiore esperienza con ***Java*** e ***ActiveX*** e trovata una soluzione circa i limiti della larghezza di banda delle reti, le ***applet*** raggiungano il livello di piena funzionalità dei *client* attualmente disponibili.

La scelta dell'***applet*** ideale per un ambiente specifico dipende in gran parte dalla tecnologia sulla quale tale ***applet*** si basa: ***Java*** o ***ActiveX***. Attualmente esistono differenze significative tra le due tecnologie, anche se nel tempo è probabile che tali differenze diminuiscano permettendo di scegliere

tra funzionalità diverse piuttosto che tra piattaforme di sviluppo diverse. In generale, i fattori che differenziano ***Java*** e ***ActiveX*** dipendono dall'approccio delle due tecnologie ad aspetti quali protezione, persistenza e supporto per piattaforme multiple. Per esempio adesso potreste non vedere l'activeX perché non è consentito dal vostro *browser*, oppure, se per *default* fosse consentito, questo potrebbe creare problemi di sicurezza.

Protezione

Le ***applet Java*** sono molto sicure. Con il linguaggio *Java* ogni ***applet*** scaricabile viene eseguita all'interno della propria macchina virtuale e non può uscire dai propri confini, né può leggere o scrivere sul disco rigido locale o in posizioni locali di memoria. Chiaramente, questo pone anche limiti alla funzionalità delle ***applet Java***. Con ***ActiveX***, invece, le ***applet*** hanno accesso completo alla memoria della macchina *client* e al supporto di memorizzazione locale. Sebbene utile, questa caratteristica rende gli oggetti ***ActiveX*** potenzialmente poco sicuri. Per risolvere il problema, è possibile registrare e assegnare una firma digitale alle ***applet ActiveX*** e garantire quindi l'origine dell'***applet***. Nel caso delle ***applet*** per la connettività, in genere acquistate presso un fornitore di fiducia e richiamate da una *Intranet* aziendale protetta, la sicurezza può passare in secondo piano.



Persistenza

Gli oggetti ***ActiveX*** sono persistenti, cioé rimangono all'interno della macchina locale dopo essere stati scaricati, mentre le ***applet Java*** vengono caricate in memoria e quindi eliminate quando il *browser* viene scaricato. La persistenza offre il vantaggio di non dover scaricare l'***applet*** ogni volta che deve essere utilizzata, ovvero permette di conservare una versione locale dell'***applet*** riducendo notevolmente il traffico di rete che il caricamento invece comporta. L'oggetto ***ActiveX*** viene scaricato solo nel caso in cui la versione presente sul ***server*** risulti più recente della versione disponibile sul *client*. Questa differenza è fondamentale a livello di rete, se accompagnata dall'uso di applicazioni specifiche eseguite sui *client* di piccole dimensioni, come le ***applet*** per la connettività.

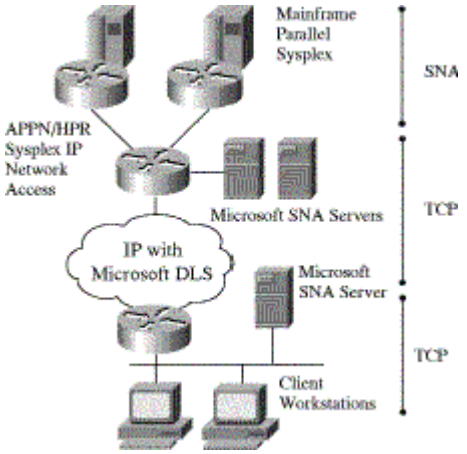
Supporto per piattaforme multiple

Al momento, le ***applet Java*** offrono un maggiore supporto per piattaforme multiple rispetto agli oggetti ***ActiveX***. I controlli ***ActiveX*** funzionano solo sulle piattaforme a 32 *bit* di *Windows* e su *Mac*

OS. Lo sviluppo del supporto per altre piattaforme è stato affidato a una società di terze parti. ***Java*** supporta non solo *Windows* 95 e NT, ma anche *Windows* a 16 *bit*, DOS, *Macintosh* e praticamente ogni variante di *UNIX*. Il supporto per piattaforme multiple diventa un fattore fondamentale al momento della scelta di un'***applet*** per la connettività solo se l'utente è in presenza di un ambiente misto con più piattaforme.

Pubblicazioni *Web-to-host*

I prodotti per la pubblicazione *host* sono strumenti di sviluppo *middleware* che consentono ai programmatori aziendali di creare applicazioni basate sul ***server*** in grado di incorporare i dati *host* in documenti HTML. Questo approccio è diverso dalla traduzione diretta precedentemente descritta, in quanto in questo caso lo sviluppatore mantiene il controllo sui dati *host* da visualizzare. Gli sviluppatori sono dunque in grado di riprogrammare il flusso dei *task* dell'**applicazione** *host*, fornendo agli utenti finali un'**applicazione** basata sul *Web* più intuitiva e semplice da usare.



I dati possono essere integrati con qualsiasi altro contenuto di tipo HTML, come la grafica, i collegamenti ipertestuali e il contenuto interattivo. Inoltre, i dati *host* provenienti da più fonti possono essere combinati con altri dati in modo da formare un unico documento HTML. Esistono due tipi di dati che vengono in genere pubblicati dagli sviluppatori in HTML: i dati memorizzati nei *database host* e i dati che vengono visualizzati sullo schermo da un'**applicazione** basata su *host*. Molte soluzioni garantiscono l'accesso a un solo tipo di dati, mentre molti progetti di pubblicazione *host* richiedono una combinazione di dati provenienti da entrambe le fonti.

Una grande percentuale dei dati aziendali risiede in grandi *database host* come DB2. Le soluzioni per la pubblicazione *host* che garantiscono l'accesso a questo tipo di dati utilizzano in genere un *driver* del *database*, spesso basato su ODBC, che traduce le interrogazioni del *database* in comandi interpretabili dal *database host*. I risultati dell'interrogazione possono quindi essere pubblicati in formato HTML.

È comunque possibile accedere alla maggior parte dei dati *host* solo attraverso le applicazioni *host* che visualizzano i dati sullo schermo di un terminale. I prodotti per la pubblicazione *host* che danno accesso a dati di questo tipo devono offrire un insieme di strumenti di programmazione in grado di semplificare l'interazione del programma con l'*host*. L'operazione più difficile nell'estrazione dei dati basati sullo schermo consiste nel creare il codice necessario per controllare la navigazione tra le varie schermate dell'**applicazione** *host*, fino a raggiungere la schermata contenente i dati.