Tecnologia per il commercio elettronico

appunti trascritti dagli studenti dell'AA 2011-2012
(ed integrati dal docente)

Antonio Lioy

v 1.10 - 19 agosto 2012

Indice

Versioni

versione	*data*	*commento*
1.0	16/3/2012	versione iniziale
1.1	24/3/2012	aggiunto capitolo 2, basato sul contributo di Francesca Lamanuzzi
1.2	31/3/2012	completato il capitolo 2 coi contributi di Angela Potenza e Rachele Rubinetti
1.3	8/4/2012	aggiunto capitolo 4, basato sul contributo di Giovanni Romano
1.4	28/4/2012	aggiunto materiale al capitolo 4 col contributo di Cremona Erica
1.5	6/5/2012	aggiunto capitolo 11 basato sul contributo di Morciano Manuel
1.6	16/6/2012	aggiunte le sezioni su HTTP/1.1 e sulla trasmissione dei form con GET e POST, basate sul contributo di Coletta Corrado
1.7	20/6/2012	aggiunte le sezioni sui form, su DOM ed i pericoli di JS, basate sul contributo di Bruni Simone
1.8	17/8/2012	aggiunto materiale al capitolo 4 e descritte architetture web statiche coi contributi di Tibaldi Jessica
1.9	18/8/2012	aggiunto capitolo 8, basato sul contributo di Perga Simone completato capitolo 8 col contributo di Bertola Alessandro aggiunto capitolo 9, basato sul contributo di Bertola Alessandro
1.10	19/8/2012	completato capitolo 9 e descritte architetture web statiche con pagina dinamiche coi contributi di Scarzello Marco

Capitolo 1

Introduzione

1.1 Premessa

Questo testo raccoglie gli appunti del corso "Tecnologia per il commercio elettronico" tenuto nell'anno accademico 2011-2012 al Politecnico di Torino dal Prof. Antonio Lioy.

Gli appunti sono stati trascritti dagli studenti indicati in tabella 1.1 e forniti al docente, che li ha impaginati ed integrati con le figure tratte dal materiale del corso. Il docente ha anche corretto alcuni errori grammaticali ed eliminato alcune inesattezze, ma non fornisce alcuna assicurazione in merito alla completezza ed esattezza delle informazioni qui riportate. In questa versione, il testo deve essere considerato per quello che è, ossia una raccolta di appunti forniti da vari studenti, con interventi minimi del docente relativi più alla forma che alla sostanza.

Questo testo è stato composto usando il sistema LaTeX, altamente consigliabile anche per la composizione delle tesi di laurea e di dottorato (nonché per qualunque testo di qualità che ecceda la decina di pagine).

1.2 Scopo e programma del corso

Lo scopo di questo corso è fornire agli aspiranti Ingegneri (primariamente quelli in Organizzazione dell'Impresa ma anche quelli di Logistica e Produzione) i riferimenti tecnologici per la valutazione, la progettazione e la gestione di sistemi di commercio elettronico basati sul paradigma web.

A questo fine, dopo un'introduzione generale alle diverse architetture distribuite di sistemi informatici (con valutazione qualitativa e quantitativa delle caratteristiche e delle prestazioni) si esaminano specificamente i vari tipi di architetture client-server a più livelli.

contributo	*studenti*
capitolo 2	Francesca Lamanuzzi Angela Potenza Rachele Rubinetti
capitolo	studente

Tabella 1.1: Studenti che hanno contribuito a questo testo.

Viene quindi approfondito il paradigma di sviluppo delle applicazioni basato sul web, introducendo il protocollo HTTP, i linguaggi HTML, CSS e Javascript, nonché l'ambiente ASP per la programmazione lato server. Pur senza trascurare la sintassi dei linguaggi studiati, l'accento viene posto soprattutto sulla loro funzionalità, reciproca integrazione e modalità preferibile d'uso per lo sviluppo di sistemi web efficienti (dal punto di vista delle risorse di calcolo e di comunicazione) ed efficaci (nell'interazione con gli utenti).

Infine vengono forniti dei cenni (necessariamente brevi data la vastità dell'argomento che potrebbe costituire un corso a sé) sui sistemi di pagamento elettronici e sulla sicurezza dei sistemi web.

Capitolo 2

Architetture di sistemi distribuiti

2.1 Le applicazioni informatiche

In generale un'applicazione informatica è sempre costituita da tre parti fondamentali (Fig. 2.1):

L'interfaccia utente (UI, User Interface)
: E' la parte tramite la quale l'utente dialoga col sistema informatico. Essa si occupa della gestione dell'input e dell'output, ovvero dei dati in entrata e in uscita. E' ciò che si frappone tra la macchina e l'utente, consentendone l'interazione.

La logica applicativa
: Svolge le elaborazioni richieste dall'utente, usando sia i dati forniti dall'utente sia altre informazioni necessarie.

I dati
: Tutti i dati e le informazioni necessari all'applicazione per fornire un dato servizio.

Storicamente le applicazioni informatiche sono state inizialmente realizzate in forma concentrata (la cosiddetta elaborazione classica) e quindi – con l'avvento delle reti – sempre più in forma distribuita.

2.1.1 Elaborazione classica

L'elaborazione classica si svolge tutta su un unico nodo di elaborazione e si basa su dati locali che possono essere condivisi o privati ed ha un unico spazio di indirizzamento (figura 2.2).

L'elaborazione avviene in modo sequenziale su un'unica CPU ed il flusso di elaborazione è univoco (fanno eccezione gli interrupt).

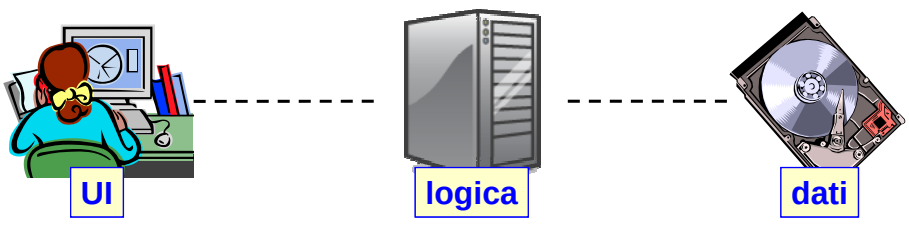

Figura 2.1: Modello di un'applicazione informatica.

```
#include <stdio.h>

int main ( )
{
    double percentuale_iva = 20;
    double prezzo;
char buf[100];
printf ("costo? ");
gets (buf);
sscanf (buf, "%lf", &costo);
    prezzo = costo * (1 + percentuale_iva / 100);
printf ("prezzo di vendita = %.2lf\n", prezzo);
return 0;
}
```


Figura 2.2: Esempio di un'applicazione classica

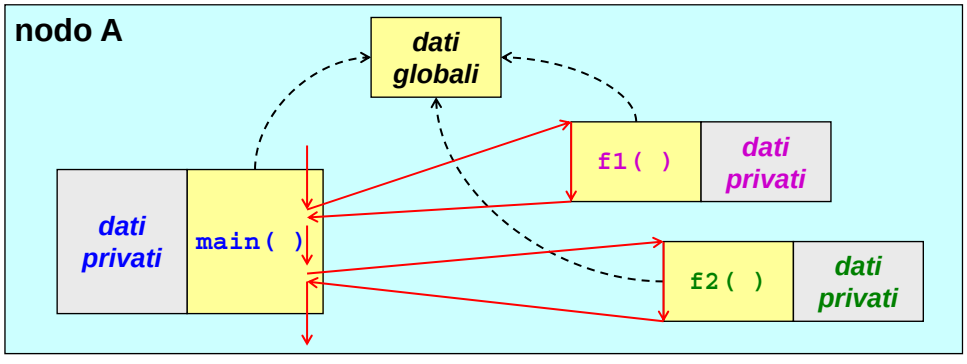

Figura 2.3: Flusso di elaborazione e localizzazione dei dati nel caso classico.

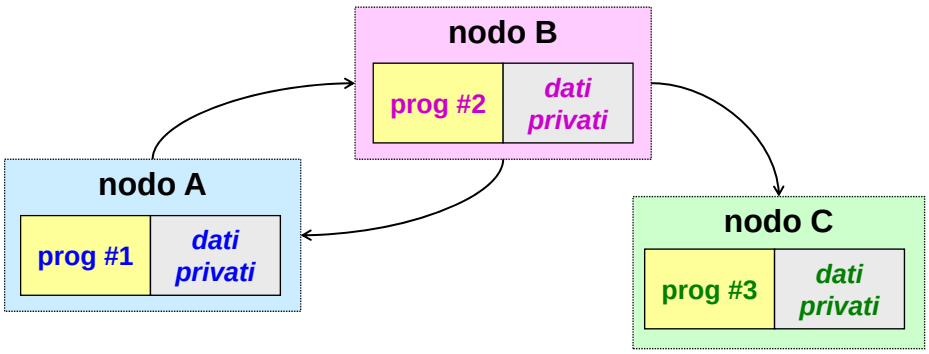

Figura 2.4: Flusso di elaborazione e localizzazione dei dati nel caso distribuito.

Come rappresentato in figura 2.3, ogni funzione del programma accede ai propri dati locali (quelli dichiarati all'interno della funzione stessa), ma può anche accedere a dati globali esterni alla funzione. Durante l'esecuzione di questo programma si può richiamare una funzione particolare che ha accesso a dati privati. In questo caso il `main` si interrompe, viene eseguita la funzione `f1` per esempio che restituirà il suo output e il programma ripartirà.

I vantaggi di questo tipo di elaborazione sono individuabili nella semplicità di programmazione e nella robustezza del programma.

Vi è, inoltre, la possibilità di ottimizzare il programma, accorpando più operazioni in una sola, in modo da ridurre i passaggi effettuati dal programma alla CPU e, conseguentemente, viene ridotto anche il tempo di esecuzione.

Non vi è alcun tipo di protezione sui dati, poiché essi sono globali. Inoltre si può accedere anche a dati di tipo privato, ciò è molto rischioso poiché chiunque può accedervi. Questo aspetto può essere migliorato con la programmazione ad oggetti, che prevede vari tipi di visibilità, anche se la protezione rimane comunque bassa.

Le prestazioni sono basse poiché vi è un'unica CPU ed è ad elaborazione sequenziale, cioè deve seguire per forza la sequenza di operazioni stabilite dal programmatore. Questa problematica può essere migliorata con dei sistemi multi-CPU o con una programmazione di tipo concorrente.

Inoltre l'uso si può effettuare solo tramite accesso fisico al sistema attraverso dei terminali. Dei collegamenti via rete o modem potrebbero ovviare a questo inconveniente.

2.1.2 Elaborazione distribuita

A differenza dell'elaborazione classica, in quella distribuita i dati sono solo privati e gli spazi di indirizzamento sono molteplici (figura 2.4). L'elaborazione non è più sequenziale bensì concorrente e si appoggia su CPU diverse. Con l'elaborazione distribuita inoltre vi sono molti flussi di elaborazione.

L'esempio in figura 2.5 rappresenta un programma utile ad un negoziante (per esempio) in cui si inserisce il costo di acquisto di un dato prodotto e in uscita fornisce il prezzo eventuale a cui, quest'ultimo, dovrebbe essere venduto, includendo l'IVA. Sul nodo A si trova la UI che dialoga con l'utente richiedendo l'inserimento da tastiera di un valore. In questo caso il dato da inserire è il costo d'acquisto di un prodotto da parte del venditore. Il nodo A non fa altro che prendere questo valore e inserirlo in una variabile chiamata appunto costo. A questo punto si passa sul nodo B su cui si trova la logica applicativa che dovrebbe svolgere

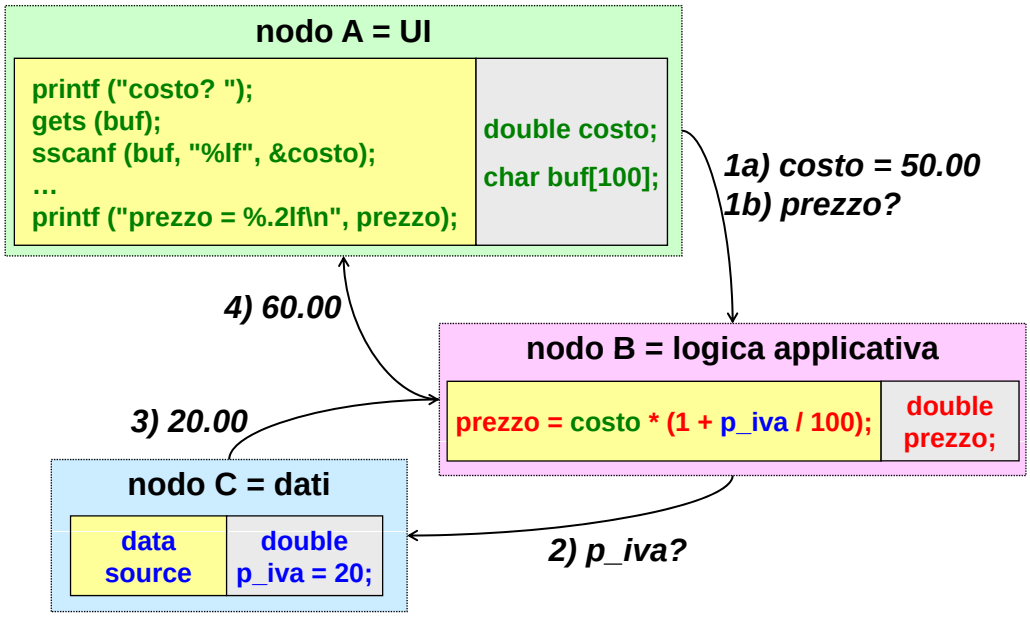

Figura 2.5: Esempio di un'applicazione distribuita

i calcoli. Per farlo ha bisogno però di un dato, la percentuale IVA da applicare, che si trova sul nodo C (preposto alla conservazione dei dati di business). Recuperato il dato, si torna sul nodo B che ora potrà effettuare il calcolo e restituire il risultato all'utente tramite il nodo A.

I vantaggi possono così sintetizzarsi:

elevate prestazioni, dovute alle molteplici CPU, quindi vi è globalmente una grande capacità di calcolo e sperabilmente anche una grande velocità di elaborazione;

buona scalabilità, poiché è più semplice aumentare il numero di CPU che la potenza di una sola di queste;

accesso tramite rete, ovvero non è necessaria la presenza fisica dell'utente poiché vi si può naturalmente accedere via rete.

Gli svantaggi sono:

complessità di programmazione perché con questo tipo di programmazione è difficile comprendere come i programmi possano interagire tra di loro, occorre preliminarmente scegliere il formato dei dati sui nodi e definire i protocolli. Inoltre la sincronizzazione delle operazioni può causare attese e rallentamenti.

scarsa robustezza poiché sono maggiori le probabilità di errore;

difficile ottimizzazione a causa della mancanza di una visione globale (il programma è spezzettato su vari nodi e quindi l'ottimizzazione può avvenire facilmente solo sui singoli nodi mentre è molto difficile fare un'ottimizzazione globale dell'applicazione).

2.2 Architetture software

Un'architettura software può essere definita come una collezione di moduli software (detti anche componenti) che interagiscono tra loro tramite un paradigma definito di comunicazio-

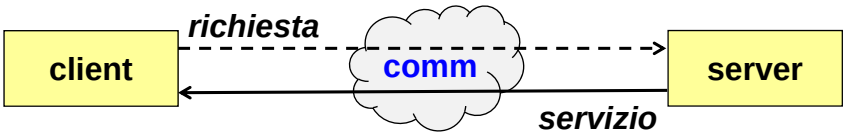

Figura 2.6: Architettura client-server.

ne (connettori). La comunicazione non avviene per forza via rete, può avvenire anche sullo stesso nodo mediante IPC (Inter-Process Communication). I modelli più diffusi di architetture software sono il client-server (abbreviato C/S) ed il peer-to-peer (abbreviato P2P). L'architettura C/S è asimmetrica poiché il server svolge la maggior parte delle attività ed il suo ruolo è determinato a priori. Invece l'architettura P2P è simmetrica perché ogni nodo può ricoprire il ruolo sia di client sia quello di server, simultaneamente o in tempi diversi. Entrambe queste architetture sfruttano i concetti di client e di server. Va specificato che il client ed il server sono distinti sia come elementi hardware sia come elementi software.

2.2.1 Server e client

Nello specifico il *server* viene attivato automaticamente al boot, ovvero con l'accensione della macchina o anche esplicitamente dal sistemista. Può accettare richieste da uno o più punti attraverso delle porte TCP-UDP (fisse e predeterminate), in modo tale che il client sappia che, per ottenere un certo tipo di servizio, deve indicare una determinata porta il cui numero è noto. Queste porte non cambiano, altrimenti il client non avrebbe più alcun riferimento. Idealmente il server non cessa mai il suo lavoro, ma in realtà potrebbe essere fermato tramite uno shutdown oppure da un'azione esplicita del sistemista.

A differenza del server, il *client* viene attivato solo su decisione dell'utente nel momento in cui quest'ultimo necessita di un dato servizio. Questa richiesta viene inoltrata su una di quelle porte fisse di cui si parlava prima. Il client, una volta inviata la richiesta, attende la risposta. Esso rimane in attesa su una porta allocata, invece, in modo dinamico, poiché potrebbero presentarsi più utenti simultaneamente quindi una porta fissa non sarebbe indicata. Il client non solo si attiva solo per fare la propria richiesta, ma una volta che ha ottenuto la risposta termina e non rimane in ascolto come il server.

2.2.2 Architettura client-server

In questo tipo di architettura, i client richiedono i servizi offerti dai server (figura 2.6). Nello specifico i vantaggi di questa architettura sono la semplicità di realizzazione e la semplificazione del client. Gli svantaggi sono il sovraccarico del server, che deve elaborare richieste da più utenti e il conseguente sovraccarico del canale di comunicazione.

Ci sono diversi tipi di architetture C/S che si distinguono in base al numero di elementi che costituiscono il server. Una di queste è la client-server 2-tier. Si chiama 2-tier perché, in inglese, tier significa strato e questa architettura è appunto a due strati.

L'architettura C/S 2-tier è quella originale in cui il client interagisce direttamente con il server senza attori intermedi. E' un'architettura che può essere sfruttata su scala locale o geografica e viene usata in ambienti di piccole dimensioni, ovvero con un massimo di 50-100 client simultanei. Ha lo svantaggio di avere una bassa scalabilità, poiché al crescere del numero di utenti decrescono le prestazioni del server proprio perché viene sovraccaricato di richieste. Poiché questa architettura dispone di soli due componenti (a fronte dei tre elementi

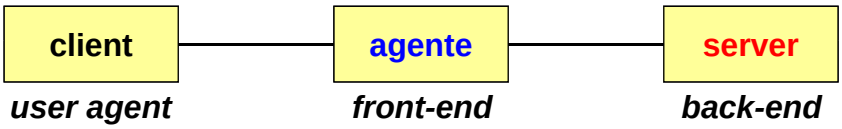

Figura 2.7: Architettura C/S 3-tier.

che concettualmente costituiscono un'applicazione informatica) c'è la possibilità di allocare differentemente la logica applicativa realizzando due diverse implementazioni.

Nell'implementazione detta *fat-client/thin-server* il client viene detto "grasso" poiché in esso coesistono sia l'interfaccia utente sia la logica applicativa, mentre il server è detto "magro" poiché si limita a gestire solo i dati di business. Per quanto riguarda questa soluzione vi è difficoltà di sviluppo perché servono dei software ad hoc e vi sono anche problemi di gestione e di sicurezza. È infatti indicata per ambienti ristretti nei quali si conoscono esattamente i client e le loro caratteristiche.

Nell'implementazione detta *thin-client/fat-server* il client è "magro" perché contiene solo l'interfaccia utente, mentre il server è "grasso" perché esegue la logica applicativa e gestisce anche direttamente i dati di business. Questa soluzione appesantisce i server, ma offre maggiore sicurezza. Internet ne è un esempio infatti lavora con questa seconda soluzione.

2.2.3 Architettura C/S 3-tier

In generale un sistema a tre livelli prevede che fra un client ed un server venga inserito un livello intermedio (spesso detto *agente* o *agent*) per svolgere compiti specifici (figura 2.7). Ad esempio, considerando uno schema generale, questo terzo livello potrebbe fungere da adattatore, bilanciatore di carico o mediatore.

Un *adattatore* è un sistema che effettua le opportune conversioni (di protocollo e di dati) per permettere il dialogo tra due sistemi altrimenti incompatibili. Ad esempio, si usano spesso degli adattatori per permettere il dialogo tra un *sistema legacy*[1] basato su mainframe e client che sfruttano il paradigma Internet. In questo specifico caso l'adattatore potrebbe effettuare un cambio di protocollo (da TCP/IP a SNA) e/o di formato dei dati (da ASCII a EBCDIC).

Un *bilanciatore di carico* (o *load balancer*) è un sistema che si presenta come un server ai client ma non fornisce esso stesso direttamente la risposta: smista la richiesta al server meno carico tra tutti quelli per cui funge da interfaccia pubblica. Ad esempio, grossi siti web come Amazon o Yahoo! usano un bilanciatore di carico per smistare il lavoro, non vi è un solo computer che risponde ma vi sono differenti server. L'apparecchiatura di rete riceve le richieste e le smista sul server che risulta più scarico; Questa definizione di bilanciatore di carico è molto semplificata rispetto alla varietà di tipologie di bilanciatori esistenti ed alle loro modalità di funzionamento. La definizione è però sufficiente per comprendere il concetto di bilanciatore ed è quindi funzionale agli scopi del corso.

Un *mediatore* (o *broker*) è un sistema che si presenta come un server ai client: ricevuta una richiesta, contatta vari server per selezionare e fornire quindi al client la soluzione migliore.

Mentre un bilanciatore di carico si interfaccia con tanti server equivalenti (ossia che fornirebbero tutti quanti la stessa risposta) e sceglie di contattare il server più scarico, un

[1] In inglese "legacy" significa "eredità". In informatica si etichetta come legacy un sistema basato su vecchie tecnologie che continua però ad essere usato per svariati motivi.

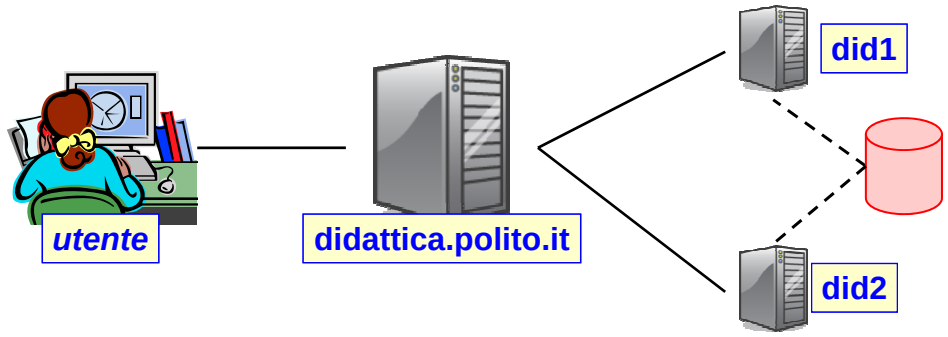

Figura 2.8: Esempio di architettura C/S 3-tier con bilanciatore di carico.

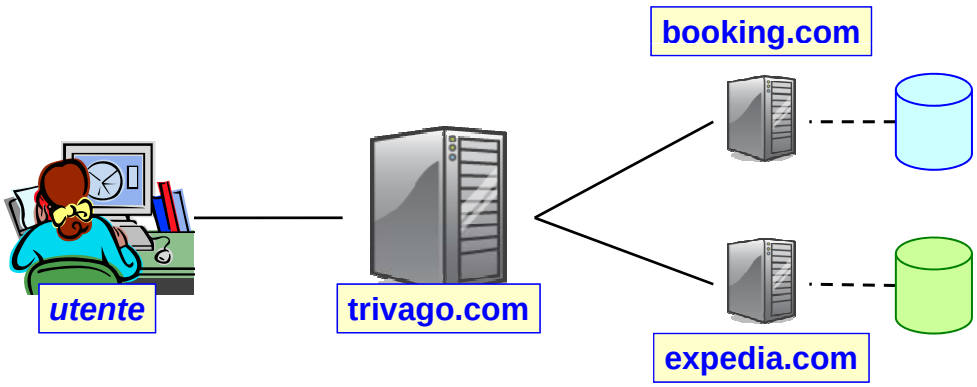

Figura 2.9: Esempio di architettura C/S 3-tier con broker.

broker si interfaccia con tanti server non equivalenti e li contatta tutti per selezionare la risposta migliore. Il compito di un broker è quindi più complesso di quello di un bilanciatore di carico.

I diversi tipi di agente trovano impiego in contesti diversi. Quando il problema è migliorare le prestazioni di calcolo, tipicamente si adopera un bilanciatore di carico. Ad esempio, il sito web `didattica.polito.it` non corrisponde ad un server reale ma è l'interfaccia pubblica di un bilanciatore di carico che si interfaccia con due server equivalenti `did1.polito.it` e `did2.polito.it` non accedibili direttamente (figura 2.8). Si osservi che i due server devono essere equivalenti (ossia fornire la stessa risposta) ma non necessariamente omogenei (ossia realizzati con la stessa tecnologia). Per migliorare la resistenza ai guasti ed ai problemi software sarebbe meglio acquistare due sistemi diversi dal punto di vista hardware e dotarli di due ambienti software differenti, pur svolgendo entrambi la stessa funzione. In qeusto modo risulta imporabile che si verifichi simultanemante lo stessio probolema su entrmabi i server, sia dal punto di vista harwdare sia da quello software. Questo approccio risulta però più costoso di quello che prevede server identici, perché non è possibile avere sconti per quantità e sono necessari sistemisti e programmatori che conoscono i due diversi ambienti software.

L'uso di broker è invece tipicamente motivato da una necessità applicativa. Ad esempio il servizio offerto da Trivago [2] (figura 2.9) consiste nel selezionare per un determinato servizio di viaggio la miglior offerta tra tutte quelle dei servizi intermediati da Trivago (Booking.com, Expedia, ...). In questo modo l'utente di dover far da sè la ricerca ed il confronto di tutte le possibili soluzioni. .

In un sistema C/S a tre livelli, il secondo livello è talvolta detto *front-end* perché è quello che si interfaccia direttamente con l'utente del servizio, mentre il terzo livello è detto *back-end*

[2] `http://www.trivago.com`

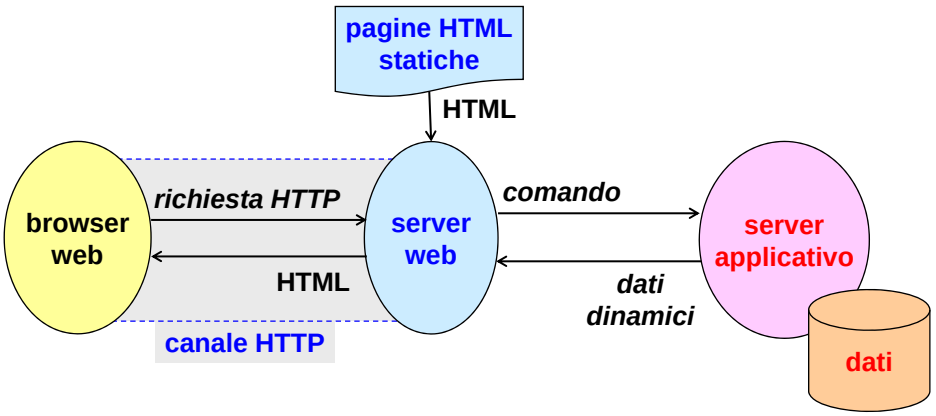

Figura 2.10: C/S 3-tier web-based – front-end leggero.

perché è quello che fornisce la base del servizio pur restando nascosto nel retro (*background* o *backstage* in inglese). Il terzo livello potrebbe essere molto complicato oppure semplicemente un sistema che fornisce file (file sistem), un database o una legacy application nel caso in cui il livello di mezzo sia quello che fa da adattatore fra formati nuovi e formati vecchi. L'UI può essere rappresentata da un Web browser, da un'applicazione java, dispositivo mobile o da un hand-held device.

2.2.4 L'interfaccia utente e il web

Per sviluppare la parte client che si interfaccia con l'utente si può seguire l'approccio personalizzato (in inglese *custom*) oppure quello basato su browser web.

Un'interfaccia custom consiste di un programma sviluppato su misura per le esigenze del cliente. Ciò rappresenta sicuramente un vantaggio per quanto riguarda la funzionalità ma al contempo richiede lo sviluppo di un programma ad hoc, il che richiede molto tempo e comporta la sua installazione e gestione su tutte le postazioni di lavoro. Inoltre avendo un'interfaccia specifica è necessario addestrare gli utenti riguardo l'utilizzo del programma e risolvere gli eventuali problemi che ne derivano.

Attualmente, dato il grande successo del paradigma web, si predilige adottare un'interfaccia web perché permette di non installare niente di aggiuntivo sulla postazione di lavoro dell'utente (nell'ipotesi che il browser web sia gìa presente) e non richiede uno specifico addestramento. In pratica, con questo approccio, l'interfaccia utente viene divisa in due parti: quella che effettua le operazioni di input-output è basata sul browser web ed è quindi presente sulla postazione di lavoro mentre sul server si usano appositi linguaggi (come HTML) per definire tutti i campi che costituiscono l'interfaccia utente e controllarne il funzionamento. In questo schema, Il browser manda richieste http e riceve risposte in html. Il server Web ha l'interfaccia descritta in html, in particolare html statico. Se il programma deve anche fare del lavoro il server Web dialoga con il server applicativo per ricevere dei dati. Il server applicativo accede ai dati posizionati sulla stessa macchina o su un quarto livello e fornirà la parte dinamica dei dati. Quello che vieni visualizzato da un utente è composto da dati statici e da dati dinamici. Questo processo è rappresentato nella figura 2.10.

Un secondo modello, rappresentato in figura 2.11, è quello in cui il server Web oltre a gestire le pagine statiche si occupa anche della parte applicativa. In questo contesto il terzo livello è soltanto il DBMS che viene contattato dal server tramite normali query SQL.

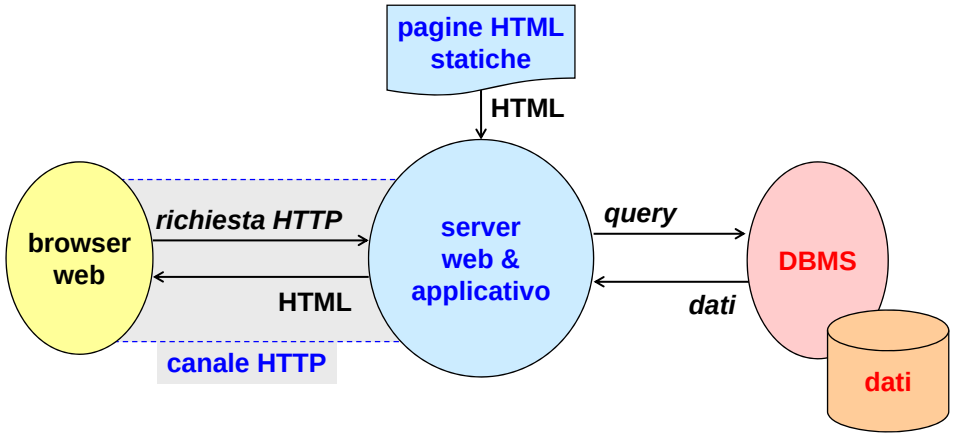

Figura 2.11: C/S 3-tier web-based – front-end pesante.

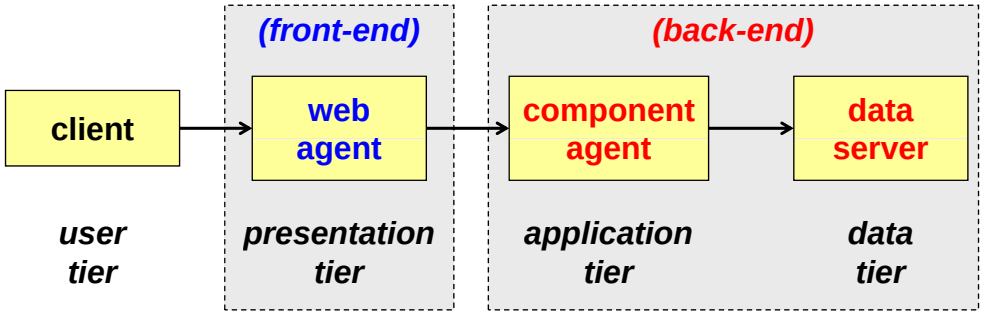

Figura 2.12: C/S 4-tier web-based.

2.3 Architettura C/S 4-tier

La soluzione migliore è quella con quattro livelli. Come si può vedere dalla figura 2.12, il lato front-end è il cosiddetto presentation tier che si occupa della presentazione dei risultati, mentre il back-end è diviso in component agent, che gestisce le logiche applicative, ed un data server, che gestisce i dati.

I quattro livelli possono essere situati su quattro o tre o minimo due macchine diverse. Le architetture a tre o quattro livelli mirano ad assegnare un determinato ruolo a ciascun livello in modo da ottimizzare le prestazioni dal punto di vista del CPU. Posso avere CPU velocissime ma il front end rimane il collo di bottiglia in quanto, anche se la CPU è veloce, l'efficienza del servizio dipende anche dalla rete. Non si può controllare la parte di rete tra client e front end, si deve perciò apportare dei miglioramenti. Per far ciò si stila una statistica sulla provenienza dei client andando a vedere, per esempio, l'indirizzo IP. Se un server è posizionato presso Telecom Italia, tutti gli utenti che hanno un contratto con Telecom Italia godranno di un servizio veloce perché la adsl sta sulla stessa rete IP dove sono collocati i server; viceversa tutti gli utenti che hanno fatto un contratto con la rete Vodafone sono svantaggiati in quanto il traffico che arriva a Telecom da utenti, che sono supportati da altri operatori, è limitato da un filtro. Colui che fornisce il servizio di Internet si chiama ISP e gli accordi tra diversi ISP si chiamano PEERING che stabiliscono il traffico che viaggia tra diversi ISP.

Se metto un server in un determinato luogo, non ho modo di sapere quale sarà la banda verso i miei clienti. E' molto importante, quindi, conoscere l'indirizzo di provenienza dei clienti e questo posso farlo attraverso il servizio WHO IS che, dato un indirizzo IP, fornisce la rete tramite la quale l'utente riesce a collegarsi. Quindi se ad un server presente sulla rete di

Telecom Italia arriva il 90 per cento delle richieste da parte di utenti Vodafone posso istallare anche un server sulla rete Vodafone. Così facendo si moltiplica il front end per ogni rete da cui provengono la maggior parte dei clienti .Ora vi è il problema di come indirizzare i clienti al front-end giusto. Ci si basa sulla lingua/dominio o sul routing (es. DNS modificato). Il DNS modificato è stato introdotto dall'azienda Akamai e con questo procedimento l'azienda non si limita a dare solo la traduzione nomi-indirizzi ma ricerca il server più vicino al cliente. Quando l'utente emette una richiesta http, il front end Web prende alcuni dei dati dalle pagine statiche e chiede la parte dinamica al component tier, il quale attiverà i suoi oggetti buisness per i calcoli. Per procedere con il calcoli però è necessario avere dati più precisi e quindi c'è un ulteriore richiesta, sottoinsieme delle prime. Dal livello data tier estrapolo i dati grezzi che vengono usati per fare i calcoli, diventano cooked data e infine verranno incorporati nella pagina html perché il browser comprende solo html. La richiesta diventa sempre più raffinata, un sottoinsieme delle precedenti, mentre la risposta aumenta di dimensione.

2.4 Client tier: browser o applicazione?

Andando a considerare il livello del client possiamo identificare due possibili scelte: browser o applicazione. Fare un'applicazione non è più un'eresia; esiste, per esempio un programma che gira su un'applicazione che cerca tutte le occorrenze in cui si è citato un'opera. Per entrambe le scelte si elencano vantaggi e svantaggi.

Web browser:

- (V) è già noto agli utenti e viene gestito da essi;
- (V) la trasmissione e la comunicazione dei dati è standard (la richiesta viene effettuata con http e i dati vengono trasferiti in html);
- (S) sia di http che di html esistono varie versioni. Qual è la versione compresa dal client? Se creo delle pagine in html 5 metà degli utenti non possono leggere queste pagine perché la versione 5 di html non è ancora supportata da tutti i browser. Inoltre per l'http ho due versioni 1.0 e 1.1;
- (S) Il browser è un interprete e quando riceve il codice html lo legge e cerca di capire cosa deve fare: se vede scritto p di paragrafo, va a capo e incomincia a scrivere del testo. Vi è differenza tra linguaggi compilati e linguaggi interpretati: un linguaggio compilato è tradotto una sola volta in linguaggio eseguibile(che è molto veloce), mentre un linguaggio interpretato deve essere letto tutte le volte. Quindi il browser è un sistema lento perché va ad interpretare il file (la pagina si compone poco per volta).;
- (S) funzionalità limitata in quanto è caratterizzato da una semplice interfaccia grafica;
- (S) per migliorare la grafica vi sono estensioni della pagina Web create con altre tecnologia come applet (scitti in Java), script client-side (pezzi di codice mandati al client), plugin(flash). Aggiungo funzionalità però restringo il numero di utenti che lo possono utilizzare.

Applicazione client custom/ad-hoc:

- (V) la funzionalità è molto ricca poiché l'interfaccia è creata sulle mie necessità;

- (V) le prestazioni sono molto elevate poiché si tratterà di un programma compilato, è quindi codice eseguibile per il mio computer;
- (S) addestramento degli utenti all'uso dell'applicazione;
- (S) scegliere su quali piattaforme, su quale hardware e sistema operativo questo programma è disponibile;
- (S) occuparsi dell'aggiornamento e del deployment;
- (S) assistenza utenti.

2.5 Architettura peer-to-peer

Il peer to peer è un'architettura in cui ciascuno può svolgere simultaneamente o in tempi diversi la funzione di client e server. Con questo tipo di scelta si distribuisce il carico di lavoro e il carico di comunicazione. Teoricamente in un'architettura peer to peer tutti dialogano con tutti distribuendo il carico, ma la rete fisica può essere diversa. Infatti se più peer sono attaccati alla medesima centrale ADSL, questa sarà il collo di bottiglia perché non vi è nessun collegamento diretto. I reali vantaggi del peer to peer,quindi, dipendono da quella che è la rete sottostante. Gli svantaggi di questo tipo di rete è che vi è difficoltà di coordinazione e di controllo e non sappiamo se il carico è davvero distribuito in quanto vi posso essere dei colli di bottiglia nascosti.

In una rete P2P ogni attore si occupa di una parte di calcolo (collaborative computing), esempio le collaborazioni chiamate grid computing. Queste collaborazioni posso essere chiuse, se si conoscono a priori il numero dei partecipanti, o aperte dove ogni peer può entrare o uscire a suo piacimento. Se si utilizza un'architettura peer to peer in azienda nasce il problema della condivisione delle informazioni e quindi il problema della riservatezza: si è disposti a dare i dati aziendali a computer sparsi nel mondo che potrebbero leggerli e conoscerli? Inoltre se ad uno dei nodi è stato assegnato un certo lavoro, si è certi che verrà terminato entro la data stabilita? Tutte queste domande pongono dei dubbi su questo tipo di struttura. In ragione di ciò questo tipo di applicazione viene usata per creare gli eadge service. Gli edge service sono reti che servono a trasportare informazioni fino ad un certo limite(edge). Un esempio è Skype: si possono fare telefonate dirette via IP (peer to peer) tra tutti quelli che hanno Skype ma sono possibili anche telefonate a telefoni fissi o cellulari; questo perché Skype è anche un eadge service(in questo caso arrivo fino al server Skype più vicino al posto dove lui si trova in quel momento e poi Skype sfrutta una rete telefonica locale).

2.6 Modelli di server

Il server gioca un ruolo importante nella rete perché da esso dipendono le prestazioni del sistema, quindi, è necessario individuare il modello migliore per le nostre esigenze. Non esiste una soluzione che vada bene per ogni problema applicativo e se si cercasse di individuarla essa rischierebbe di essere troppo complessa e costosa.

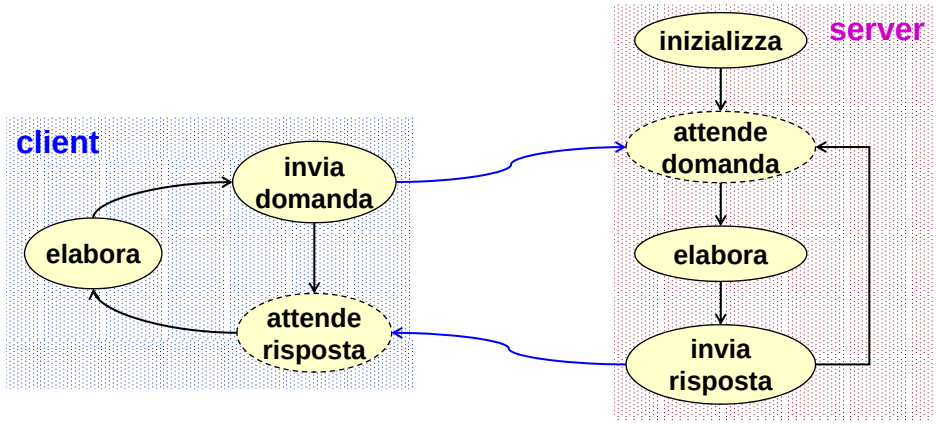

Figura 2.13: Server iterativo.

2.6.1 Server iterativo

Nel server iterativo (figura 2.13) il client invia delle richieste al server. Inviata la richiesta il client si pone in attesa della risposta. Il server, inizialmente avviato con boot (inizializzazione), è in una fase di attesa. Quando riceve una richiesta dal client si sblocca, procede a fare tutti i calcoli che sono necessari, fornisce la risposta e ritorna nello stato di attesa. Il client solo quando riceve la risposta si rimette a lavoro finché non interrogherà nuovamente il server. Qualora arrivino al server due richieste da due client simultaneamente, esso prenderà in esame la richiesta che arriverà per prima. Per tale motivo questo modello viene detto iterativo (il server elabora una richiesta per volta). Solo quando è terminata la prima elaborazione verrà presa in considerazione la seconda.

Il server iterativo si usa tutte le volte che il ciclo per fornire una risposta è breve. Ad esempio sono tipicamente realizzati in modo iterativo i cosiddetti TCP/IP *small services* tra cui:

- *daytime* – collegandosi alla porta 13 TCP o UDP di un server, si ottiene data e ora;
- *qotd* (Quote Of The Day) – collegandosi alla porta 17 TCP o UDP si ottiene una frase celebre o un motto di spirito;
- *time* – collegandosi alla porta 37 TCP o UDP si ottiene data e ora in formato coordinated universal time.

Si noti che in tutti questi casi il server chiude il collegamento non appena ha fornito la risposta e può quindi passare subito a servire un altro client.

In generale si adotta uno schema iterativo quando non si vuole sovraccaricare il server. Su un computer, tutti i processi attivi condividono le stesse risorse e l'assegnazione di queste risorse è opera dello schedulatore che è anch'esso un programma. Se il numero di processi è troppo elevato rispetto alle risorse disponibili sul sitema allora si verifica il cosiddetto trash, uno stato in cui nessun processo riesce ad eseguire le sue operazioni poiché tutto il tempo viene consumato dallo schedulatore per gestire l'ordine di esecuzione dei vari processi. Se si limita il numero di processi simultanei, Nel caso di un server iterativo vi è un solo processo per volta, tutte le risorse, tra cui la CPU, saranno sempre dedicate a questo unico processo che potrà quindi essere svolto alla massima velocità.

I vantaggi sono la facilità di programmazione e la velocità di risposta se ci si riesce a collegare per primo. Ma se un client arriva per quinto allora deve aspettare e quando toccherà

ad esso, l'elaborazione della richiesta sarà velocissima. Bisogna quindi distinguere il tempo di elaborazione dal tempo di attesa dovuto alla coda dei client che stanno richiedendo un servizio.

Lo svantaggio è il carico estremamente limitato, massimo un utente per volta.

Le prestazioni di un server iterativo non sono influenzate dal numero di CPU perché il server elabora una richiesta alla volta. Indichiamo con T_e il tempo di elaborazione di una richiesta (tempo per leggere la domanda, fare i calcoli ed inviare sulla rete la risposta) e supponiamo che T_e sia molto maggiore di T_r (tempo di trasmissione in rete). Ciò è quasi sempre vero. Le prestazioni massime in condizioni ottimali sono:

$$P = \frac{1}{T_e}\text{servizi/s}$$

Nel caso di richieste simultanee da parte di più client, quelli che non vengono serviti subito rientrano in competizione (provano a rifare la richiesta più tardi) perché non esiste una coda, a meno che essa abbia ampiezza maggiore di 1 ma di solito non è così.
La latenza, misurata in secondi, è il tempo che intercorre tra l'istante in cui il client fa la domanda e l'istante in cui riceve la risposta. Il tempo di latenza sarà uguale al tempo di elaborazione se il client è il primo ad essere servito, ma se non è il primo bisognerà attendere che il server si liberi. Quindi il tempo di latenza sarà maggiore o uguale al tempo di elaborazione:

$$T_e \leq L \leq T_e \cdot W$$

dove W è uguale al numero di clienti simultanei. Il tempo di latenza medio sarà uguale al tempo di elaborazione moltiplicato per il valor medio di W:

$$E(L) = T_e \cdot E(W)$$

2.6.2 Esercizio (calcolo delle prestazioni per un server iterativo)

Tema d'esame del 10 giugno 2009:

> Un server web di tipo iterativo è installato su un computer con CPU a 2 GHz, 4 GB di RAM, un disco da 500 GB con tempo di accesso di 10 ms ed una scheda di rete a 10 Mbps. Calcolare il massimo throughput (misurato in servizi/s) del server trascurando la dimensione delle richieste HTTP e sapendo che in media ogni risposta ha una dimensione di 10 MB e per generarla il server deve effettuare 10 letture da disco ed eseguire 100000 istruzioni macchina.

Si calcola il tempo necessario per eseguire 100000 istruzioni macchina. La CPU lavora a 2 GHz, quindi svolge 2×10^9 istruzioni/s. Dovendo svolgere 100000 istruzioni, allora il tempo di CPU è pari a:

$$T_c = \frac{100000}{2 \cdot 10^9} = 0.05\,\text{ms}$$

Per generare la risposta si devono leggere i dati dal disco (sul quale non si danno informazioni riguardo la frammentazione, throughput). Per leggere dal disco si impiegano 0.01 s e poiché si devono fare 10 letture, avrò che il tempo per leggere i dati è pari a:

$$T_d = 10 \cdot 0.01 = 0.1\,\text{s}$$

Ora si deve calcolare il tempo di rete, ricordando che il tempo della domanda è trascurabile. Il tempo della rete per la risposta, indicato con T_r, è pari alla dimensione della risposta, 80 Mbit, diviso la velocità della rete, 10 Mbps. Quindi si ha:

$$\frac{80}{10} = 8\,\text{s}$$

Quindi il tempo totale per l'elaborazione della risposta – dato dalla somma dei tre tempi calcolati – è pari a:

$$T_e = T_c + T_d + T_r = 8.15\,\text{s/servizio}$$

Il valore delle prestazioni è dato dal reciproco di T_e, quindi:

$$P = \frac{1}{8.15} = 0.12\,\text{servizi/s}$$

Il risultato indica che il servizio non è ottimale poiché le prestazioni sono basse e per migliorare il sistema si possono effettuare varie modifiche. Dato che il collo di bottiglia è il tempo di rete, pari a 8 secondi, si può istallare una scheda di rete più veloce. Il problema però non è risolto in quanto le prestazioni sopra trovate sono da considerarsi nel caso ottimale. Può succedere che, pur avendo una scheda di rete velocissima, le prestazioni non migliorino. Questo accade perché il problema è la rete tra un determinato client e tutti gli altri utenti e non è detto che la stessa velocità viene mantenuta fino al client. Quindi un altro modo per migliorare le prestazione sta nel diminuire i dati da trasmettere attraverso la compressione. Questo aumenta i tempi di CPU ma riduce notevolmente il tempo di trasmissione dei dati. Come terza soluzione posso inserire un buffer di rete per avere trasmissione asincrona. Ogni volta che il server, che è in attesa, riceve una richiesta dal client, esso legge i dati dal disco e fa i calcoli mediante la CPU e rimanda indietro la risposta tramite la rete. Dato che il tempo di trasmissione è molto ampio, sarebbe meglio avere la possibilità di assegnare il compito di mandare i dati a terzi mentre il server può tornare a elaborare una successiva richiesta. Ma come si può implementare ciò? Esistono schede di rete che contengono un buffer in cui vengono immagazzinati i dati da mandare. Così facendo delego al buffer il compito di mandare i dati e il server può continuare il suo lavoro. Questo modo di procedere prende il nome di trasmissione asincrona. Se trasferisco dati da 10 MB e ho un buffer da 32 MB posso solo accodare 3 richieste. Il buffer mi server quando il carico non è uniforme perché assorbe le differenze di carico quando ci sono dei picchi.

2.6.3 Server concorrente

Per gestire diversi clienti contemporaneamente si usa un server di tipo concorrente (figura 2.14). I processi che riguardano il lato client rimangono invariati rispetto al modello iterativo. Quando il server, che è stato inizializzato, riceve una richiesta genera un figlio. Quindi il server padre riceve le richieste e le smista ai server figli che sono uguali al padre e vengono creati a seconda delle necessità. Il figlio elabora la risposta, la manda al padre e muore. Il padre quando riceve la risposta dal figlio la invia al client. Questo tipo di server si chiama concorrente perché nel caso in cui arrivino più clienti simultaneamente, il padre genera più figli, i quali entreranno in competizione sulle stesse risorse.

I servizi gestiti da un server concorrente sono servizi di cui non so quanto dura il lavoro di elaborazione. Esempi di server concorrente sono la maggior parte di servizi standard TCP/IP:

- *echo* – (porta 7 TCP/UDP) misuro il tempo di andata e ritorno di una richiesta (si mandano dei dati al server ed esso li rimanda indietro);

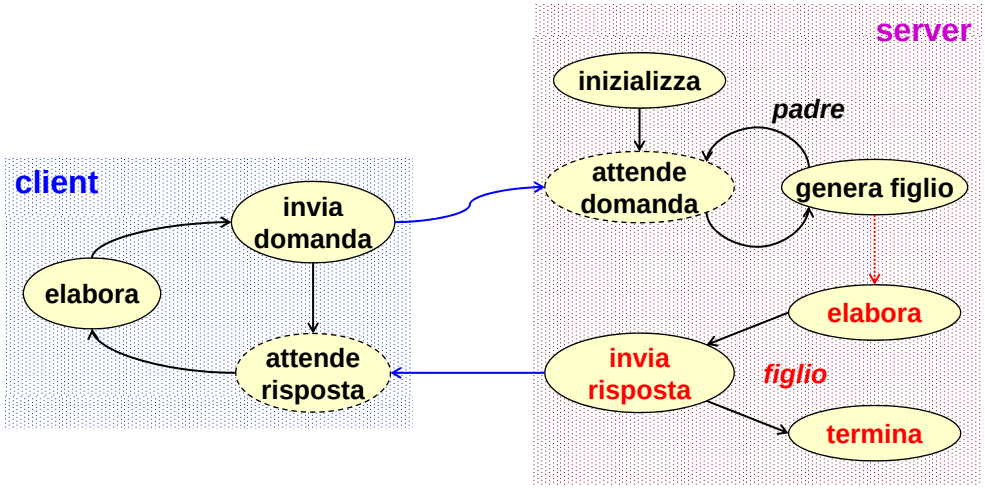

Figura 2.14: Server concorrente.

- *discard* – (porta 9 TCP/UDP) misuro i tempi di solo andata di una richiesta (si mandano dei dati al server ed esso li elimina);
- *chargen* – (porta 19 TCP/UDP) misuro solo i tempi di ritorno di una richiesta (ci si collega ad un server ed esso ci manda dati a raffica);
- *telnet* – (porta 23 TCP) collegamento interattivo con il server (il tempo di durata del servizio non è conosciuto a priori);
- *SMTP* (porta 25 TCP) invio di un e-mail (il tempo di trasmissione dipende dalla dimensione del messaggio).

Generalmente si usa un server concorrente tutte le volte in cui il servizio è di lunga durata e lo si vuole fornire a più utenti simultaneamente oppure la durata di un servizio non è prevedibile a priori. Possiamo effettuare un'analisi sui vantaggi e svantaggi di questo modello di server.

Vantaggi:

- teoricamente il carico è limitato (aumento il numero di figli a seconda del carico delle richieste ma aumentando il numero di figli si riduce la parte di risorsa che verrà usata da ognuno).

Svantaggi:

- complessità di programmazione (scrivere programma di questo tipo è complicato perché vi è competizione per le risorse);
- lentezza di risposta (quando arriva la domanda si deve creare un figlio e questo tempo è significativo per le prestazioni del sistema);
- il carico massimo reale è limitato perché ogni figlio richiede una parte delle risorse del sistema(RAM, CPU, accesso al disco).

Le prestazione del server concorrente sono influenzate dal numero di CPU in quanto si possono elaborare più richieste simultaneamente. Identifichiamo il tempo di creazione di un figlio con T_f misurato in secondi. Le prestazioni massime in condizioni ottimali sono:

$$P = \frac{C}{T_f + T_e}[servizi/s]$$

Anche in questo caso andiamo a calcolare il tempo di latenza ed esso è uguale o maggiore della somma del tempo di elaborazione più il tempo di creazione del figlio.

$$(T_f + T_e) \leq L \leq (T_f + T_e)\frac{W}{C}$$

dove W è il numero di client simultanei e C il numero di CPU.

2.6.4 Esercizio (calcolo prestazioni per un server concorrente)

Tema d'esame del 4 settembre 2009:

> Un server web di tipo concorrente è installato su un computer con due CPU a 1 GHz, 2 GB di RAM, un disco da 250 GB con tempo di accesso di 20 ms ed una scheda di rete a 100 Mbps. Calcolare il massimo throughput (misurato in servizi/s) del server trascurando la dimensione delle richieste HTTP e sapendo che in media ogni risposta ha una dimensione di 10 MB e per generarla il server deve effettuare 100 letture da disco ed eseguire 100000 istruzioni macchina.

Si calcola il tempo della CPU dividendo il numero di istruzioni macchina (100000) per la velocità della CPU (10^9), ottenendo:

$$T_c = \frac{10^5}{10^9} = 10^{-4}\,\text{s} = 0.1\,\text{ms}$$

Per calcolare il tempo totale di lettura dal disco moltiplicare il numero delle letture per il tempo necessario per fare una lettura; quindi:

$$T_d = 100 \cdot 0.02\,\text{s} = 2\,\text{s}$$

Per calcolare il tempo di rete, si divide la dimensione dei dati per la velocità di rete e si trova:

$$T_r = \frac{80}{100} = 0.8\,\text{s}$$

Quindi, trascurando il tempo per generare un figlio T_f, il tempo di elaborazione è pari alla somma dei tre tempi calcolati:

$$T_e = T_c + T_d + T_r \approx 2.8\,\text{s/servizio}$$

Avendo due CPU le prestazioni saranno:

$$P = \frac{2}{2.8} = 0.71\,\text{servizi/s}$$

In questo contesto il collo di bottiglia è il disco, poiché ha il tempo maggiore. Posso quindi comprare un disco più veloce che impieghi meno tempo oppure posso considerare la chace/-buffer del disco che, leggendo più dati alla volta, migliora la velocità. Inoltre posso diminuire la frammentazione del disco o usare particolari dischi, i RAID2 , dove i dati sono divisi tra i due dischi e posso leggere i dati simultaneamente, aumentando il doppio della velocità.

Le prestazioni sopra descritte si riferiscono al caso ottimale. Se si studia tutto il processo, si nota che la CPU è occupata solo durante la fase di elaborazione e solo minimamente durante la lettura del disco e la scrittura in rete. Per meglio comprendere quanto detto,si quardi la figura 2.15 Quando si ricevono due richieste da due client diversi, le due richieste

slide numero 47 del gruppo di slide -architettura dei sistemi distribuiti-

Figura 2.15: Esempio 4 Settembre 2009.

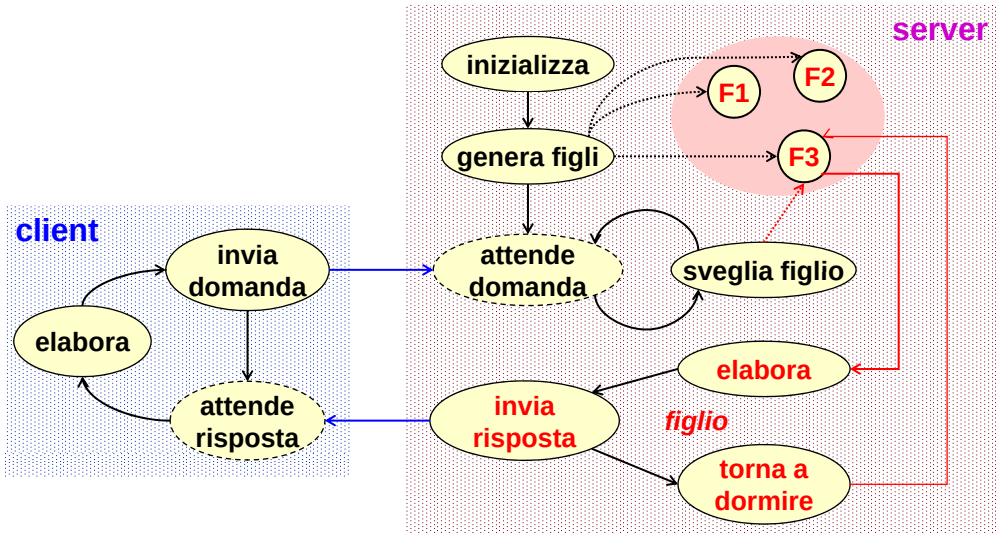

Figura 2.16: Server a crew

vengono elaborate dalle due CPU; ma l'accesso al disco non può essere fatto simultaneamente e supponendo che il figlio 1 stia leggendo da disco, l'altro è in una fase di attesa (idle). Quando il figlio 1 finisce, libera il disco che viene usato dal figlio 2 ma occupa la rete. Quando il figlio 1 finisce di elaborare la sua richiesta, ne riceve un'altra e quando andrà a leggere da disco, lo troverà occupato dal figlio e quindi deve aspettare. A seconda di quanto durano i diversi tempi, le sovrapposizioni sono differenti. Se consideriamo che il tempo di elaborazione è pari a 0.1 ms, il tempo di lettura da disco è pari a 2 s e il tempo di scrittura in rete è pari a 0.8 s, si ottiene un tempo di attesa pari a 1.2 s. Questo vuol dire che il tempo di elaborazione, che era pari a 2.8 secondi/servizio, è da aumentare di 1.2 secondi (tempo di attesa). Il tempo di attesa è dovuto al fatto che alcune risorse, come il disco o la rete, non sono duplicate. Le prestazioni in questo contesto peggiorano rispetto a quanto calcolato precedentemente e sono pari a 30 servizi/minuto.

2.6.5 Server a "crew"

Nei server concorrenti ci sono due problemi: il tempo per creare un figlio e la condivisione delle risorse tra un numero di figli non conosciuto a priori. La soluzione è il server a crew. Nel modello a crew (figura 2.16) dopo la fase di inizializzazione si crea un certo numero di figli che si pongono in attesa. Se si creano tre figli si ipotizza che ci siano almeno tre client simultanei. Il padre attende la richiesta del client; nel momento in cui riceve la richiesta, sveglia un figlio (tempo breve) che deve occuparsi della sua elaborazione. Il figlio terminato il suo lavoro manda la risposta al padre e torna nello stato di attesa. Il padre invia la risposta al client. Annullo così il tempo di creazione del figlio e sono certo che al massimo si ha un determinato numero di figlio che condividono una risorsa limitata. Il numero di figli, quindi, è commisurato con le risorse del sistema.

Esempi di server a crew sono:

- tutti i server concorrenti possono essere sostituiti con server a crew perché si tratta solo di una diversa fase di inizializzazione;

- tipicamente adatto per tutti i server di rete ad alta prestazione (server sottoposti ad alto carico oppure server in cui il ritardo di risposta non è accettabile poiché si vuole ridurre la latenza);
- esempi: servizi Web per il e-commerce, server di database perché la velocità dell'accesso ai dati è fondamentale.

Possiamo effettuare un'analisi sui vantaggi e svantaggi di questo modello di server.

Vantaggi:

- il carico è idealmente illimitato (si possono creare figli addizionali in funzione del carico che moriranno a lavoro ultimato) ;
- velocità di risposta (il tempo per svegliare un figlio è minore di quello per crearlo);
- possibilità di limitare il carico massimo (solo ai figli che sono stati pre-generati).

Svantaggi:

- complessità di programmazione ancora di tipo concorrente;
- la gestione dell'insieme dei figli: conoscere chi è libero e chi è occupato (si parla di children-pool: i figli sono in attesa di ricevere richieste dal padre);
- sincronizzazione e concorrenza degli accessi alle risorse condivise.

Il calcolo delle prestazioni massime non cambia rispetto al server concorrente se non per il fatto che T_f viene sostituito con T_a, cioè il tempo necessario per attivare un figlio (spesso trascurabile rispetto agli altri tempi in gioco):

$$P = \frac{C}{T_a + T_e}$$

Se il sistema può generare altri figli allora biosgna effettuare la media pesata tra il caso in cui si attiva semeplicemente un figlio e quello in cui lo si crea. Indicando con G la probabilità di dover generare nuovi figli (e quindi con $1 - G$ la probabilità di dover attivare un figlio già creato) le prestazioni massime sono data da:

$$P = (1 - G) \cdot \frac{C}{T_a + T_e} + G \cdot \frac{C}{T_f + T_e}$$

2.7 Programmazione concorrente

Nell'implementazione di server concorrenti o a crew possiamo avere due tipi di modelli di programmazione: a processi o thread (figura 2.17). Nel modello a processi, ogni processo (codice eseguibile che è stato mandato in esecuzione) è indipendente dagli altri e presenta il proprio codice eseguibile, il proprio program counter (PC che indica la prossima istruzione da eseguire) e la sua zona di memoria RAM. Se ho due processi avrò due PC, due codici e due zone di memorie dedicate ad ogni processo. Nel modello a thread non abbiamo processi diversi ma, un determinato processo si suddivide in due o più thread (linea di esecuzione). In un sistema multicore ogni thread viene eseguito su una differente core e ogni thread ha il proprio codice e PC e può lavorare su tutta la memoria assegnata al processo.

slide numero 52 del gruppo di slide -architettura dei sistemi distribuiti-

Figura 2.17: Programmazione concorrente: a processi o a thread.

- attivazione di un modulo:
  - (P) lenta perché per creare un processo è necessario del tempo;
  - (T) veloce in quanto il thread si trova all'interno della stessa zona del processo padre.

  Quindi dal punto di vista della velocità si preferisce un modello a thread.
- comunicazione tra moduli:
  - (P) difficile perché ogni modulo ha la propria RAM e per comunicare si ha bisogno di servizi specifici che prendono il nome di IPC da richiedere al sistema operativo (se ho due processi che vogliono comunicare, il processo P1 dialoga con il sistema operativo che riferirà al processo P2 e viceversa);
  - (T) facile perché i vari moduli lavorano sulla stessa zona di memoria.

  Quindi anche la comunicazione tra moduli è a favore dei thread.
- protezione tra moduli:
  - (P) ottima, ogni processo può commettere errori solo nella sua parte di memoria e relativi al solo lavoro che svolge senza attaccare gli altri processi;
  - (T) pessima, i vari processi lavorano sulla stesso parte di memoria e quindi se un processo genera degli errori, questi danneggiano tutti i vari thread attivi e compromettono l'intero processo. Ogni thread per scrivere nella zona di memoria comune deve adottare delle tecniche di sincronizzazione che stabiliscono l'ordine di scrittura sulla memoria. Se non si usano queste tecniche si può incorrere nel cosiddetto "deadlock"(abbraccio mortale).

  Dal punto di vista della protezione si preferisce il modello a processi
- debug:
  - (P) non banale ma possibile (si devono analizzare due o più processi simultaneamente e quindi ho bisogno di una o più finestre che mostrino ciò);
  - (T) molto difficile perché ho PC locali ed è difficile decidere l'ordine di esecuzione delle varie operazioni (può accadere che il processo funzioni solo utilizzando il debuger. Questo succede perchè il debug sta alterando l'ordine di esecuzione delle istruzioni.);

  Quindi se si vuole lavorare con il debug si preferisce avere un modello a processi.

In conclusione usiamo un modello a thread quando è importante la velocità (sistemi utente come Windows). Invece usiamo un modello a processi quando è importante la solidità del sistema (sistema ferroviario).

Capitolo 3

Programmazione in ambiente web

3.1 Il World Wide Web (WWW)

Il World Wide Web, spesso abbreviato semplicemente come "web", è un insieme di protocolli applicativi e formati dati appoggiato su canali di tipo TCP/IP. I protocolli di comunicazione sono necessari a permettere lo scambio d'informazioni tra client e server e server diversi tra loro.

Come illustrato in figura 3.1, a livello rete il web sfrutta il protocollo IP mentre a livello trasporto usa il protocollo TCP e a livello applicazione troviamo indistintamente HTTP, FTP o altri protocolli idonei alla trasmissione delle informazioni. Inoltre possiamo osservare alcuni formati dati compatibili con il web come CSS, PNG, JS, HTML, XHTML.

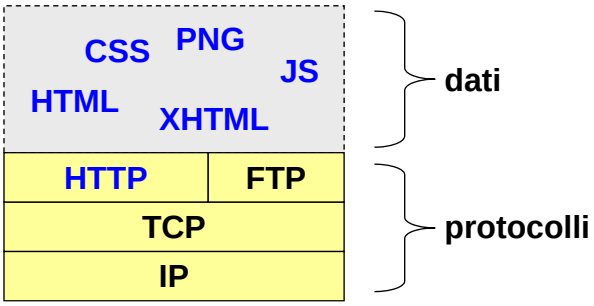

Figura 3.1: Struttura del web.

Per quanto riguarda la trasmissione dell'informazione nel web, non ha molta importanza il protocollo di trasporto implementato, l'importante è che questo sia in grado di trasportare i dati. Quindi, si possono utilizzare diversi protocolli applicativi. Ci sono però delle limitazioni a seconda del tipo di protocollo usato, infatti, nel web le funzionalità ottenibili sono dettate dal protocollo applicativo e quindi sono limitate a quelle disponibili per un certo tipo di protocollo.

I protocolli come FTP utilizzati inizialmente, non erano stati concepiti direttamente per il web e per questo rappresentavano limitazioni e complicazioni. FTP si limita semplicemente alle funzionalità di GET e PUT di un file.

Alcuni tipi di protocolli già esistenti non comportano semplici limitazioni di funzionalità ma altre complicazioni come la lentezza nella risposta e problemi nella visualizzazione di parti di una pagina o dell'intera pagina. Per questo motivo fu definito un nuovo protocollo applicativo, HTTP, che è oggi il più utilizzato nel web proprio perché pensato direttamente

per questo e arricchito di funzionalità utili al trasporto dei dati. Una delle caratteristiche per cui HTTP differisce dagli altri protocolli applicativi è quella di chiudere le connessioni una volta soddisfatta una richiesta o una serie di richieste. Ciò rende il protocollo particolarmente indicato per il web, in cui le pagine molto spesso contengono dei collegamenti a pagine ospitate da altri server.

3.2 Il web statico

Un'architettura web statica è definita da un modello client-server a due livelli (two-tier), in cui il client è il browser ed ha la funzione di interpretare e visualizzare la pagina web, mentre il server è costituito dal server HTTP (spesso indicato come HTTPD, dove la D sta per Daemon o Demone[1]). Nel modello più semplice si ipotizza che la risorsa richiesta dal client sia un file.

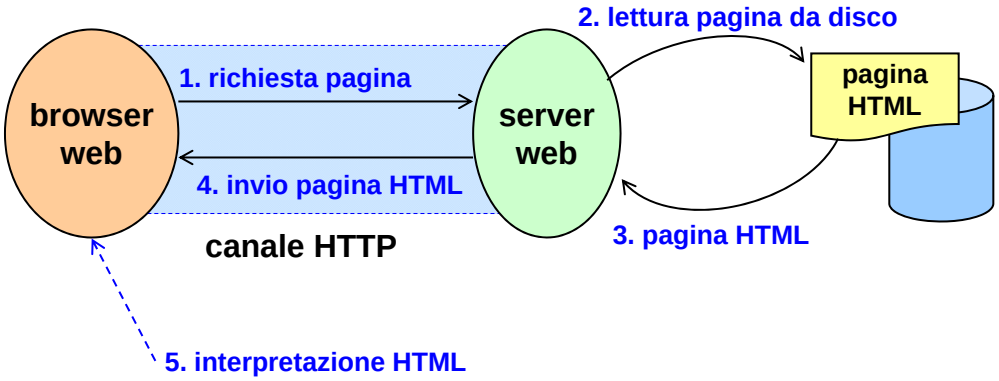

Figura 3.2: Modello di architettura web statica.

La procedura di richiesta e ottenimento della risorsa è quella di cui abbiamo un esempio nella figura 3.2. Per rendere possibile la comunicazione viene creato un canale HTTP tra browser web (client) e server web. Dopodiché:

1. il browser web richiede la pagina al server;
2. la richiesta arriva al server web che legge la pagina dal disco;
3. ; il server web ottiene la pagina dal disco
4. la pagina viene inviata al browser web attraverso il canale HTTP;
5. il browser interpreta il file HTML e lo visualizza sullo schermo.

In un contesto di web statico le pagine web non cambiano mai il loro contenuto finché l'autore non le modifica esplicitamente. Quindi, il contenuto delle pagine non dipende in nessun modo dall'interazione con l'utente e nemmeno dalle informazioni inviate al server dal client o dall'istante di tempo in cui è effettuata la richiesta. La pagina è implementata in HTML/CSS e ad ogni pagina web corrisponde un unico file HTML.

[1]I demoni sono i processi server del sistema perativo Unix, ambiente in cui è stato originariamente sviluppato il web.

3.2.1 Web statico: vantaggi e svantaggi

I principali vantaggi di un'architettura web statica sono:

massima efficienza, infatti il server non deve fare alcun calcolo ed il lato client non ha molto lavoro da fare, semplicemente interpretare HTML e visualizzare la pagina;

possibilità di fare caching delle pagine, perché una pagina richiesta per la prima volta da un client può essere temporaneamente memorizzata in RAM sul server oppure sul disco del client o di un proxy, così da minimizzare i tempi di trasferimento quando la stessa pagina viene nuovamente richiesta dallo stesso o da altri client;

pagine indicizzabili dai motori di ricerca, infatti essendo il loro contenuto statico è facilmente leggibile ed indicizzabile.

Per quanto riguarda il caching delle pagine, si noti che esiste un trade-off: più la pagina è conservata in una cache vicina al client e più sarà trasferita velocemente, ma sarà accessibile ad un numero minore di client. Per questo motivo spesso una pagina è conservata in diverse cache.

Invece i principali svantaggi di un'architettura web statica sono:

staticità dei dati, il che la rende adatta solo per fornire informazioni che cambiano raramente o mai;

nessuna adattabilità ai client e alle loro capacità, infatti essendo i dati statici essi vengono forniti sempre uguali a qualunque client, indipendemente dalle sue caratteristiche o capacità.

3.2.2 Richiesta di una pagina statica

Quando viene richiesta una pagina statica, l'indirizzo di questa viene suddiviso in vari campi, ognuno dei quali ha un preciso compito all'interno del web. Ad esempio, se si effettua la richiesta della pagina principale di questo corso

`http://security.polito/~lioy/01eny/}`

il server HTTP interpreta questa URL nel modo seguente:

`http://`
: identifica il protocollo applicativo con cui si desidera che sia trasportata la risorsa (fornisce implicitamente anche il numero della porta, 80, ed il protocollo di trasporto, TCP);

`security.polito.it`
: è il nome DNS del server su cui si trova la risorsa, associato a un determinato indirizzo IP (130.192.1.8);

`/~lioy/01eny/`
: identifica la risorsa richiesta all'interno dello spazio disco del server dedicata alle pagine web.

Questa procedura prende il nome di "mapping", cioè l'associazione tra il nome logico della risorsa e quello con cui è stata memorizzata nel server. Il server effettua un "rewriting" (riscrittura) della URL richiesta dal client grazie alla quale riesce a risalire alla risorsa memorizzata e a inviarla al client. Nel nostro esempio la URL effettiva che il server utilizza è

`/u/lioy/public_html/01eny/index.html`.

Risale quindi alla risorsa memorizzata in una specifica cartella del disco e precisamente al file `index.html`. Questo perché al termine della URL è stato specificato il carattere"/" che indica il file di default, su un server Apache[2].

3.2.3 Modello delle prestazioni nel web statico

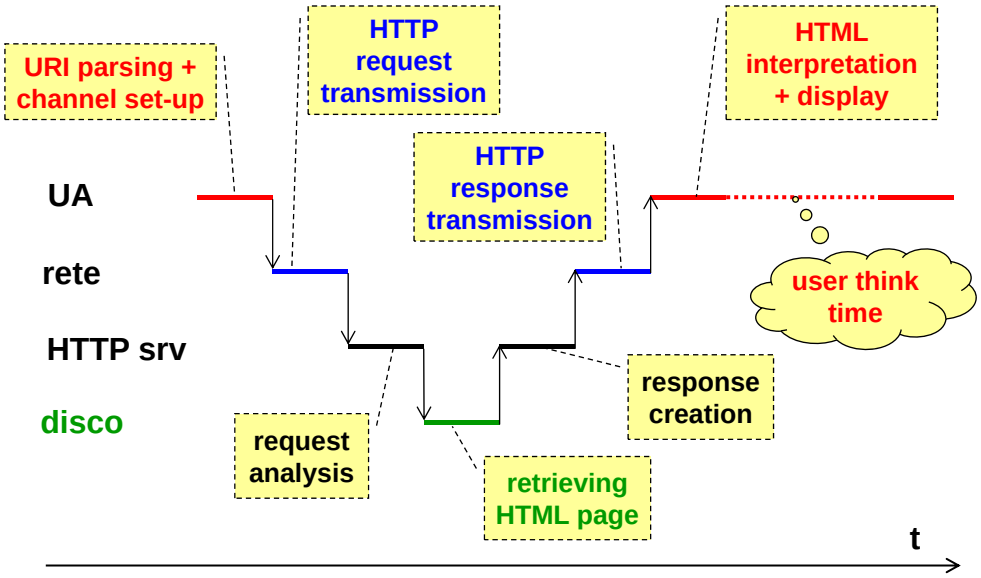

Figura 3.3: Prestazioni in web statico.

Nella figura 3.3 è mostrato l'andamento dei diversi processi a vari livelli.

Lo User Agent (UA), ovvero il browser, crea il canale HTTP per mezzo del quale client e server possono comunicare e interpreta l'indirizzo della pagina. In seguito nella rete avviene la trasmissione della richiesta, tramite il canale instaurato, e il server HTTP che riceve la richiesta, la analizza e inizia la ricerca nel disco. Il disco recupera la pagina richiesta dal server e gliela invia. Questo, dopo averla ricevuta, genera una risposta che percorre a ritroso il canale HTTP. Una volta che la risorsa giunge al client, il browser web interpreta la pagina e la visualizza sullo schermo.

Nella figura 3.3, i segmenti colorati associati a ciascun protagonista della comunicazione possono avere lunghezze diverse, a seconda di quale sia il processo interessato. Nella rete, infatti, sono presenti processi lenti e processi più veloci. Ad esempio, sappiamo che il disco è molto lento. La velocità del processo di trasmissione tra client e server invece dipende dalla velocità della rete. Questa può variare a seconda dei casi.

[2]Apache – -- è uno dei server HTTP più usati a livello mondiale.

3.2.4 Agent, server, proxy e gateway

Come detto sopra, lo User Agent è il browser, ma questa funzione può anche essere svolta da spider, robot ed altri programmi eseguiti lato client.

L'Origin Server invece è il fornitore del servizio desiderato, cioè colui che possiede i dati originali e quindi è in grado di fornire una risposta alle richieste dello UA.

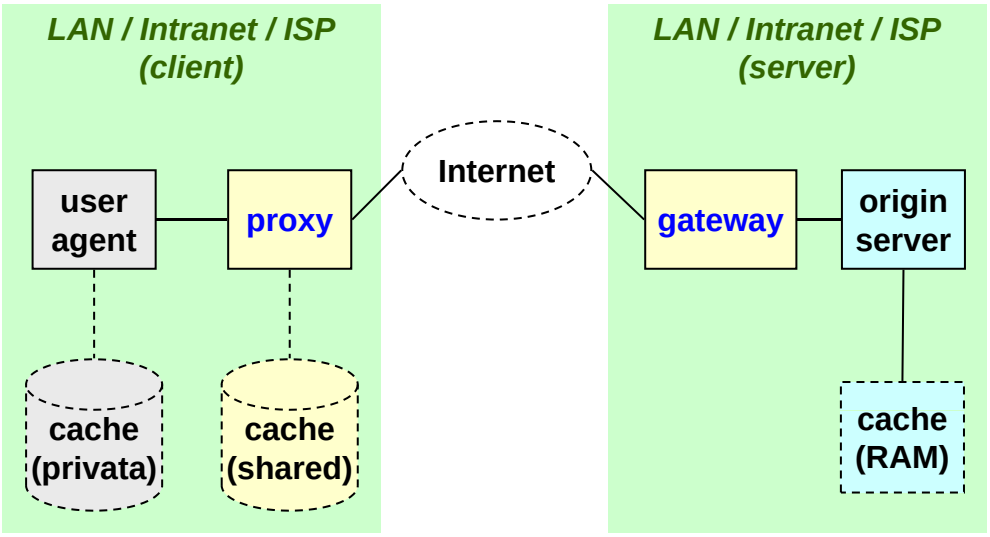

Figura 3.4: Schema di un canale HTTP con proxy e gateway.

Come illustrato nella figura 3.4 tra UA ed OS possono esistere altri elementi che spezzano la comunicazione HTTP. Questi elementi sono ifronti, ossia da un lato si comportano come un client e dall'altro come un server secondo la necessità (v. figura). Questi elementi sono il gateway (posto lato server) ed il proxy (posto lato client). Entrambi questi elementi possono avere una cache in cui memorizzare il contenuto delle pagine a loro richieste, per rendere più veloce il processo di risposta, nel caso in cui una stessa pagina sia richiesta una seconda volta.

Come possiamo vedere nella figura 3.4, la cache dello UA è privata, accessibile solo dal client sul quale è installata, quella del server si trova sulla RAM perché deve essere veloce nel dare le risposte (come abbiamo già detto la RAM è più veloce del disco), mentre le cache di gateway e proxy possono essere condivise da più utenti e, in genere, vengono memorizzate su disco perché la RAM non sarebbe sufficiente per tutti gli utenti.

gateway Funziona come un'interfaccia pubblica per il server. Ad esempio, viene usato per la sicurezza o il load balancing.

proxy Lavora per conto del client, si occupa di trasmettere la domanda al server o, nel caso in cui abbia in memoria la risorsa richiesta, risponde direttamente al client. Usato anche con funzione di autenticazione e autorizzazione.

3.2.5 Proxy

Questo elemento della rete mantiene memorizzate nella cache solo le pagine statiche. Infatti, le pagine dinamiche hanno come caratteristica quella di variare il loro contenuto in particolari situazioni, come, ad esempio, interazione con utenti, scadenza di un determinato periodo, etc.

E' necessario inserire il campo `<meta http-equiv="`*Expires*`" content="...">` nell'head di pagine dinamiche, in modo che sia chiara a tutti gli utenti, la scadenza di determinati contenuti. Si può anche ricorrere al seguente codice: `<meta http-equiv="`*cache-control*`" content="`*no-cache*`">`, che serve a segnalare che la pagina non può essere memorizzata nella cache perché varia di continuo e quindi un contenuto memorizzato in un certo momento non sarebbe più affidabile poco dopo.

E' possibile riscontrare non solo semplici proxy, ma anche gerarchie di questi. Alcuni esempi sono la rete del Politecnico (POLITO), IT, EU, etc.

Il proxy, inoltre, viene spesso usato da ISP per migliorare la velocità di navigazione dei client.

Funzionamento Il proxy può avere un funzionamento trasparente, cioè non altera la richiesta, tranne che per alcune parti obbligatorie, o non trasparente, cioè riscrivere la richiesta (es. anonymizer).

Configurazione su UA La configurazione sullo User Agent può essere esplicita, e quindi richiedere un intervento sul client, oppure implicita. In quest'ultimo caso è richiesta la presenza di intelligenza nella rete.

3.3 Web statico con pagine dinamiche

Il termine web dinamico viene utilizzato per indicare tutte quelle applicazioni Web che variano il proprio contenuto in relazione all'interazione con l'utente; tuttavia la dinamicità è solamente lato utente, dal punto di vista server non cambia niente. Questo tipo di programmazione si contrappone al Web statico che non permette all'utente di interagire attivamente con la pagina.

3.4 Vantaggi e svantaggi delle pagine dinamiche

Tra i principali vantaggi derivanti da questa tecnica di programmazione notiamo:

- maggiori funzionalità del nostro sito Web e quindi un migliore servizio offerto al cliente;
- buona efficienza lato server (basso carico di CPU).

Tuttavia si contrappongono degli svantaggi, tra i quali:

- maggiore inefficienza lato client (medio-alto carico di CPU, in funzione del tipo di dinamicità adottato);
- staticità dei dati;
- funzionalità della pagina variabili (dipendono dalle capacità del client).

Oltre questi parametri di giudizio, alcuni di essi risultanti positivi e altri meno, è bene considerare una serie di altri fattori che non incidono negativamente sul sistema ma in alcuni casi possono limitarne l'efficienza. Infatti il Web dinamico riduce la possibilità di fare caching, dato che le pagine devono essere interpretate, e inoltre la parte dinamica non è indicizzabile dai motori di ricerca che comprendono solo codice HTML. Tutti queste considerazioni devo essere tenute presenti nella realizzazione di questo tipo di pagine.

3.5 Metodi d'implementazione

Un modo per rendere possibile la realizzazione di pagine dinamiche è l'utilizzo di applet Java o componenti ActiveX. Entrambi sono dei programmi che possono essere eseguiti dai web browser; i primi richiedono una Java Virtual mMchine mentre i secondi il sistema operativo MS-windows ed un client con architettura hardware Intel x86.

Tuttavia questa tecnica di progettazione nasconde una serie di problemi riguardo:

- La compatibilità della JVM o della versione del sistema operativo;
- Il carico poiché richiedono l'esecuzione del software;
- La sicurezza dato che si esegue un programma vero e proprio. Le applet Java vengono eseguite in una sandbox che garantisce un minimo di controllo mentre activeX installa una DLL e il programma è eseguito senza nessuna garanzia.

Inoltre in termini di carico computazionale le applet Java risultano essere più pesanti perché necessitano che il bytecode venga interpretato.

Un'alternativa ad applet e componenti ActiveX lato cliente sono gli script, generalmente semplici programmi il cui scopo è l'interazione con altri programmi, molto più complessi, in cui avvengono le operazioni più significative. Gli script si distinguono dai programmi con cui interagiscono, solitamente implementati in un linguaggio differente e non interpretato. Gli script vengono generati mediante una serie di linguaggi di scripting. Tra questi compaiono JavaScript, il cui nome può trarre in inganno per la sua apparente derivazione da java (in realtà i due linguaggi non hanno quasi nulla in comune), JScript simile al precedente ma sviluppato da Microsoft e VBScript anch'esso prodotto da Microsoft, che lavora solo con il browser Internet Explorer.

3.6 Client-side scripting

L'uso di linguaggi di scripting come si è visto precedentente determina una serie di vantaggi in termini di carico e sicurezza rispetto all'applet. Il principale linguaggio di riferimento fu sviluppato da Nescape e da Sun Microsystems con il nome iniziale di LiveWire, in seguito rinominato "JavaScript". Dato il suo successo, Microsoft progettò un linguaggio compatibile, conosciuto come JScript. La necessità di specifiche comuni fu alla base dello standard ECMA 262 che sostanzialmente determina la fusione dei due linguaggi.

La caratteristica principale di JavaScript è quella di essere un linguaggio interpretato: il codice non viene quindi compilato.Altri aspetti di interesse sono che il codice viene eseguito direttamente sul client e non sul server. Il vantaggio di questo approccio è che, anche con la presenza di script particolarmente complessi, il server non viene sovraccaricato a causa delle richieste dei clients. Di contro, nel caso di script che presentino un sorgente particolarmente grande, il tempo per lo scaricamento può diventare abbastanza lungo. Un altro svantaggio è il seguente: ogni informazione che presuppone un accesso a dati memorizzati in un database remoto dev'essere rimandata ad un linguaggio che effettui esplicitamente la transazione, per poi restituire i risultati ad una o piu variabili JavaScript; operazioni del genere richiedono il caricamento della pagina stessa.

Con l'avvento di AJAX, acronimo di Asynchronous JavaScript and XML, molti di questi problemi sono stati risolti. Lo sviluppo di applicazioni HTML con AJAX si basa su uno

scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. In questo modo si riduce la quantità di dati scambiati e quindi il tempo di caricamento della pagina sul client.

Altra funzionalità del client side scripting è permettere di eseguire una funzione associata ad un evento scatenato dall'interazione con la pagina, ad esempio al completamento di un form validare i dati prima di trasmetterli risparmiando traffico inutile in rete e semplicando l'architettura server side. Tuttavia anche in presenza di queste funzionalità lato client è buona norma effettuare diversi controlli anche sul server.

3.6.1 Inserimento di script lato client

Per inserire uno script in una pagina HTML è necessario l'uso del tag `<script>`. Questo tag non è parte del linguaggio JavaScript in sé, ma serve solo come "contenitore" all'interno di una pagina HTML. All'interno del tag occorre definire come parametro obbligatorio il tipo di linguaggio usato tramite l'attributo `type`, con possibili valori `text/javascript`, `text/vbscript` ed altri linguaggi. Ad esempio:

```
<script type="text/javascript">
... script lato client ...
</script>
```

Inoltre è possibile usare l'attributo `src` per indicare la URI di un file esterno che contiene il codice dello script, come nel seguente esempio:

```
<script type="text/javascript" src="controlli.js">
</script>
```

Poiché non è noto se lo UA che riceve la pagina è in grado di interpretare lo script inviatogli (potrebbe non averne la capacità oppure l'utente potrebbe avere disabilitato l'esecuzione degli script lato client per motivi di sicurezza) è consigliato usare il tag `<noscript>` per fornire un contenuto alternativo che verrà visualizzato solo da quegli UA non in grado di interpretare lo script ricevuto. Ad esempio:

```
<noscript>
   Attenzione! questo sito usa JavaScript
   per l'aggiornamento dei dati in tempo reale.
   Il tuo browser non supporta JavaScript (oppure ne &egrave; stata
   disabilitata l'esecuzione) e quindi tale funzionalit&agrave;
   non sar&agrave; disponibile.
</noscript>
```

La figura 3.5 contiene un semplice esempio di uso di uno script lato client per visualizzare un saluto oppure l'informazione che lo UA non è ingrado di interpretare lo script.

La figura 3.6 contiene invece un esempio più complesso: lo script viene usato per generare dinamicamente lasto client la tavola dei quadrati mediante un ciclo for, senza dover scrivere manualmente tutti i dati come avverrebbe con una pagina statica.

Per una spiegazione più dettagliata del linguaggio JavaScript e degli script lato client si rimanda il lettore ai capitoli 6 e 7.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
     <title>Esempio 1 con JavaScript</title>
   </head>
   <body>
   <script type="text/javascript">
      <!--
      document.writeln("Ciao!");
      // -->
   </script>
   <noscript>
   AARGH! il tuo browser non supporta JavaScript o &egrave; disabilitato.
   </noscript>
   </body>
</html>
```

Figura 3.5: Un semplice esempio di script lato client.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd"><html>
<html>
   <head><title>Tavola dei quadrati</title></head>
   <body>
      <h1>Tavola dei quadrati</h1>
      <script type="text/javascript">
      var i;
      for (i=1; i<=20; i++) {
         document.writeln(
            "<p>"+i+"<sup>2</sup> = "+i*i+"</p>"
         );
      }
      </script>
   </body>
</html>
```

Figura 3.6: Script lato client usato per generare la tavola dei quadrati.

Capitolo 4

Il linguaggio HTML

4.1 Cenni storici

Il linguaggio HTML (HyperText Markup Language) è stato creato al laboratorio CERN di Ginevra, come strumento per creare collegamenti logici tra documenti diversi di testo, generando così un cosidetto *ipertesto*.

Al contrario di quello che si potrebbe pensare, HTML non è un linguaggio assoluto. Nel corso degli anni è andato via via evolvendosi. Nelle sue prime versioni era un semplice (ma ben strutturato) linguaggio di marcatura che permetteva principalmente di formattare i testi, definire alcuni tipi di dati dentro apposite marcature e creare collegamenti ipertestuali tra i documenti. Nelle versioni successive sono state implementate una serie di nuove marcature che permettevano di strutturare dati complessi sotto forma tabellare e creare strutture complesse tramite frame, livelli, ed altro ancora.

Tuttavia questo comportamento ha avuto un effetto incoerente rispetto alla device indipendency, infatti, le diverse versioni presentano tra loro elementi di incompatibilità.

Quando si crea una pagina si sceglie unilateralmente la versione con la quale essa verrà realizzata ma si è del tutto all'oscuro delle capacità dei client. Per questo motivo, siccome lo scopo principale della realizzazione di una pagina è di diffonderne la visibilità, non sempre è consigliabile utilizzare la versione più recente ma quella più diffusa; per questo motivo l'XHTML non ha ancora trovato una larga diffusione.

4.2 Caratteristiche di un documento HTML

Un documento di tipo HTML si può realizzare con un qualsiasi editor che utilizzi normale testo US-ASCII come ad esempio il classico note-pad i cui caratteri sono codificati a sette bit; al contrario non si può utilizzare un documento come Word poiché presenta caratteri di formattazione, ritorno a capo e caratteri speciali come le lettere accentate (queste, infatti, non potranno essere rappresentate cliccando direttamente sul tasto della nostra tastiera che le rappresenta). Inoltre con un documento di questo tipo si possono inserire una serie di capacità aggiuntive grazie ai **TAG** e agli **ATTRIBUTI**, in generale è possibile ottenere una pagina dove:

- Si può alternare al testo anche immagini e video grazie all'utilizzo di puntatori ipertestuali(link) e ipermediali;

- Si può formattare il testo anche se in modo moderato per permettere una sua migliore presentazione; è molto importante parlare di "moderata capacità presentativa" poiché tutti gli artifici servono semplicemente a presentare la pagina in modo gradevole, non si deve eccedere cercando effetti strani che tipicamente non hanno un buon effetto.

4.2.1 I tag

Il tag definisce un elemento della pagina HTML. E' definito da un nome identificativo circondato dai simboli "<" e ">". Di solito l'elemento viene identificato da due tag, uno d'apertura e uno di chiusura; il tag di chiusura è identico al primo ma si distingue grazie alla presenza di uno slash / come nel seguente esempio:

`<p>` ... *testo di un paragrafo* ... `</p>`

In rari casi non esiste il tag di chiusura, come nel caso del tag `<br>` che inserisce un ritorno a capo[1]. Si noti che l'uso di questo tag è fortemente sconsigliato poiché non ha una sua funzione logica (per quale motivo si desidera andare a capo?) e non contribuisce quindi alla semantica del documento.

All'interno di un tag ne possono essere aperti altri in modo nidificato, come nel seguente esempio in cui si vuole scrivere una parola in grassetto all'interno di un paragrafo:

```
<p>questa parola &egrave; <b>importante</b>
e per questo motivo la scrivo in grassetto.</p>
```

L'importante è ricordarsi di chiudere i tag sempre in modo inverso di come sono stati aperti (ossia deve essre chiuso per primo l'ultimo tag che è stato aperto)

Per HTML non è importante l'uso di lettere maiuscole o minuscole nello scrivere i tag (è "*case-insensitive*") ma in XHTML i tag devono essere scritti obbligatoriamente in minuscolo. Si suggerisce quindi di scrivere sempre tutti i tag in minuscolo in modo da avere un primo grado di compatibilità.

4.2.2 Gli attributi

Un attributo è una caratteristica opzionale di un tag che permette di caratterizzarlo meglio, fornendo maggiori informazioni ad esempio riguardo la sua posizione o dimensione. Ogni attributo è definito dal suo nome e da un valore (opzionale) e deve essere inserito all'interno del tag di apertura come in questo esempio:

`<hr width="90%">` ... *codice* ... `</hr>`

che inserisce una riga orizzontale con una lunghezza pari al 90% della corrispondente dimensione della pagina.

4.2.3 Il browser

L'HTML è quasi un linguaggio di programmazione che invece di fare dei calcoli o realizzare i percorsi che un algoritmo deve eseguire, si occupa di indicare come una pagina dev'essere realizzata. Per questo motivo il solo linguaggio da solo non è sufficiente ma deve essere

[1] In inglese "line break", da cui il nome

accoppiato ad un interprete in grado di comprendere quanto descritto, di stamparlo a video e di permettere la navigazione da una pagina ad un altra: tutto questo è compito di un BROWSER HTML.

Il lavoro di questo interprete può essere brevemente sintetizzato in tre azioni:

1. Legge il codice sorgente (HTML + estensioni);
2. Cerca di capirlo (sperando che non contenga errori...);
3. Fa del suo meglio per visualizzare quanto descritto dal codice sorgente.

Potrà sembrare strano ma non esiste un compilatore per poter vedere se il codice della nostra pagina sia corretto oppure no. Questo perché ci si accorge di eventuali errori solo in fase di fruizione della pagina; essi vengono quindi segnalati a chi legge che, anche accorgendosene, non può correggerli poiché accede ai documenti solo in lettura. A tutto questo si aggiunga anche il fatto che i browser non si comportano tutti allo stesso modo sia per ciò che riguarda la visualizzazione ma soprattutto per il comportamento dinanzi ad un errore che può essere gestito sia cercando di capire come l'errore può essere risolto sia con un salto netto della parte in questione.

Il modo di ragionare di un browser è totalmente logico, infatti nella rappresentazione non hanno alcun effetto i ritorno a capo che vi sono all'interno dello scheletro HTML. Allo stesso modo anche degli spazi multipli verranno considerati alla strenua di una singola spaziatura. Questo perché la formattazione non può essere "forzata" a priori ma si adatta alla finestra (si pensi al momento in cui si ingrandisce una finestra le scritte prima rappresentate su righe multiple appaiono poi su una sola).

Infine, come qualsiasi altro strumento informatico, necessita di frequenti aggiornamenti; l'HTML è un linguaggio estensibile e spesso si aggiungono nuovi tag o attributi. Se il browser non li riconosce perché non è aggiornato li tratterà allo stesso modo degli errori ovvero li ignorerà completamente visualizzando solo il testo racchiuso al loro interno.

La guerra dei browser

Con "guerra dei browser" si intende l'aspro conflitto commerciale e d'immagine tra i diversi produttori che tentano d'imporsi sul mercato del browser web. Ognuno di noi tende ad utilizzare un particolare browser piuttosto che un altro per vari motivi come preferenza personale, piattaforma di uso, tipo di personalizzazione ed altro ancora. Chi realizza una pagina HTML deve far in modo che sia possibile visualizzarla su ogni tipo di browser ma è molto interessato alle informazioni statistiche riguardo l'utilizzo di un tipo di browser piuttosto che un altro perché così curerà la sua pagina in modo che dia meno problemi possibili su quello che viene utilizzato dalla maggior parte degli utenti.

Se si vuole accedere a tali statistiche di utilizzazione si possono utilizzare:

- http://www.w3schools.com/browsers/browsers_stats.asp
  che a febbraio 2012 riporta le seguenti statistiche: IE=19.5% FX=36.6% Chrome=36.3% Safari=4.5% Opera=2.3%
- www.upsdell.com/BrowserNews/stat.htm
  che riporta in generale una grande variabilità tra cui una percentuale crescente di browser realizzati per apparecchi "mobile"

- www.pgts.com.au/pgtsjpgtsj0212d.html
  che a marzo 2012 riporta le seguenti statistiche: IE=41,2% FX=20.4% Chrome=12.7% Safari=11.4% Opera=3.0% Mobile=1.5% Unknown=8.1%, riportando quindi una percentuale non indifferente di browser sconosciuti che non possono essere identificati con certezza.

Queste informazioni stanno diventando via via meno accessibili poiché molti utenti cambiano la dichiarazione del loro browser così da divenire anonima, questo si verifica sia per superare alcuni vincoli politici (si pensi alle politiche del governo cinese che ostacolano l'accesso ad alcuni siti), sia per difendersi dai malware (che impiegheranno più tempo per capire con che tipologia di browser hanno a che fare e in che modo infettarlo).

Ci sono due categorie principali di browser: testuale e grafico. Un browser testuale si limita esclusivamente all'interpretazione del testo riportando invece di immagini e video delle descrizioni testuali, inoltre non fa vedere la formattazione o i colori. Viene utilizzato nelle regioni del mondo dove internet ad alta velocità è ancora un miraggio (modem 56k). Un browser grafico è invece in grado di riprodurre tutte le specifiche HTML (incluse quelle con effetti grafici); questo è il tipo di browser più diffuso come Firefox, Chrome, Opera o Internet Explorer.

4.3 Struttura generale di un documento HTML

Come illustrato nella figura 4.1 una pagina HTML ha una struttura standard caratterizzata da tre parti principali:

- il document type declaration (DTD);
- l'intestazione o header;
- il corpo del documento o body.

Le ultime due sezioni sono annidate all'interno del tag `<html>` che indica al browser il codice da interpretare. Fare attenzione a questo tipo di delimitazione poiché anche se un browser legge l'eventuale codice scritto dopo quest'ultimo tag di chiusura non sarebbe tenuto a farlo e, magari, un secondo browser non lo fa categoricamente per cui bisogna sempre inserire tutto il codice all'interno del tag `<html>`.

Un esempio semplice ma completo di una pagina HTML è riportato in figura 4.2.

4.3.1 Il DTD

Dice al browser che tipo di versione HTML è utilizzata. Se non è presente il browser prova comunque ad eseguire un'interpretazione ma non è assicurata la buona riuscita dell'operazione. In questo caso cercherà di leggere il codice con una versione inferiore ad HTML 4 perché prima di essa non era obbligatorio inserire il DTD.

Il DTD definisce gli elementi leciti all'interno del documento. Non si possono usare altri elementi se non quelli definiti. Una specie di "vocabolario" per le pagine che lo useranno. In pratica definisce la struttura di ogni elemento. La struttura indica cosa può contenere ciascun elemento, l'ordine, la quantità di elementi che possono comparire e se sono opzionali

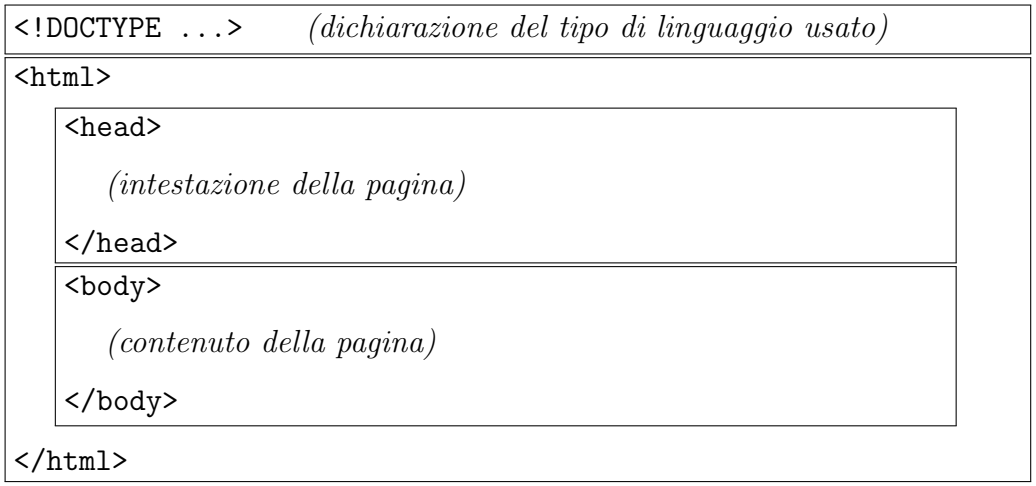

Figura 4.1: Struttura di una pagina HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <title>Esempio di pagina HTML</title>
   </head>
   <body>
      Qui posso inserire il testo del mio documento
      che, se non uso i tag di formattazione,
      viene visualizzato come semplice testo.
   </body>
</html>
```

Figura 4.2: Un semplice esempio HTML.

o obbligatori. Una specie di "grammatica". Dichiara una serie di attributi per ogni elemento e che valori possono o devono assumere questi attributi.

Vi sono tre principali tipi di DTD: Strict, Transitional e Frameset.

Un DTD di tipo Strict utilizza solo elementi non deprecati e si richiama con la seguente dichiarazione:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

Molto utile se poi vogliamo passare le pagine attuali alle nuove versioni HTML che inseriscono tutta la parte grafica nel CSS e nella layout invece di formattazioni all'interno della pagina che non hanno nessun riferimento logico del perché vengono utilizzate. Rappresenta alla lettera la sintassi e le idee dell'HTML preparando il proprio codice alla migrazione verso una nuova versione.

Un DTD di tipo Transitional permette l'uso di quasi tutti gli elementi deprecati (ad eccezione dei Frame) e si richiama con la seguente dichiarazione:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
```

Di solito sono tutti quegli elementi che realizzare in forma non deprecata sarebbe troppo difficoltoso.

Un DTD di tipo Frameset permette l'uso di qualunque elemento (inclusi quelli deprecati ed i Frame) e si richiama con la seguente dichiarazione:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
```

Altamente sconsigliato

4.3.2 L'intestazione (head)

E' uno dei campi più importanti di tutta la pagina anche se molto spesso passa in secondo piano. Al suo interno sono racchiuse numerose informazioni che, se anche non visibili, costituiscono la parte più importante per i motori di ricerca e i server. Grazie ad esso la pagina viene indicizzata, ovvero viene legata a dei riferimenti così che quando un utente ricerca delle informazioni che sono contenute nell'header della pagina questa viene reperita facilmente.

Si deve fare molta attenzione quando si inseriscono tali informazioni. Non devono essere generiche ma al contrario specifiche dell'argomento. Ad esempio non ha senso indicizzare una pagina che contiene il capitolo di un libro al numero del capitolo ma piuttosto al nome del libro e dell'autore.

All'interno dell'intestazione di una pagina possiamo utilizzare solo un numero ristretto di tag: quelli che definiscono il titolo, vari meta-dati e le relazioni logiche con altre pagine o componenti del web.

Il titolo

Il titolo della pagina si definisce col tag `<title>` ed è uno dei tag più importanti per i motori di ricerca. Quanto digitato dall'utente viene cercato sia nel titolo delle pagine sia nei

metadati come le parole chiave; per questo motivo deve essere inserito con cura in modo da specificare l'argomento che la pagina espone. Oltre a questo è anche responsabile del nome che appare sulla testata della finestra a cui spesso gli utenti non fanno nemmeno caso.

I meta-dati (meta)

Racchiudono una serie di informazioni aggiuntive relative alla pagina, quali l'autore o una parola chiave, come nel seguente esempio:

```
<meta name="author" content="Antonio Lioy">
<meta name="keywords" content="HTML">
```

Come si può facilmente intuire la sintassi del tag è definita dal termine `meta` seguito dall'attributo `name` che specifica il tipo di meta-dato e dall'attributo `content` che contiene l'informazione vera e propria. Spesso capita che un sito per aumentare la propria visibilità inserisca nei meta-dati delle parole chiave non pertinenti al contenuto (es. relative ad un personaggio famoso anche se la pagina non tratta quell'argomento), tuttavia dopo un certo periodo di tempo, i motori di ricerca capiscono questo trucco perché esaminano anche il contenuto.

Una particolare categoria di meta-dati sono quelli relativi ad informazioni che devono essere inviate sul canale HTTP prima della trasmissione della pagina. Questi meta-dati sono caratterizzati dall'attributo `http-equiv` che assume come valore quello di uno degli header di una risposta HTTP

```
<meta http-equiv="Content-type" content="text/html; charset=ISO-8859-1">
<meta http-equiv="Expires" content="Sun, 28 Feb 2010 23:59:00 GMT">
```

Che indicano invece, rispettivamente il contenuto della pagina che nell'esempio è testo scritto con codice HTML oppure l'eventuale scadenza di una pagina, si pensi al sito di un supermercato con tutte le offerte che varranno solamente fino ad un certo giorno.

4.3.3 L'internazionalizzazione di HTML

L'HTML nasce al CERN di Ginevra per questo motivo tutte le versioni antecedenti all'HTML-4 venivano scritte con una codifica ISO-8859-1 poiché pensate e progettate per quel tipo di ambiente. Tuttavia non era tollerabile che il linguaggio più utilizzato nella rete mondiale non permettesse l'utilizzo di tutte le altre lingue. Per questo motivo, l'HTML-4 incorpora l'RFC-2070 il quale specifica le regole di internazionalizzazione (spesso abbreviato in "i18n"). Così l'HTML-4 adottò lo standard ISO/IEC:10646 che indica l'UCS ovvero l'Universal Character Set che utilizza 32 bit per carattere. E' bene ricordare che un sottoinsieme dell'UCS è l'Unicode il quale, invece, utilizza 16 bit per carattere e permette solo l'utilizzo delle lingue più importanti incluso il cinese. Infine, un ulteriore sottoinsieme dell'Unicode è l'ISO utilizzato per le lingue dell'Europa Occidentale con i suoi 8 bit per carattere.

L'User Agent (browser) a sua volta determina la lingua da usare di volta in volta mediante alcuni elementi, qui elencati in ordine decrescente di priorità:

- il response header HTTP che specifica il tipo MIME, come in questo esempio

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

- specifico tag `meta` nell'header HTML, come in questo esempio

```
<meta http-equiv="Content-Type" content="text/html;
   charset=iso-8859-1">
```

- attributo charset di un elemento che punta ad una risorsa esterna, come in questo esempio

```
<a href="http://www.polito.it/" content="text/html;
   charset=iso-8859-1">
```

Le relazioni logiche (link)

Il web è per antonomasia la più grande fonte d'informazione oramai esistente. Contiene così tanti dati e pagine che spesso risulta difficile riuscire a capire le relazioni tra le varie informazioni. Il tag `<link>`, da non confondere con gli hyperlink (tag `<a>`), serve proprio ad esprimere un collegamento logico tra due o più risorse, fornendo anche una spiegazione del motivo che le accomuna. Per capire questo concetto si può immaginare un libro trasposto sul web creando una pagina HTML per ogni capitolo: è chiara la necessità di indicare che le varie pagine sono collegate perché insieme costituiscono un libro.

Un link è caratterizzato da vari attributi:

- `href` indica la URI della risorsa collegata alla pagina corrente;
- `rel` esprime la relazione esistente tra la pagina corrente e la risorsa puntata dal link e può assumere i seguenti valori:
  - `stylesheet`, il foglio di stile (capitolo 8) da applicare alla pagina;
  - `alternate`, una versione alternativa della pagina attuale (ad esempio in una lingua differente o in un formato diverso);
  - `start`, l'inizio di una sequenza di pagine correlate;
  - `contents`, la pagina che contiene l'indice[2] di una sequenza di pagine correlate;
  - `prev`, la pagina precedente in una sequenza di pagine correlate;
  - `next`, la prossima pagina in una sequenza di pagine correlate;
- `lang` indica la lingua relativa alla risorsa collegata, secondo la specifica definita nel RFC-1766 (es. `it` per l'italiano o `en-us` per l'inglese nella variante americana);
- `media` indica il formato della risorsa collegata (es. `screen` per una risorsa video o `aural` per una risorsa sonora).

La figura 4.3 contiene un esempio di uso di vari tag link.

4.3.4 Il contenuto della pagina (body)

Come dice la parola il body rappresenta tutto il corpo della pagina, in sostanza, tutto ciò che deve essere stampato a video si trova in questa sezione. Esprime come deve essere composta la pagina.

E' possibile inserire dei commenti (come nelle altre parti della pagina) che possono occupare senza problemi varie righe ma che sono racchiusi tra `<!--` e `-->` come:

[2] in inglese TOC (Table-Of-Contents)

```
<!-- this is the page chapter2.html, written in English -->
<head>
  <title>Chapter 2</title>
  <link rel="contents" href="../toc.html">
  <link rel="next" href="chapter3.html">
  <link rel="prev" href="chapter1.html">
  <link rel="stylesheet" type="text/css" href="mystyle.css">
  <link rel="alternate" media="aural" href="chapter2.mp3">
  <link rel="alternate" lang="it" href="chapter2_italian.html">
</head>
```

Figura 4.3: esempio di uso del tag `<link>`.

```
<!-- questo è un commento -->

<!-- questo
 commento occupa
 quattro righe
--!>
```

Si tenga conto che in tutto il codice scritto non essendoci formattazione posso utilizzare un qualunque numero di ritorni a capo che non avranno nessun effetto oppure posso inserire una serie di spazi in sequenza che verranno trattati come fossero uno; per cui nel momento in cui si scrive bisogna evitare di cadere nell'errore di voler utilizzare questi comportamenti per imporre una certa formattazione. Per poter capire il perché di questo comportamento basta pensare come il browser: solitamente si va a capo quando la pagina finisce ma la finestra non ha una dimensione propria, essa può essere facilmente ingrandita o ridotta, per cui automaticamente il browser, in relazione allo spazio disponibile, decide quando andare a capo.

Nonostante la presenza dei tag utilizzabili in questa sezione siano molto numerosi qui di seguito si potrà trovare una piccola descrizione dei più utilizzati, ovvero di quelli basilari nella realizzazione di una pagina HTML.

titoli o intestazioni

Esistono sei livelli di titolo o intestazione. La loro funzione è molto simile a quella del tag `<title>` usato nell'head, tuttavia in questo caso il dato verrà anche presentato di norma con un font di dimensione maggiore e in grassetto. Tuttavia non si deve confondere la presentazione con il contenuto logico poiché l'"enfatizzazione" dei titoli sta ad indicare la sua importanza informativa e non deve essere utilizzato solo perché in un punto della pagina ci sembra che una parola (che magari non ha motivo di essere relazionata direttamente alla pagina) stia meglio in grassetto e con una dimensione maggiore. Comportarsi in questo modo significa infatti fare un errore scambiando la forma col contenuto e dimostrando di non aver compreso la funzione di questo tag.

Esistono sei livelli di titolo o intestazione, indicati coi tag `<h1>`...`<h6>`. Nell'utilizzarli però bisogna seguire un ordine. In particolare è scorretto usare `<hN>` se non è preceduto da `<hN-1>`. Qui di seguito riportiamo il codice di tutti i livelli.

I paragrafi

Un paragrafo è delimitato dall'uso del tag `<p>`. Al termine di un paragrafo il browser va a capo automaticamente e può anche lasciare un piccolo spazio verticale. All'interno di questo tag si devono inserire un insieme di frasi che concettualmente identificano un tutt'uno per cui attenzione a suddividere il testo in modo corretto. Invece l'uso del tag di ritorno a capo `<br>` è fortemente deprecato perché si inserisce una formattazione fisica senza una formattazione logica ovvero chi legge il codice non capisce il significato dell'andare a capo (fine del paragrafo? inizio citazione? si sta per fare un esempio?).

rette orizzontali

E' un elemento accettato ma assume solo la funzione di abbellimento grafico; non deve essere usato come elemento logico se non insieme a quelli appropriati (ovvero tra una sezione e un altra posso inserire una riga a patto che vi sia anche il tag d'intestazione `<hN>`) questo perché il browser quando interpreta il codice gli riconosce solo l'importanza grafica. Realizza una retta che di default è centrata con larghezza 100%. Tali caratteristiche possono però essere facilmente modificate con gli appositi attributi:

- `size=`*npixel* indica lo spessore in numero di pixel;
- `width=`*npixel* indica la lunghezza assoluta in numero di pixel;
- `width=`*percentuale* indica la lunghezza come percentuale del contenitore;
- `align=`*posizione* indica la posizione all'interno del contenitore e può assumere i valori `left`, `right` o `center` per indicare una riga sbandierata a destra, sinistra o centrata.

Si tenga a mente che tutti gli attributi in cui si utilizza il numero di pixel sono deprecati poiché non si può sapere a priori quanto sarà grande lo schermo e la dimensione delle finestre dell'utente che aprirà la pagina. Non è facile nemmeno parlare di standard dato che ora come ora gli smartphone e gli applet hanno segnato una nuova frontiera dell'accesso alla rete rispetto a ciò che prima era eseguito solo dai PC.

4.3.5 Elenchi e liste

Con le liste è possibile inserire una serie di stringhe che verranno sequenziate automaticamente dal browser una sotto l'altra in modo perfettamente allineato anche quando il simbolo che precede tutti gli elementi occupa via via uno spazio crescente (si pensi alle liste ordinate con i numeri romani oppure con numeri decimali se si devono inserire più di nove righe). Come si può osservare di seguito esistono diversi modi per creare una lista, ognuna della quale avrà il suo tag specifico di inizio e fine. All'interno di qualunque lista è ammesso solo il tag `<li>` che identifica un elemento della lista. E' possibile creare liste ordinate o non ordinate.

Le liste non ordinate sono generate col tag `<ul>` (abbreviazione di Unordered List). Con l'attributo `type` è possibile modificare il tipo grafico con cui viene segnalato l'elemento della lista, anche se questa specifica è ormai deprecata perché si preferisce usare per los tesso scopo il CSS:

- `type=disc` per un pallino vuoto;

codice HTML	*risultato (nel browser)*

`Per superare l'esame 01ENY:`
`<ol type="I">`
`<li>frequentare`
`le lezioni</li>`
`<li>svolgere`
`le esercitazioni`
`di laboratorio</li>`
`</ol>`
	Per superare l'esame 01ENY:
	I. frequentare le lezioni
	II. svolgere le esercitazioni di laboratorio

Figura 4.4: Esempio di lista ordinata con numeri romani.

- `type=circle` per un pallino pieno;
- `type=square` per un quadratino pieno.

Le liste ordinate si creano col tag `<ol>`. Anche in questo caso è possibile variare il modo di presentare graficamente gli elementi grazie agli attributi. In questo caso si può utilizzare `start=`*indiceDelPrimoElemento* per dichiarare quale dev'essere l'indice di partenza degli elementi della lista, così da poter creare due liste separate ma di cui una è la continuazione logica dell'altra. Si pensi ad esempio ad una pagina che contiene i primi 10 classificati di una competizione e poi una seconda pagina che contiene un'altra lista con la classifica di tutti gli altri partecipanti: chiaramente nella seconda lista l'indice di partenza deve essere 11. L'attributo `type` definisce il modo in cui un elemento della lista viene marcato rispetto agli altri e ha numerose possibilità (può essere specificato sia sulla lista sia sul singolo elemento):

- `type=1` per usare come marcatori i numeri decimali interi (`1 2 3 4 ...`);
- `type=A` per marcatori alfabetici maiuscoli (`A B C D ...`);
- `type=a` per marcatori alfabetici minuscoli (`a b c d ...`);
- `type=I` per usare come marcatori i numeri romani maiuscoli (`I II III IV ...`);
- `type=i` per usare come marcatori i numeri romani minuscoli (`i ii iii iv ...`).

La figura 4.4 presenta un esempio di lista ordinata.

Le liste di tipo *directory* (deprecate) si creano col tag `<dir>` ed elencano tutti i componenti di un determinato insieme (es. i file presenti in una cartella, oppure le persone che abitano in un palazzo).

Le liste di tipo *menù* (deprecate) si creano col tag `<menu>` ed elencano tutte le possibili scelte relative ad un elemento (es. colori in cui è disponibile un capo di abbigliamento).

Lista di definizioni

Questo tipo di lista viene usata per realizzare elenchi di termini e corrispondenti definizioni, come ad esempio un glossario. La lista è definita col tag `<dl>` (Definition List) ed al suo interno sono ammessi solo due tipi di tag, sempre presenti in sequenza:

- il tag `<dt>` racchiude il Definition Term, ovvero la parola di cui si sta per fornire la definizione;
- il tag `<dd>` contiene la Definition Description, ovvero la definizione vera e propria del termine che lo precede.

4.4 Strumenti di controllo

Nella sezione in cui abbiamo parlato dei browser si è detto che non esistono dei veri e propri compilatori che segnalino gli errori; essi, infatti, si manifestano in lettura e a seconda di che browser viene utilizzato. A questo punto però è lecito chiedersi se esista un qualche sistema di controllo per poter osservare se effettivamente c'è qualcosa che non va e se si può intervenire tempestivamente evitando così di dover consegnare un lavoro errato.

In effetti ci sono alcuni strumenti che facilitano il lavoro di chi realizza una pagina HTML. Qui di seguito ne forniamo alcuni esempi a seconda che si tratti di pagine statiche o generate in modo dinamico.

Nel primo caso si parla di una pagina HTML che è presente all'interno di un server. A seconda di chi o del momento in cui verrà richiesta non cambierà mai. In questo caso possiamo semplicemente agire a partire dal server dove si trova la pagina in questione dandola in pasto ai vari programmi di controllo che sono nella maggior parte dei casi utilizzabili in rete ma in alcuni casi permettono anche una installazione a livello locale. Ad esempio possiamo utilizzare:

- il validatore ufficiale W3C, accessibile alla pagina http://validator.w3.org/, permette di verificare se una pagina è scritta rispettando completamente la sintassi ufficiale, fornendo anche spiegazioni dettagliate sugli errori e su come correggerli;
- il programma Tidy, disponibile su Sourceforge alla pagina http://tidy.sourceforge.net/, "ripulisce" il codice HTML e lo trasforma in una versione più recente; si può installare localmente o utilizzare in rete alla URL http://cgi.w3.org/cgi-bin/tidy/.

La validazione nel caso di pagine dinamiche non può essere fatta con gli strumenti sopra citati, perché tali pagine non esistono come file sul server ma sono generate dal server in base ad una specifica richiesta di un browser. Ad esempio si può pensare alla richiesta degli orari dei treni. Non esiste nel server della ferrovia dello stato una singola pagina per ogni combinazione di origine, destinazione, data e ora di partenza. Al contrario in base alle informazioni inserite dall'utente che visita il sito delle ferrovie il server realizza "al volo", dopo aver elaborato i dati e aver eseguito su di essi vari calcoli, la pagina e la consegna al browser. Chiaramente fare un controllo sul server non ha alcun senso; al contrario, tutte le operazioni di controllo vanno eseguite sul client attraverso uno speciale plug-in per il browser oppure visionando una ad una tutte le pagine che si possono realizzare dinamicamente e osservare se vengono riscontrati degli errori. Un ottimo plug-in per FireFox è

http://users.skynet.be/mgueury/mozilla/index.html

Va configurato in modalità "SGML parser" per avere gli stessi risultati del validatore W3C. Utilizzandolo nel browser appare anche un piccolo semaforo che ha solo il colore rosso e verde. Nel primo caso cliccandolo è anche possibile visualizzare gli errori mentre il secondo colore, com'è facile capire, indica una situazione senza problemi.

Infine bisogna fare attenzione ad alcuni errori tipici (soprattutto con gli script client-side) attraverso

http://www.htmlhelp.com/tools/validator/problems.html

4.5 Formattazione del testo

Fino adesso, abbiamo visto come suddividere il testo in porzioni logiche (intestazioni,liste,paragrafi), questo è il contenuto del testo. Dall'altra parte, esiste la formattazione del testo, cioè come si presenta fisicamente e visivamente il contenuto,lo stile fisico. In generale, si predilige (in HTML 5) usare solo la parte logica e definire da un'altra parte la parte fisica.

Con XHTML e HTML 5 sono scomparsi i tag di formattazione, cioè si deve obbligatoriamente usare CSS, in modo che esistano due sintassi diverse per definire il contenuto HTML e la formattazione CSS.

4.5.1 Stili fisici del testo

HTML 4 offre diversi tag per modificare lo stile di visualizzazione di un testo:

- `<b>` per testo in grassetto (in inglese *bold*);
- `<i>` per testo in corsivo (in inglese *italic*);
- `<u>` per testo sottolineato (in inglese *underlined*);
- `<tt>` per testo a spaziatura fissa (come quello generato da una macchina da scrivere o da una telescrivente, in inglese *teletype*);
- `<blink>` per testo lampeggiante (in inglese *blinking*);
- `<sup>` per testo sopraelevato (apice, in inglese *superscript*);
- `<sub>` per testo sottoelevato (pedice, in inglese *subscript*);
- `<s>` oppure `<strike>` per testo barrato orizzontalmente.

Si noti che l'uso di tutti questi stili (tranne `<sup>` e `<sub>`) è fortemente sconsigliato perché un motore di ricerca o un utente che legga il codice non ha modo di capire perché una determinata porzione di testo debba essere visualizzata in un modo particolare (es. grassetto o corsivo). In altre parole, manca l'indicazione della funzione logica associata allo stile fisico di visualizzazione.

Un problema sorge se supponiamo, per esempio, di fare in grassetto tutti i termini che non sono in italiano. Se un giorno, qualcuno decidesse che le parole inglesi si scrivono in blu e non in grassetto, invece tutte le altre parole, non inglesi, si scrivono in grassetto, non si può pensare di fare search and replace, perché alcune parole dovranno diventare blu (quelle in inglese) e tutte le altre dovranno rimanere in grassetto. Non è una buona idea, nell'HTML e nella composizione di testi, utilizzare il programma Word per scrivere pagine di testo, perché si rischia che il documento non abbia uno stile uniforme.

Per quanto riguarda l'uso di apici e pedici (tipici delle formule matematiche) si noti che le capacità dell'HTML 4 sono molto limitate mentre in HTML 5 è stato introdotto il tag `<math>` che permette di scrivere formule matematiche complesse usando il linguaggio MathML.

4.5.2 Stili logici del testo

Se vogliamo inserire degli stili fisici, sarebbe meglio, inserire degli stili logici :

- `<cite>` citazione `</cite>`

  bisogna definirla da qualche parte per esempio: ristretta, italico etc... in questo modo se si volesse cambiare italico con grassetto, basterebbe andare dove è stata definita la citazione e cambiarlo.

- `<code>` codice (programma) `</code>`
- `<em>` enfasi`</em>`
- `<kdb>` tastiera`</kdb>`
- `<samp>` esempio`</samp>`
- `<strong>` rinforzo `</strong>`
- `<var>` variabile`</var>`

  utile per variabili matematiche

- `<dfn>` definizione `</dfn>`

4.5.3 Altri stili logici

- `<big>` testo grande `</big>`
- `<small>` testo piccolo `</small>`

Questi due stili logici fanno scattare di un gradino in su, nella scala delle grandezze, il testo.

Si possono usare ripetutamente in modo annidato per ottenere un effetto maggiore:

- `<big><big>` testo molto grande `</big></big>`

4.5.4 Formattazione: blocchi di testo

Si possono racchiudere anche blocchi di testo:

- `<address>` . . . `</address>`

  indirizzo (tipicamente indirizzo e-mail)

- `<blockquote>` . . . `</blockquote>`

  consigliato per grosse citazioni (composta da molti paragrafi)

- `<center>` . . . `</center>`

  testo centrato (sconsigliato, perché ho l'effetto senza la spiegazione della funzione associata)

- `<pre>` . . . `</pre>`

  testo preformattato (in questo caso HTML prende il testo come è stato scritto, non serve per creare pagine che abbiano un formato fisso, ma solo per esempi)

4.6 Riferimenti a caratteri non US-ASCII

HTML normalmente è scritto in US-ASCII con MSB=0; per caratteri speciali o che comunque non rientrano nel formato US-ASCII esistono le sequenze di encoding:

per avere ...	*si scrive ...*
<	`<`
>	`>`
&	`&`
″	`&quot;`
È	`&Egrave;`
é	`&eacute;`
©	`&copy;`

I primi quattro caratteri sono importanti perché sono caratteri riservati di HTML e non possono quindi essere usati direttamente nel testo. I restanti caratteri (e simili, quali altre lettere accentate o simboli) possono anche essere generati con opportune codifiche del testo ma si consiglia fortemente l'uso delle sequenze di encoding perché è facile commettere errori ed ottenere quindi del testo non corrispondente a quanto desiderato.

Attenzione al ";" finale.

L'elenco completo di tutte le sequenze di encoding è riportato nella sezione 24 (pagina 299) dello standard HTML 4.01 e comprende:

- caratteri estesi di ISO-8859-1 (ad esempio `&raquo;` per »)
- simboli matematici (ad esempio `&exist;` per ∃)
- lettere greche (ad esempio `&alpha;` per α)
- simboli internazionali (ad esempio `&euro;` per €)

4.7 I collegamenti (hyperlink)

HTML permette di creare dei collegamenti da una pagina all'altra; utilizzando un collegamento (detto àncora in HTML) è possibile spostarsi da una risorsa ad un'altra. Il tag che identifica la presenza di un collegamento è l'àncora,indicata con `<a>`.

4.7.1 Come inserire un hyperlink

- aprire il tag di inizio àncora: `<a>`
- inserire uno spazio
- inserire l'url della risorsa, preceduto da href= e racchiuso tra virgolette
- chiudere il tag di apertura con `>`
- inserire il testo da evidenziare (quello associato all'àncora,detto" hotword")
- chiudere l'àncora: `</a>`

Ipotesi: link presenti nella pagina `http://www.lioy.it/01eny/esame.html`

URI relativa	*. . . e corrispondente URI completa*
`biblio.html`	`http://www.lioy.it/01eny/biblio.html`
`../cv.html`	`http://www.lioy.it/cv.html`
`ris/a1.html`	`http://www.lioy.it/01eny/ris/a1.html`
`/faq.html`	`http://www.lioy.it/faq.html`

Figura 4.5: esempi di link relativi.

```
Esempio:  <a href="http://www.polito.it/">POLITO</a>
```

/ posto dopo polito.it indica che deve andare nella radice (cartella dove ci sono tutti i file), nella default page (pagina con nome speciale).

4.8 Link assoluti e relativi

Nella creazione di un sito web,si mettono tipicamente tanti file HTML nella stessa cartella; se si volesse saltare da un file ad un altro è un problema, perché bisognerebbe specificare tutto il nome completo nel campo href; si hanno anche problemi se si spostano le pagine da un sito ad un altro oppure da una cartella ad una sottocartella.

È consigliabile non scrivere link assoluti (quelli completi) a meno che siano per risorse esterne; per collegamenti fra le varie pagine che costituiscono un sito si scrivono URI parziali o relative. Se si omettono parti della URI si parla di link "relativo" e le parti mancanti assumono il valore della pagina corrente. Al proposito si ricordi che il carattere "." indica la cartella corrente, la stringa ".." indica la cartella padre ed il caratter "/" separa il nome di una cartella da altri componenti della URI oppure, se usato da solo, indica la cartella radice dei dati del server web (ossia l'inizio della zona ove sono memorizzati le pagine web). La figura 4.5 mostra vari esempi di link relativi.

4.9 Punti d'accesso a documenti

MANCA SLIDE PUNTI D'ACCESSO A DOCUMENTI

Se nel link non viene specificato il punto d'accesso, ci si trova in testa alla pagina. I link con specifica di un punto d'accesso o di atterraggio permettono di andare nella sezione prescelta. Per esempio:

```
http://security.polito.it/~lioy/01eny/#materiale
```

andrà direttamente nella sezione materiale, questo si chiama punto di atterraggio; se mancasse questa indicazione andrebbe direttamente all'inizio della pagina.

Nel documento bersaglio bisogna definire il punto d'accesso tramite un'àncora con attributo name, esempio:

```
<h1>
<a name="cuc_ita">La cucina italiana</a>
</h1>
```

Non è un link cliccabile (perché non c'è l'attributo `href`), perché è un punto di atterraggio non di partenza. Il tag àncora è bivalente: se contiene `href` è un punto di partenza mentre se contiene `name` è un punto di arrivo.

Nel documento di partenza bisogna includere nell'url il nome del punto di accesso, per esempio:

```
<a href="doc2.html#cuc_ita">
```

L'attributo name è definito solo per pochi tag; nell'HTML 4.01, XHTML e HTML 5 la tendenza è di non usare più name, ma id il quale è un identificativo che può essere usato su qualunque tag. Per ciò che concerne l'àncora non fa differenza tra name e id. Il punto d'accesso può anche essere un qualsiasi elemento identificato tramite il suo "id":

```
<h1 id="cuc_ita">
La cucina italiana
</h1>
```

Questo fa si che qualunque tag possa essere un punto di atterraggio.

4.10 Immagini

Se oltre al testo si desidera inserire anche delle immagini,si utilizza il tag image (img) specificando con source (src) il puntatore all'oggetto grafico che si vuole inserire nella pagina. Esempio:

```
<img src="polito.gif">
```

(inserisce un'immagine contenuta nel file polito.gif)

Per le persone non vedenti o per coloro che si collegano tramite una linea lenta, caricare un'intera pagina può essere fastidioso. È molto importante e in certi casi obbligatorio per alcuni siti web, (es. pubblica amministrazione) che devono essere accessibili ai disabili, inserire anche l'attributo alt, il quale deve fornire un testo alternativo. In questo modo inserisce l'immagine `polito.gif` ma, se il browser non prevede l'uso della grafica, al suo posto comparirà la descrizione dell'immagine. Ad esempio:

```
<img src="polito.gif" alt="Foto del Politecnico">
```

verrà visualizzato il testo: "Foto del Politecnico".

Si presti attenzione alla differenza tra inserimento di un'immagine nella pagina o link ad essa. Il seguente codice:

```
<img src="polito.gif">
```

inserisce l'immagine all'interno della pagina, mentre il seguente codice:

```
<a href="polito.gif">foto</a>
```

inserisce un link seguendo il quale si visualizza una pagina che contiene solo l'immagine.

4.10.1 Posizionamento reciproco a testo e immagini

Di seguito, vi sono degli attributi che permettono di specificare come l'immagine debba essere posizionata rispetto al testo:

```
<img align =left . . . >
<img align =right . . . >
<img align = top . . . >
<img align = texttop. . . >
<img align = middle . . . >
<img align = absmiddle . . . >
<img align = baseline . . . >
<img align = bottom . . . >
<img align = absbottom . . . >
```

4.10.2 Formato delle immagini

Le immagini hanno una dimensione in pixel; quando si inserisce un'immagine se non si specifica nulla, l'immagine viene caricata nella sua dimensione originale. Si ha la possibilità di specificare l'ampiezza (width) e l'altezza (hight), le quali devono essere specificate in pixel. Questo permette di fare la resize dell'immagine, ossia scalata; però se si sbaglia, si rischia di deformare l'immagine. Per esempio, se avessimo un'immagine rettagolare e facessimo width=10 pixel e hight=10 pixel diventerebbe quadrata; meglio specificare solo una delle due dimenisioni.

```
<img width=w height=h . . . >
```

Permette di visualizzare la pagina velocemente (non occorre aver caricato tutta l'immagine per sapere quale spazio occupa). Esistono altri formati:

- `<img vspace=v hspace=h . . . >`
  specifica la distanza minima tra testo e immagine
- `<img border=b . . .>`
  specifica la dimensione del bordo

Supponiamo di avere un'immagine 4000×3000 pixel con 24 bit/pixel, ossia un'immagine da circa 36 MB), molto pesante come immagine. Se decidessimo di visualizzarla nella pagina in uno spazio 100×100 pixel, allora sarebbe inutile, perché faremo scaricare all'utente 36 MB per usarne solo circa 30 kB (corrispondneti a 100×100 pixel a 24 bpp). In questi casi bisogna usare un programma di grafica per salvare l'immagine direttamente nella dimensione che verrà usata (in questo caso 100×100 pixel) e non dover scaricare inutilmente grosse quantità di dati.

4.11 Font

Il tag `<font>` specifica alcune caratteristiche del testo incluso nel tag. Il suo uso è oggi sconsigliato a favore dell'uso del CSS.

I possibili attributi di questo tag sono:

colore	*nome HTML*	*valore RGB*	*colore*	*nome HTML*	*valore RGB*
	Black	#000000		Green	#
	Silver	#c0c0c0		Lime	#
	Gray	#808080		Olive	#
	White	#ffffff		Yellow	#
	Maroon	#800000		Navy	#
	Red	#ff0000		Blue	#
	Purple	#800080		Teal	#
	Fuchsia	#ff00ff		Aqua	#

Figura 4.6: colori standard HTML.

- `size=`*dimensione*
- `color=`*colore*
- `face=`*font-family* (ad esempio Arial, Helvetica, Times)

La dimensione può esser data in pixel oppure con un numero N (compreso tra 1 e 7, default 3), +N, -N

4.11.1 Colori

E' possibile specificare un colore con il formato RGB, fornendo il valore delle sue tre componenti in esadecimale preceduto dal carattere `#`. Ad esempio:

```
<font color="#ff0000">Rosso!</font>
```

Far ciò è sbagliato, perché non tutti i colori sono standard e visibili a tutti gli UA. Un colore non disponibile su uno specifico UA viene mappato sul colore disponibile più vicino nello spazio RGB.

E' anche possibile specificare un colore fornendone il nome, nel caso che tale nome sia standard e predefinito. In figura 4.6 sono riportati i nomi ed i valori RGB dei colori definiti nello standard HTML 4.01 (sezione 6.5). Specificando il nome di un colore standard (invece della sua codifica RGB) si ottengono due vantaggi: si capisce subito il colore che vogliamo usare ed abbiamo la garanzia che quel colore esista e sia supportato da tutti gli UA.

4.12 Tabelle

Per creare una tabella si usa il tag `<table>` inserendo al suo interno le singole righe col tag `<tr>`[3]

A volte, il fatto di poter organizzare un testo in righe e colonne viene abusato da chi progetta siti web, creando ad esempio una riga per l'intestazione della pagina, una colonna per il menù ed una per il testo e così via. Ciò è assolutamente errato perché quello è il layout della pagina e non una tabella.

I principali attributi di una tabella sono:

[3] In inglese "table row".

- `align` per indicare il posizionamento orizzontale rispetto al contenitore (possibili valori: `left`, `center` o `right`);

- `border` per specificare lo spessore dei bordi;

- `width` per specificare la larghezza della tabella (può essere una dimensione o una percentuale del contenitore);

- `cellspacing` per specificare la distanza in pixel tra una cella e l'altra, oppure tra una cella e il bordo (di default è pari ad un pixel, dunque occorrerà sempre azzerarlo esplicitamente se non lo si desidera);

- `cellpadding` indica la distanza tra il contenuto della cella e il suo bordo. Se il valore viene indicato con un numero intero, la distanza è espressa in pixel; il cellpadding tuttavia può anche essere espresso in percentuale. Di default la distanza è nulla.

- `summary` contiene un breve testo che riassume il contenuto della tabella (non viene visualizzato ma è molto utile per i motori di ricerca);

- `frame` specifica su quali lati si desiderano i bordi, con possibili valori `void` (nessun lato, è il valore di default), `above` (solo nel lato superiore), `below` (solo nel lato inferiore), `hsides` (solo nei lati superiore e inferiore), `lhs` (solo nel lato sinistro, left-hand side), `rhs` (solo nel lato destro, right hand side), `vsides` (solo nei lati sinistro e destro), `box` (tutti e quattro i lati) e `border` (tutti e quattro i lati);

- rules = none(da nessuna parte,è il valore di default)/groups (righe separano i gruppi siano essi gruppi di righe: < thead>, < tfoot>, < tbody> o gruppi di colonne: < colgroup>)/rows(le righe separano i vari < tr>)/cols (le righe separano le colonne)/all (le righe separano tanto i < tr>, quanto le colonne)

Ad esempio, per avere una tabella centrata con ampiezza pari al 90% dello spazio disponibile si usano i seguenti attributi:

```
<table align="center" width="90%">
```

4.12.1 Dati in tabella

Dentro una tabella si usa il tag `<tr>` per indicare una riga. Una riga conterrà al suo interno i dati delle sue colonne, specificati tramite il tag `<td>` (nel caso di una cella normale) oppure il tag `<th>` (nel caso di una cella di intestazione, come quelle contenute nella prima riga di una tabella).

Nel caso una cella debba occupare più colonne o più righe si utilizzano rispettivamente i seguenti attributi:

- `colspan=`*numero-colonne*

- `rowspan=`*numero-righe*

4.12.2 Elementi (opzionali) di una tabella

È possibile suddividere questi dati utilizzando:

- `<thead>` indica l'intestazione, ossia la parte iniziale della tabella, quella che contiene ad esempio indicazioni sul contenuto delle celle;
- `<tbody>` racchiude il corpo della tabella, ossia il contenuto vero e proprio;
- `<tfoot>` indica il footer, ossia la parte finale della tabella (ad esempio, i totali dei dati riportati nelle colonne);
- `<caption>` inserisce una didascalia per illustrare il contenuto della tabella.

I tag `thead`, `<tfoot>`, `<tbody>` consentono di individuare gruppi di righe ("row group").

Da notare che, contrariamente a quello che si potrebbe pensare, il tag`<tfoot>` che chiude la tabella, è anteposto rispetto al `<tbody>`. L'idea di base è che il browser nell'eseguire il rendering del codice tenga conto della parte iniziale e della parte finale della tabella, e il corpo vero e proprio sia sviluppato nella sua interezza tra le due estremità.

Un'altra particolarità è che le celle all'interno del tag `<thead>` possono essere indicate con `<th>` ("table head"), al posto del consueto `<td>` ("table data"), in questo caso il contenuto delle celle è automaticamente formattato in grassetto e centrato.

4.12.3 Table: attributi di riga,header e dati

- align = allineamento - orizzontale (left, center, right)
- valign = allineamento - verticale (top, middle, right) (baseline)
- bgcolor = colore di background

4.12.4 Table: gruppi di colonne

È possibile raggruppare colonne tramite

- `<colgroup span=n width=. . . align=. . . valign=. . .>`

  gruppo strutturale di n colonne,ciascuna con gli attributi specificati
- `<col span=n width=. . . align=. . . valign=. . . >`

  definizione di attributi per una o più colonne

Il tag `<colgroup>` va posto subito dopo il tag `<caption>` e prima di `<thead>`, e consente di impostare un layout unico per le colonne senza avere la necessità di specificare allineamento del testo, o il colore di sfondo per ogni singola cella. Con l'attributo `span` possiamo impostare il numero di colonne che fanno parte del gruppo.

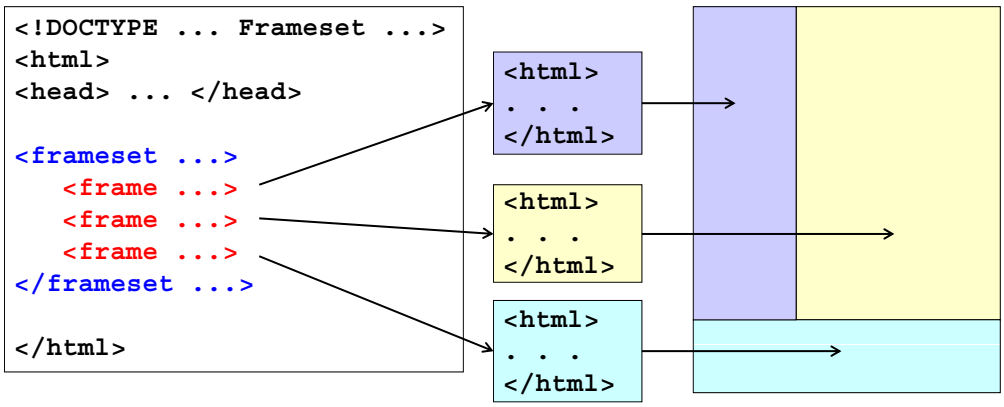

Figura 4.7: Esempio di pagina strutturata in frame.

4.13 I frame

Una pagina web si dice strutturata in frame quando è divisa in zone il cui contenuto è specificato da altri file HTML. I contenuti di ogni frame sono indipendenti gli uni dagli altri.

Quando furono inventati, i frame ebbero un grande successo poiché permettevano indubbi vantaggi. Ad esempio, permettevano di non ricaricare l'intera pagina, ma soltanto la parte di essa desiderata, che era un gran vantaggio dal momento che la navigazione era molto lenta. Un altro aspetto vantaggioso era quello di poter mantenere un menù o un footer fissi in una parte della pagina, senza doverli necessariamente inserire in tutte le pagine del sito.

Nel passato per creare il layout delle pagine erano molto usate anche le tabelle ricorrendo ai tag `<thead>`, `<tbody>` e `<tfoot>`. Questi permettevano di creare una pagina divisa in sezioni.

Con il tempo anche l'uso delle tabelle è venuto meno, poiché non esisteva collegamento tra la formattazione delle pagine e il significato logico. Inoltre, creare un layout di pagina facendo uso di tabelle era difficile da gestire, perché mischiava alla visualizzazione dei dati i dati stessi e riempiva le pagine con molto codice rallentando lo scaricamento.

Oggi sia l'uso dei frame sia quello delle tabelle sono fortemente deprecati. Si consiglia invece di usare fogli di stile (CSS, Cascading Style Sheets) come spiegato nel capitolo 8.

4.13.1 Frameset e frame

La struttura della pagina organizzata in frame, come illustrato nella figura 4.7, è simile a quella tradizionale, sostituendo il tag `<body>` con `<frameset>`. Inoltre è possibile annidare frame set per creare suddivisioni più complesse della pagina.

Il contenuto di ciascun frame è specificato nel codice HTML della pagina tramite:

`<frame src="`*URI*`" name="`*nome_del_frame*`">`

Nel caso in cui il browser su cui viene visualizzata la pagina non sia in grado di riconoscere la struttura dei frame, la pagina non viene visualizzata; in questo caso è utile l'uso del tag `<noframe>` per avvisare l'utente di tale problema inserendo all'interno del tag un opportuno avviso.

4.13.2 Spazio occupato dal frame

Nell'utilizzare la struttura dei frame è possibile specificare la porzione della pagina occupata da ciascun frame, usando: la percentuale di spazio occupato, il valore assoluto della dimensione (in pixel) oppure la notazione con asterisco (*) per indicare l'utilizzo di tutto lo spazio disponibile. L'uso dei pixel, come unità di misura, è sconsigliato, poiché non si conoscono a priori la dimensione e le caratteristiche della finestra su cui sarà aperta la pagina. Usando invece misure percentuali si possono mantenere facilmente le proporzioni desiderate della pagina su qualunque tipo di schermo.

In caso di "overflow", cioè se lo spazio riservato a un frame non è sufficiente a contenere la pagina, si attivano le barre di scorrimento, orizzontale e/o verticale.

4.13.3 Navigazione dei frame

Quando si inserisce un link in una pagina che viene aperta in un frame, è bene indicare in quale frame (o finestra) deve essere visualizzata tale pagina (pagina bersaglio)

`<a href="`*URI*`" target="`*nomeframe*`">`*hot-word*`</a>`

In questo esempio l'attributo `target` viene usato per indicare che il link deve essere aperto in un determinato frame all'interno della pagina. Per questo motivo può essere utile assegnare dei nomi ai frame, in modo da poterli distinguere fra loro.

L'attributo target può essere usato anche coi seguenti valori speciali:

- `_blank` apre la pagina in una nuova finestra;
- `_self` apre la pagina nello stesso frame (opzione di default);
- `_parent` apre la finestra nel frame set di ordine superiore;
- `_top` apre la finestra in modo da occupare l'intera pagina.

4.13.4 Un esempio di pagina organizzata a frame

```
<!-- pagina iniziale -->
<frameset rows="80%,20%">
  <noframe>
    <p>Pagina non visualizzabile</p>
  </noframe>
  <frameset cols="100,*">
    <frame src="menu.html">
    <frame src="p1.html" name="content">
  </frameset>
  <frame src="footer.html">
</frameset>
```

Nell'esempio possiamo vedere come effettuare una suddivisione in frame della pagina. Una pagina di questo tipo è divisa in due frame orizzontali, il primo dei due a sua volta diviso in altri due frame verticali. La struttura della pagina è quella rappresentata nella figura 4.7.

In un secondo esempio possiamo vedere com'è composta una delle pagine inserita in un frame. Questa consiste in una pagina HTML completa.

```
<!-- menu.html -->
<html>
<head>...</head>
<body>
<p><a href="p1.html" target="content">Pag. 1</a>
<p><a href="p2.html" target="content">Pag. 2</a>
<p><a href="p3.html" target="content">Pag. 3</a>
</body>
</html>
```

La pagina `menu.html` contiene dei link ad altre pagine, che devono essere aperte nel frame denominato "`content`".

4.14 I frame in-line (iframe)

Un frame in-line (o semplicemente *iframe*) contiene una normale pagina HTML (come i normali frame) ma viene trattato come un singolo oggetto, ad esempio come se fosse un'immagine. Per questo motivo può essere posizionato in un punto qualsiasi all'interno della pagina, senza bisogno di suddividerla tutta in frame.

La sintassi per gli iframe è identica a quella dei frame, ma si usa il tag `<iframe>` anziché `<frame>`. Gli attributi `height` e `width` specificano lo spazio da riservare per l'iframe, attivando eventualmente le barre discorrimento nel caso che la dimensione del contenuto ecceda lo spazio riservatogli. Il nome del frame è specificato con l'attributo `name`, come nel caso di normali frame. Si possono utilizzare per definire le caratteristiche dell'iframe anche i seguenti attributi: `frameborder`, `marginwidth`, `marginheight`, `scrolling` (che può assumere i valori `yes`, `no` o `auto`), `align`, `vspace`, `hspace`.

Se si inserisce del testo tra `<iframe>` e `</iframe>`, questo viene ignorato da tutti i browser che interpretano gli iframe mentre viene visualizzato da quelli che non capiscono questo tipo di tag. Quindi, conviene inserire del testo per segnalare all'utente l'eventuale errore di interpretazione da parte del browser.

Inizialmente questo tipo di frame era supportato solo da Internet Explorer ma oggi è interpretato da tutti i maggiori browser.

4.15 I tag DIV e SPAN

I tag `<div>` e `<span>` sono stati introdotti in HTML 4.0 per raggruppare parti della pagina. Non possiedono uno specifico significato logico ma sono utili per applicare più facilmente uno specifico layout o formato.

Il tag `div` identifica un blocco di testo. Tipicamente il browser va a capo alla fine di un blocco di questo tipo, come se si trattasse di un paragrafo specificato col tag `p`.

Il tag `span` identifica una parte all'interno di un blocco di testo (esempio tipico poche parole all'interno di un paragrafo).

Questi tag sono molto usati per creare, con un opportuno CSS, un layout di pagina senza ricorrere alle tabelle o ai frame. Associando gli attributi `id` e `class` a questi due tag è possibile farvi riferimento dal CSS. In questo modo anche i tag che non ammettono l'attributo `name` possono essere formattati tramite CSS.

4.16 Attributi generali dei tag HTML

I seguenti attributi valgono per tutti i tag presenti in HTML e svolgono le seguenti funzioni:

- aaa
- `id=`*stringa*
  identificativo univoco dell'elemento all'interno della pagina, che può servire per vari scopi (es. àncora per un link, riferimento all'elemento da parte di uno script, riferimento per uno stile specifico in CSS);
- `class=`*classe1 classe2 ...*
  elenco di classi CSS da applicarsi all'elemento;
- `title=`*titolo*
  breve testo visualizzato come pop-up quando si passa il puntatore sopra l'elemento;
- `lang=`*lingua*
  lingua in cui è scritto il testo dell'elemento, utile nel caso in cui la pagina venga letta automaticamente o se la pagina contiene testi in diverse lingue; può assumere i seguenti valori: `en`, `it`, `fr`, `de`, ...

4.17 Favourite icon

La *favourite icon* è l'immagine in miniatura (anche detta icona) caratteristica di un sito web che viene visualizzata dai browser nella barra degli indirizzi prima della URL della pagina. Quando è stata introdotta in HTML è stata decisa solo la sua dimensione (deve essere un'immagine 16×16 pixel) e non altre sue caratteristiche. I browser più vecchi visualizzano l'icona solo se è un'immagine in formato "MS icon" contenuta in un file con nome (fisso) `/favicon.ico` presente nella root (cartella radice) del server web.

Si è quindi usata la sintassi dei link per esplicitare le caratteristiche dell'icona ma potendone variare solo il nome e la posizione:

```
<link rel="shortcut icon" type="image/vnd.microsoft.icon" href="/my.ico">
```

Infine, i nuovi browser supportano lo standard de-facto che permette di specificare anche altri formati grafici, come nel seguente esempio che usa il formato PNG:

```
<link rel="icon" type="image/png" href="/icons/my.png">
```

Si noti che col parametro `href` è possibile specificare un'icona corrispondente ad un file presente in qualunque punto dello spazio web.

Capitolo 5

I form HTML ed il loro uso nel web dinamico

5.1 Struttura di base di un form HTML

Il *form* è un costrutto HTML fondamentale per creare un'architettura web dinamica perché permette di creare una pagina HTML in cui l'utente può inserire dei valori e trasmetterli al server (o a uno script locale). Un form è creato tramite il tag `<form>` ed ha la seguente struttura:

`<form name=`*identificatore* `action=`*URI* `method=`*metodo_HTTP* `>`
... controlli del form
... altro codice HTML
`</form>`

All'interno di un form, oltre a normale codice HTML, possono essere inseriti speciali tag – detti *controlli HTML* – per creare elementi che permettono l'interazione con l'utente, ovvero permettono all'utente di controllare in qualche modo il comportamento del sito web.

L'attributo `action` può contenere una URL (attiva) a cui inviare i dati del server; sulla base di questi può essere generata dal server una pagina di risposta dinamica.

L'attributo `method` può contenere i valori `get` o `post` per indicare la modalità di trasmissione dei dati al server, rispettivamente col metodo HTTP GET o POST. I due metodi differiscono per quanto riguarda la riservatezza delle informazioni trasmesse e la possibilità di fare caching e debug.

Il form, così come ciascuno suo controllo, può essere identificato tramite un attributo `name`, oppure tramite un attributo `id` (identificativo univoco, usabile per qualunque tag). Entrambi gli attributi possono essere usati per accedere ad uno specifico elemento tramite uno script lato client (es. JavaScript o VBScript) oppure per identificare un dato trasmesso al server (e poterlo quindi elaborare tramite uno script o programma lato server).

5.2 I controlli di input

Il tag `<input>` è caratterizzato da uno specifico `type` che può assumere i valori `text` (per la creazione di un campo di testo), `password`, `checkbox`, `radio`, `image`, `file`, `hidden`, `reset` e `button`.

5.2.1 Tipi di pulsante

I pulsanti possono essere creati attraverso il tag `<input>` specificando come tipo `submit`, `reset` o `button` oppure tramite il tag `<button>` specificando poi come tipo un valore tra `submit`, `reset` o `button`. In entrambi i casi i pulsanti di tipo `submit` consentono l'invio dei dati inseriti nel form al server, mentre un pulsante di tipo `reset` reimposta il form ai valori iniziali (non necessariamente nulli); un pulsante di tipo `button` invece non svolge nessuna azione predefinita, ma può essere associato, tramite un evento DOM, ad una funzione Javascript. Rispetto al tag `<input>`, utilizzando `<button>` si ha a disposizione una sintassi più ricca (che consente ad esempio di inserire immagini). Inoltre il tag `<button>` può essere utilizzato anche esternamente ad un form, ma in questo caso non può essere di tipo `submit` o `reset`, proprio perché non è associato ad un form.

5.2.2 Controlli orientati al testo

Il tag `<input>` ammette una serie di controlli opzionali, utilizzabili per definire ulteriori impostazioni del form:

- `size=`n permette la creazione di una zona di testo che consente la visualizzazione di n caratteri;
- `maxlength=`m impedisce che nella zona di testo vengano scritti più di m caratteri;
- impostando `type=password` si fanno comparire sullo schermo una serie di asterischi (*) al posto dei caratteri realmente inseriti (non è comunque un modo sicuro per la trasmissione di password);
- impostando `type=hidden`, viene trasmesso al server il valore di un campo non visibile all'utente (in genere preimpostato tramite `value`).

E' bene notare che, anche se viene usato `hidden`, l'utente può comunque conoscere l'esistenza ed il contenuto del campo nascosto, visualizzando il sorgente HTML della pagina. Inoltre, salvando la pagina ed aprendola tramite un editor HTML, può anche modificarne il contenuto ed inviare al server dati diversi da quelli precedentemente presenti (è quindi opportuno ricontrollare lato server i dati trasmessi dal client). Uno dei possibili utilizzi di campi nascosti consiste nel trasmettere al link di destinazione informazioni su dove l'utente ha trovato il collegamento, permettendo così di sapere quanti utenti arrivano su un dato sito a partire da un altro che lo pubblicizza.

Utilizzando la struttura

`<textarea rows=`*numero_righe* `cols=`*numero_colonne* `name=`*identificativo*`>`
... testo_iniziale ...
`</textarea>`

è possibile creare un'area di testo, indicando il numero di righe e di colonne da visualizzare desiderato; in questo caso il testo preimpostato non va inserito tramite `value`, bensì tra i tag di apertura e chiusura (come mostrato nell'esempio precedente).

```
<form action="/cgi-bin/query" method="get">
<p>your name: <input type="text" name="nome"></p>
<p>your home page: <input type="text" name="home" value="http://"></p>
<p>password: <input type="password" id="pswd"></p>
<p>
  <input type="submit" value="ok">
  <input type="reset" value="annulla">
</p>
</form>
```

Figura 5.1: esempio di un form.

5.2.3 Esempio di form

La figura 5.1 contiene un semplice esempio di creazione di un form. Tramite questo codice HTML si apre il form dichiarando la `action` (si fa qui riferimento ad un'applicazione CGI) ed il metodo HTTP da usare per il trasferimento (nel nostro caso GET), si apre il paragrafo, si inserisce il testo `your name`, si crea il campo per l'inserimento del nome (avente `id=''nome''`), si crea il campo per l'inserimento della home page, valorizzandolo con `http:/` (l'utente può comunque cancellare il testo preinserito) e si inserisce il campo destinato a contenere la password e i due pulsanti per l'invio dei dati (*submit*) ed il ripristino dei valori iniziali (*reset*); si chiude infine il paragrafo ed il form.

5.2.4 Controllo a scelta singola (menù)

È possibile creare un controllo a scelta singola (visualizzato graficamente dal browser tramite un menù a tendina) attraverso il tag `<select>`. Il tag `<option>` racchiude poi le varie opzioni; tra queste è possibile indicarne una di default tramite l'attributo `selected`, come mostrato nel seguente esempio:

```
<select name=...>
       <option label=...>
       <option>...</option>
       <option selected>...</option>
</select>
```

Qualora fosse necessario realizzare un menù a due livelli si può utilizzare il tag `<optgroup>`. Si può osservare come il nome dell'opzione possa essere indicato sia scrivendolo direttamente tra i tag di apertura e chiusura dell'opzione sia tramite l'attributo `label` (in genere si tende a preferire quest'ultima modalità).

5.2.5 Controlli a scelta multipla

Checkbox

Tramite l'attributo `checkbox` si può creare un menù a scelta multipla in cui ogni elemento può, indipendentemente dagli altri, essere ON oppure OFF; tutte le opzioni selezionate (*checked*) vengono quindi inviate al server; è necessario tenere in considerazione che con questo tipo di controllo l'utente può selezionare un numero arbitrario di opzioni (eventualmente anche nessuna).

```
<form action="/cgi-bin/query" method="get">
<p>
Compose your own fruit salad:
<br>
<input type="checkbox" id="banana"> Banana
<input type="checkbox" id="apple" checked> Apple
<input type="checkbox" id="orange"> Orange (red)
<br>
<input type="submit">
<input type="reset">
</p>
</form>
```

Figura 5.2: esempio di utilizzo di `checkbox`.

```
<form action="/cgi-bin/query" method="get">
<p>
Select your preferred fruit:
<input type="radio" name="frt" value="banana">
Banana <br>
<input type="radio" name="frt" value="apple" checked>
Apple <br>
<input type="radio" name="frt" value="orange">
Orange (red) <br>
<input type="submit">
<input type="reset">
</p>
</form>
```

Figura 5.3: esempio di utilizzo di `radio`.

La figura 5.2 contiene un semplice esempio di creazione di un controllo a scelta multipla utilizzando `checkbox`.

Radio

Tramite l'attributo `radio` è possibile creare un menù a scelta multipla con opzioni mutuamente esclusive: l'utente deve quindi selezionare una sola opzione; in questo caso i vari elementi di tipo ON/OFF sono identificati dallo stesso `name` e non si può utilizzare `id`. La figura 5.3 contiene un semplice esempio di creazione di un controllo a scelta multipla con `radio`.

5.2.6 Controlli a sola lettura o disabilitati

Talvolta in un form può essere necessario avere dei campi che non possano essere modificati dagli utenti; a tal fine è possibile utilizzare gli attributi `readonly` e `disabled`. In ogni caso bisogna tenere conto che l'utente può agire, tramite un apposito editor, direttamente sul sorgente, eventualmente anche rimuovendo questi attributi e modificando il valore dei parametri protetti.

Readonly

Impostando un campo come `readonly` (valido nei controlli `input` e `textarea`) il valore di questo può essere cambiato solo attraverso uno script client-side, ma l'utente non può modificarlo direttamente (perlomeno non nel browser). Un campo con questo attributo può, per esempio, essere usato per visualizzare il totale di un'operazione o l'ammontare complessivo di un acquisto.

Disabled

Utilizzando l'attributo `disabled` si impedisce all'utente di modificare il campo ed inoltre, a differenza di quanto avviene con `readonly`, questo non verrà trasmesso al server. L'attributo `disabled` è valido nei seguenti controlli:

- input;
- textarea;
- button;
- select;
- option;
- optgroup.

5.3 Interazione tra form e script

Tramite gli eventi DOM è possibile far eseguire uno script al verificarsi di un determinato evento (es. `onClick`, `onSubmit`, `onChange`), come mostrato nel seguente esempio:

```
onClick= esegui_azione();
```

Lo script può essere scritto all'interno dello stesso sorgente HTML come funzione o essere contenuto in un file esterno.

All'interno dello script si può accedere ai dati presenti nel form in due diversi modi: o indicando la gerarchia completa dei nomi secondo il modello DOM o estraendo direttamente l'elemento tramite l'ID univoco. Esempio:

```
alert (document.f1.frt.value)
```

o in alternativa:

```
alert (document.getElementById("frt").value)
```

```
<script type="text/javascript">
function validateForm()
{
  formObj.nome.value=="") {
    alert ("Non hai introdotto il nome!");
    return false;
  }
  else if (formObj.eta.value=="") {
    alert ("Non hai introdotto l'età!");
    return false;
  }
  else if ... return false;
  // tutte le verifiche sono OK
  return true;
}
</script>

...

<form name="sample" method="post" action="..." onSubmit="return
   validateForm()">
<p>Nome: <input type="text" name="nome" size="30"></p>
<p>Et&agrave;: <input type="text" name="nome" size="3"></p>
<p>Data di nascita: <input type="text" name="nascita" size="10"></p>
<p><input type="submit"> <input type="reset"> </p>
</form>
```

Figura 5.4: Esempio di codice per la validazione di un form.

5.4 Validazione client-side dei valori dei controlli di un form

Tramite gli eventi DOM è possibile eseguire uno script client-side che effettui dei controlli formali sui dati inseriti dall'utente; si evita in questo modo di dover trasmettere inutilmente dati al server. L'evento a cui normalmente si associa uno script di validazione dati è `onSubmit`: se lo script restituisce valore "true" i dati vengono inviati al server, mentre se il valore restituito è "false" si visualizza un messaggio di errore (il più esplicativo possibile) all'utente, che può così correggere i dati errati e tentare un nuovo invio. Talvolta può essere preferibile eseguire il controllo sui dati man mano che l'utente li inserisce e non al momento dell'invio del form; in questi casi è possibile utilizzare l'evento `onChange`, facendo scattare la verifica non appena l'utente si sposta dal campo valorizzato. La figura 5.4 riporta un esempio di codice per la validazione di un form.

5.4.1 Linee guida per la validazione di un form

I controlli da fare variano in base al tipo di campo richiesto: si può ad esempio andare a verificare che l'utente abbia effettivamente inserito un valore e che il tipo di dati (numeri, testo, data) corrisponda a quello atteso. In generale è conveniente seguire un approccio

di tipo "looks good" piuttosto che "doesn't look bad": è infatti più semplice verificare che il valore inserito sia del tipo atteso, piuttosto che impostare un elenco esauriente di controlli volti ad individuare dati non accettabili. Oltre a rilevare gli errori, è fondamentale comunicare nel miglior modo possibile all'utente le cause che lo hanno generato (spiegare cioè per quale motivo quel valore è inaccettabile), agevolandolo così nella correzione; è inoltro opportuno valutare caso per caso se sia preferibile segnalare un errore per volta o tutti gli errori in un unico messaggio. Un esempio concreto di validazione di un campo può consistere nel controllo della correttezza di un CAP: in questo caso un primo controllo può consistere nel verificare che sia stato effettivamente inserito un valore, che questo contenga solo caratteri numerici ('0'...'9') e che questi siano esattamente cinque; inoltre, se si ha a disposizione un elenco esaustivo di tutti i CAP esistenti, si può verificare che il valori inserito sia effettivamente presente tra di essi.

5.5 Trasmissione dei parametri di un form

Quando scriviamo un form, dobbiamo specificare il metodo da usare per la trasmissione dei valori dei controlli al server, scegliendo tra il metodo `GET` e quello `POST`. In questa sezione analizzare le differenze e criticità di questi metodi.

5.5.1 Trasmissione di un form tramite metodo GET

Se il metodo specificato è Get, la URI corrispondente al campo action verrà modificata. In coda alla URI, infatti, verrà aggiunto il simbolo di punto interrogativo `?` e tutti i parametri espressi nella codifica application/x-www-form-urlencoded. Il body della richiesta rimarrà vuoto poiché i parametri del form verranno inseriti sulla riga di comando.

La stringa che contiene i parametri è formata dal nome del controllo seguita dal simbolo di uguale `=` e dal suo valore. Nel caso in cui vi sia più di un parametro, il simbolo di e commerciale & servirà da separatore fra i controlli. Esempio di stringa

```
nome_ctrl1=val_ctrl1&nome_ctrl2=val_ctrl2
```

Alcuni caratteri introdotti nel nome o fra i valori inseriti possono però creare problemi se messi in una URI. Fra questi vi è lo spazio, che, se inserito in una url, la interromperebbe. Questo carattere viene quindi sostituito dal simbolo più `+`. Altri caratteri speciali o con significati particolari vengono sostituiti dai caratteri %xx dove xx indica il numero esadecimale del loro codice ISO-8859-1. La figura 5.5 e figura 5.6 mostrano i caratteri e i rispettivi codici, ricordarsi di aggiungere % all'inizio.

Un esempio di un form che usa il metodo GET è il seguente

```
<form name="sample" method="get" action="/cgi-bin/acquisisci">
  Nome e cognome:
    <input type="text" name= "cognome" size= 30 ><br>
  Numero di figli:
    <input type="text" name="figli" size="3"><br>
  Data di nascita:
    <input type="text" name="nascita" size="10"><br>
  <input type="submit">
  <input type="reset">
</form>
```

CARATTERI STAMPABILI					
HEX	CARATTERE	HEX	CHARACTER	HEX	CARATTERE
0x20	<SPACE>	0x40	@	0x60	`
0x21	!	0x41	A	0x61	a
0x22	"	0x42	B	0x62	b
0x23	#	0x43	C	0x63	c
0x24	$	0x44	D	0x64	d
0x25	%	0x45	E	0x65	e
0x26	&	0x46	F	0x66	f
0x27	'	0x47	G	0x67	g
0x28	(	0x48	H	0x68	h
0x29	)	0x49	I	0x69	i
0x2A	*	0x4A	J	0x6A	j
0x2B	+	0x4B	K	0x6B	k
0x2C	,	0x4C	L	0x6C	l
0x2D	-	0x4D	M	0x6D	m
0x2E	.	0x4E	N	0x6E	n
0x2F	/	0x4F	O	0x6F	o
0x30	0	0x50	P	0x70	p
0x31	1	0x51	Q	0x71	q
0x32	2	0x52	R	0x72	r
0x33	3	0x53	S	0x73	s
0x34	4	0x54	T	0x74	t
0x35	5	0x55	U	0x75	u
0x36	6	0x56	V	0x76	v
0x37	7	0x57	W	0x77	w
0x38	8	0x58	X	0x78	x
0x39	9	0x59	Y	0x79	y
0x3A	:	0x5A	Z	0x7A	z
0x3B	;	0x5B	[	0x7B	{
0x3C	<	0x5C	\	0x7C	|
0x3D	=	0x5D	]	0x7D	}
0x3E	>	0x5E	^	0x7E	~
0x3F	?	0x5F	_	0x7F	<DEL>

Figura 5.5: Caratteri stampabili.

CARATTERI ASCII ESTESI					
HEX	CARATTERE	HEX	CARATTERE	HEX	CARATTERE
0x80	€	0xAB	«	0xD6	Ö
0x81		0xAC	¬	0xD7	×
0x82	‚	0xAD		0xD8	Ø
0x83	ƒ	0xAE	®	0xD9	Ù
0x84	„	0xAF	¯	0xDA	Ú
0x85	…	0xB0	°	0xDB	Û
0x86	†	0xB1	±	0xDC	Ü
0x87	‡	0xB2	²	0xDD	Ý
0x88	ˆ	0xB3	³	0xDE	Þ
0x89	‰	0xB4	´	0xDF	ß
0x8A	Š	0xB5	µ	0xE0	à
0x8B	‹	0xB6	¶	0xE1	á
0x8C	Œ	0xB7	·	0xE2	â
0x8D		0xB8	¸	0xE3	ã
0x8E	Ž	0xB9	¹	0xE4	ä
0x8F		0xBA	º	0xE5	å
0x90		0xBB	»	0xE6	æ
0x91	‘	0xBC	¼	0xE7	ç
0x92	’	0xBD	½	0xE8	è
0x93	"	0xBE	¾	0xE9	é
0x94	"	0xBF	¿	0xEA	ê
0x95	•	0xC0	À	0xEB	ë
0x96	–	0xC1	Á	0xEC	ì
0x97	—	0xC2	Â	0xED	í
0x98	˜	0xC3	Ã	0xEE	î
0x99	™	0xC4	Ä	0xEF	ï
0x9A	š	0xC5	Å	0xF0	ð
0x9B	›	0xC6	Æ	0xF1	ñ
0x9C	œ	0xC7	Ç	0xF2	ò
0x9D		0xC8	È	0xF3	ó
0x9E	ž	0xC9	É	0xF4	ô
0x9F	Ÿ	0xCA	Ê	0xF5	õ
0xA0	<space>	0xCB	Ë	0xF6	ö
0xA1	¡	0xCC	Ì	0xF7	÷
0xA2	¢	0xCD	Í	0xF8	ø
0xA3	£	0xCE	Î	0xF9	ù
0xA4	¤	0xCF	Ï	0xFA	ú
0xA5	¥	0xD0	Ð	0xFB	û
0xA6	¦	0xD1	Ñ	0xFC	ü
0xA7	§	0xD2	Ò	0xFD	ý
0xA8	¨	0xD3	Ó	0xFE	þ
0xA9	©	0xD4	Ô	0xFF	ÿ
0xAA	ª	Õ	Õ		

Figura 5.6: Caratteri estesi.

In questo form sono stati creati tre campi di testo, dove l'utente può inserire rispettivamente il nome e cognome, il numero di figli e la data di nascita. Se il precedente form venisse riempito dal signor Marco Noè, nato il 30/10/74 e padre di 3 figli la stringa che verrebbe creata sarebbe la seguente

```
cognome=Marco+No%E8&figli=3&nascita=30%2F10%2F74}
```

Questo è quello che si vede usando uno sniffer anche se l'utente ha inserito i valori usando i caratteri "è", "/" e lo spazio, che sono stati automaticamente trasformati dal browser rispettivamente in `%E8`, `%2F` e `+`. Analogamente, se scriviamo una pagina ASP e utilizziamo i comandi Request.QueryString o Request.Form (in base al metodo usato), i dati che leggeremo saranno già trasformati lato server e ritroveremo quindi i caratteri originali.

```
<form method="get" action="/cgi/insaula">
  <table>
    <tr>
      <td>Numero aula:</td>
      <td><input type="text" size="8" name="num"></td>
    </tr>
    <tr>
      <td>Sede:</td>
      <td><input type="text" size="15" name="sede"></td>
    </tr>
    <tr>
      <td><input type="submit" value="Invia"></td>
      <td><input type="reset" value="Cancella"></td>
    </tr>
  </table>
</form>
```

Figura 5.7: Form per prenotazione aula.

5.5.2 Trasmissione di un form tramite POST

Se i parametri del form vengono trasmessi tramite Post, la uri coincide col campo action e quindi non viene modificata. Se non specifichiamo nient'altro, nell'header http, il campo Content-Type sarà di tipo application/x-www-form-urlencoded ed il body non sarà vuoto ma avrà una sua lunghezza, specificata nel campo Content-Length. Il body conterrà la stessa stringa che nel Get veniva inserita dopo il punto interrogativo.

Si può anche decidere un altro tipo di trasmissione specificando che il Content-Type dev'essere multipart/form-data. In questo caso il body non sarà un semplice testo ma un messaggio mime vero e proprio che contiene una sezione per ogni parametro. L'unico caso in cui conviene utilizzare questo tipo di trasmissione è quello in cui abbiamo usato un controllo di tipo file cioè quando vi è una richiesta di upload di un file. In questo caso se non utilizziamo una trasmissione di tipo multipart il trasferimento non funzionerà, questo perché il file è binario e tipicamente lungo e non può quindi essere trasmesso sulla riga di comando.

5.6 Esempio trasmissione di un form con GET

Consideriamo come esempio un form di prenotazione (Figura 5.7) i cui campi richiedono l'inserimento della sede e del numero dell'aula da prenotare. Analizziamo cosa succederebbe nel caso in cui il form fosse contenuto nella pagina `http://www.server.it/formaula.html`, il controllo associato al numero dell'aula fosse riempito col valore `12A` e quello relativo alla sede con `Sede Centrale`.

Innanzitutto, per creare la richiesta, alla URI verrebbero concatenati i valori dei due controlli di testo, opportunamente codificati; nella barra degli indirizzi del browser comparirebbe quindi la seguente stringa:

```
http://www.server.it/cgi/insaula?num=12A&sede=Sede+Centrale
```

Sul canale TCP verrebbe trasmessa una richiesta HTTP contenente solo l'header e nessun body:

```
GET /cgi/insaula?num=12A&sede=Sede+Centrale HTTP/1.1
Host: www.server.it
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Referrer: http://www.server.it/formaula.html
Accept-Language: it
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.0)
Connection: Keep-Alive
```

Lato server, nella variabile d'ambiente `QUERY_STRING` sarà presente tutto il testo della URI dopo il carattere punto interrogativo, nella forma codificata:

```
num=12A&sede=Sede+Centrale
```

L'applicazione CGI dovrà quindi eliminare la codifica per interpretare correttamente i valori assegnati ai vari controlli. Invece con ASP, PHP o altre tecnologie dinamiche lato server si possono ottenere direttamente i valori delle variabili corrispondenti ai controlli del form. Ad esempio in ASP/JS, usando `Request.QueryString(num)` si avrà come risposta `12A`, usando invece `Request.QueryString(sede)` la risposta sarà `Sede Centrale` (già convertita come normale stringa, ossia il carattere `+` usato per trasmettere uno spazio è stato già rimpiazzato).

5.7 Esempio trasmissione di un form con POST

Consideriamo lo stesso form già visto in precedenza (Figura 5.7), assumendo che l'attributo `method` abbia come valore `post` e che vengano inseriti gli stessi valori (`12A` e `Sede Centrale`) nei controlli del form. In altre parole la prima riga dell'esempio diventa:

```
<form method="get" action="/cgi/insaula">
```

Quando viene premuto il pulsante Invia, la URI presente nella barra degli indirizzi del browser sarà uguale a quella specificata nell'attributo `action`, senz'alcuna aggiunta:

```
http://www.server.it/cgi/insaula
```

A livello di canale TCP, la richiesta HTTP questa volta conterrà un body (per trasmettere i valori dei controlli) e di conseguenza l'header HTTP avrà in più gli header `Content-Type` e `Content-Length`:

```
POST /cgi/insaula HTTP/1.1
Host: www.server.it
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Referrer: http://www.server.it/formaula.html
Accept-Language: it
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.0)
Connection: Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 23

num=12A&sede=Sede+Centrale
```

Lato server, la variabile di ambiente `QUERY_STRING` non conterrà alcun valore poiché i dati del form sono inseriti nel body che viene passato alla'applicazione CGI come standard input.

L'applicazione dovrà quindi leggere i dati da standard input ed effettuarne la decodifica. Se invece usiamo ASP, tramite `Request.form(num)` e `Request.form(sede)` possiamo estrarre i valori, già decodificati, dei valori assegnati a questi due controlli.

5.8 Esempio trasmissione di un form con multipart

Consideriamo lo stesso form già visto in precedenza (Figura 5.7), assumendo che l'attributo `method` abbia come valore `post` e che venga specificata la codifica `multipart` per la trasmsisione dei valori dei controlli del form. In altre parole la prima riga dell'esempio diventa:

```
<form method="post" action="/cgi/insaula" enctype="multipart/form-data">
```

La URI nella barra degli indirizzi del browser non cambia rispetto alla trasmissione con `POST`:

```
http://www.server.it/cgi/insaula
```

Invece a livello di canale TCP cambia sia l'header HTTP, per specificare la codifica richiesta e la diversa dimensione del body, sia il body, che ora è strutturato secondo il formato MIME, con le varie parti separate dalla stringa specificat come parametro `boundary`:

```
POST /cgi/insaula HTTP/1.1
Host: www.server.it
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Referrer: http://www.server.it/formaula.html
Accept-Language: it
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.0)
Connection: Connection: Keep-Alive
Content-Type: multipart/form-data; boundary=AaBbCc
Content-Length: 145

--AaBbCc
Content-Disposition: form-data; name="num"

12A
--AaBbCc
Content-Disposition: form-data; name="sede"

Sede Centrale
--AaBbCc--
```

Si noti che con questo tipo di codifica i valori dei controlli non vengono codificati perché non fanno più parte della URI ma del body HTTP (che permette la trasmissione di qualunque carattere con codifica a 8 bit).

5.9 I campi vuoti in un form

Ad eccezione dei controlli di tipo select e radio, tutti gli altri possono non trasmettere dati. Un campo di testo non riempito invierebbe solamente il nome del controllo mentre nel caso

di una checkbox vuota non verrebbe trasmesso neanche il nome del controllo. Le applicazioni che ricevono input da un form devono saper trattare tutti questi casi.

Consideriamo il seguente esempio di un form per introdurre i dati di una carta di credito:

```
<form name="sample" method="get"
   action="http://www.negozio.it/pagamento.asp">
  <p>Carta di credito: <input type="text" name="cardno" size="16"></p>
  <p>Tipo:
    MasterCard <input type="radio" name="cc" value="mastercard">
    Visa <input type="radio" name="cc" value="visacard">
  </p>
  <p><input type="submit"> <input type="reset"></p>
</form>
```

In base all'input dell'utente, sono possibili i seguenti due casi:

```
cardno=123456789012345&cc=visa      (specificato il numero di carta ed il tipo Visa)
cardno=&cc=mastercard               (non specificato né il numero di carta né il tipo)
```

In quest'altro esempio abbiamo un form con un campo di testo per inserire un cognome e delle checkbox per indicare gli eventuali hobby:

```
<form name="sample" method="get"
   action="http://www.amici.it/persona.asp">
  <p>cognome: <input type="text" name="cogn" size="30"></p>
  <p>hobby:
    <ul>
    <li>pesca <input type="checkbox" name="cb_pesca"></li>
    <li>sci <input type="checkbox" name="cb_sci"></li>
    </ul>
  </p>
  <p><input type="submit"> <input type="reset"></p>
</form>
```

In base all'input dell'utente, sono possibili vari casi tra cui i seguenti:

```
cogn=                                  (cognome ed hobby non specificati)
cogn=De+Rossi                          (inserito il cognome ma non specificati hobby)
cogn=De+Rossi&cb_pesca=on              (inserito il cognome e selezionato un hobby)
cogn=De+Rossi&cb_pesca=on&cb_sci=on    (inserito il cognome e selezionati due hobby)
```

5.10 Upload di un file

Questo è l'unico caso in cui si può, in qualche modo, interagire con i file locali usando HTML 4. I file possono essere solamente letti e trasmessi al server. La forma esatta del controllo dal punto di vista grafico dipende dal browser, ma solitamente contiene due elementi:

- un campo di testo per inserire direttamente il nome del file;
- un pulsante per attivare un'interfaccia grafica che permette all'utente di navigare all'interno del file system e selezionare graficamente il file da trasmettere.

Quando in un form compare almeno un controllo di tipo file, si deve obbligatoriamente usare il metodo POST ed il formato multipart, in caso contrario si genererà un errore e non avverrà la comunicazione HTTP. Con questa configurazione tutti i dati del form verranno trasmessi come parti di un messaggio MIME.

Consideriamo la seguente pagina esempio, che permette l'upload di un file da stampare. Il form contiene un controllo di tipo file per scegliere il file da stampare, un campo di testo per inserire il numero di copie da stampare ed il classico pulsante di Submit.

```
<form action="/cgi/fileprint" enctype="multipart/form-data"
   method="post">
  <p>File da stampare: <input type="file" name="myfile"></p>
  <p>Numero di copie: <input type="text" name="ncopie" size="2"></p>
  <p><input type="submit" value="Stampa"></p>
</form>
```

Ipotizziamo che l'utente scelga di stampare due copie del file `orario.txt` che è di tipo testo e contiene solo la seguente riga:

```
8:30-12:30 aula 12
```

Il trasferimento del file sul canale HTTP avverrebbe nel seguente modo:

```
POST /cgi/fileprint HTTP/1.1
Host: www.server.it
Content-Type: multipart/form-data; boundary=AaBb
Content-Length: Length: 199

--AaBb
Content-Disposition: form-data; name="myfile"; filename="orario.txt"
Content-Type: text/plain

8:30-12:30 aula 12
--AaBb
Content-Disposition: form-data; name="ncopie"

3
--AaBb--
```

Si noti che, grazie al tipo multipart, è possibile mischiare la trasmissione di un file con altri dati (in questo caso, il numero di copie da stampare) perché il body è diviso in parti. Inoltre anche nel caso in cui il client avesse scelto di stampare un file binario (ad esempio un PDF) questo sarebbe stato inserito nel body perché la trsamissione HTTP è pulita a 8 bit.

5.11 Confronto tra i metodi GET e POST per i form

Il metodo GET ha una serie di vantaggi:

- permette di fare caching della pagina di risposta, infatti il browser può associare la risposta ai parametri che si forniscono in input dato che questi sono trasmessi nella URI;

- salvando la URI completa coi parametri del form, si può creare un bookmark o creare/scambiare un link alla risorsa, che non punterà alla pagina generica ma a quella specifica generata inviando anche i valori prescelti dei controlli;
- è più facile effettuare il debug di un'applicazione web perché i parametri compaiono nella barra degli indirizzi e sono quindi visibili all'utente che può accorgersi di eventuali errori.

Il metodo GET presenta però anche alcune criticità:

- I server tengono traccia di tutte le URI che sono state richieste, ciò significa che nel log del server compariranno anche i valori inseriti nei controlli del form. Se quindi è presente un campo password o dati sensibili, questi valori saranno visibili nel file del log, creando un problema di privacy o di sicurezza.
- Alcuni server limitano la lunghezza della query string a 256 caratteri se questa è all'interno della URI, quindi se un form ha tanti controlli o se l'utente ha inserito valori molto lunghi si corre il rischio che il form non funzioni (è quindi meglio utilizzare GET quando sono presenti pochi controlli con valori di dimensione limitata).

Il metodo POST non pone invece limiti al numero dei controlli ed alla dimensione dei valori trasmessi; può quindi essere usato per form di qualunque complessità. Inoltre nei log compare solo la URI corrispondente alla `action` del form e quindi i valori dei controlli non vengono registrati sul server; non ci sono quindi problemi di privacy o sicurezza. D'altra parte, effettuare il debug di un form trasmesso con POST è decisamente più complesso, perché occorre dotarsi di strumenti che intercettano e visualizzano le comunicazioni HTTP (sulla rete, sul client o sul server). Inoltre salvare in cache il risultato generato dalla trasmissione un form con POST non ha senso perché non vengono salvati i valori dei controlli che hanno generato tale pagina. Allo stesso modo, creare un bookmark per tale pagina non ha senso perché la URI non contiene i valori dei controlli usati; ha quindi senso solo salvare come bookmark la URI della pagina che contiene il form.

In definitiva, soppesando vantaggi e svantaggi, in generale si preferisce il metodo POST. L'apparente vantaggio di GET relativo a cache e bookmark risulta spesso un'arma a doppio taglio perché la risposta salvata in cache può essere errata se dipende dal tempo in cui è stata effettuata la richiesta (si pensi al caso il cui sia stato salvato in cache il risultato della ricerca dei treni da Torino a Milano in partenza "entro un'ora"). Per la fase di debug può essere utile usare GET, ma solo sino a quando l'applicazione non è stata corretta.

Capitolo 6

Il linguaggio Javascript

Capitolo 7

Modello DOM e sicurezza Javascript

7.1 Il modello DOM

Il *DOM* (*Document Object Model*) è un formalismo per rappresentare la struttura dati di un generico documento, quale, ad esempio, una pagina HTML; è considerato uno standard dal W3C. Tramite esso gli script client-side (es. JS o VBS) possono interagire con gli elementi della pagina. Il DOM al quale si fa qui riferimento è il "DOM livello 1" del 1998.

7.1.1 Accedere agli oggetti DOM

La figura 7.1 mostra una pagina web composta da vari elementi: si può notare la presenza di due immagini, di un form (con due campi di testo e un pulsante) e di un link. Gli elementi che compongono la pagina web sono organizzati in *array* (es. `images`, `links`, `forms`). Si può accedere ai singoli elementi specificando l'array di cui fanno parte e il loro indice all'interno di esso (es. `images[0]`, `images[1]`, `forms[0]`, `links[0]`); si può anche accedere singolarmente ai vari elementi del form, individuando il sottoelemento desiderato (es. nella figura 7.1 `forms[0].elements[1]` rappresenta il controllo di testo usato per inserire l'indirizzo e-mail).

La figura 7.2 mostra la gerarchia completa degli oggeti DOM della pagine in figura 7.1 ; la sua conoscenza permette di accedere ai valori contenuti nei vari elementi, ad esempio memorizzandoli in una variabile ed usandoli in un pop-up come col seguente script:

```
<script type="text/javascript">
  name= document.forms[0].elements[0].value
  alert("Ciao " + name)
</script>
```

Sebbene sia possibile accedere ad un elemento usando il riferimento gerarchico sopra descritto, generalmente si preferisce identificarlo tramite l'attributo `name` (disponibile solo per alcuni tag) o `id` (disponibile per tutti i tag). Questo perché se nella pagina venissero aggiunti nuovi componenti, l'indice di uno specifico elemento negli array DOM cambierebbe mentre il suo riferimento tramite uno specifico identificativo no (a meno che tale identificativo non sia volontariamente cambiato dal programmatore web).

7.1.2 Gerarchia degli oggetti DOM

Il modello DOM ci permette di accedere anche a risorse esterne al documento. Come si può osservare dallo schema in figura 7.3, il `document` è solo una parte della `window`: oltre ad

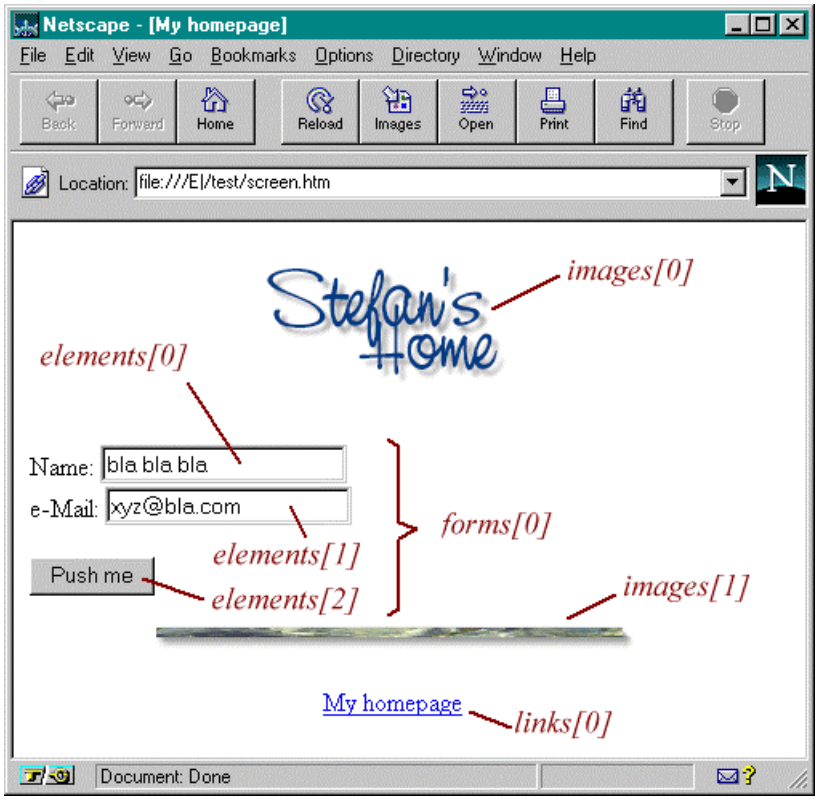

Figura 7.1: Componenti DOM di una pagina.

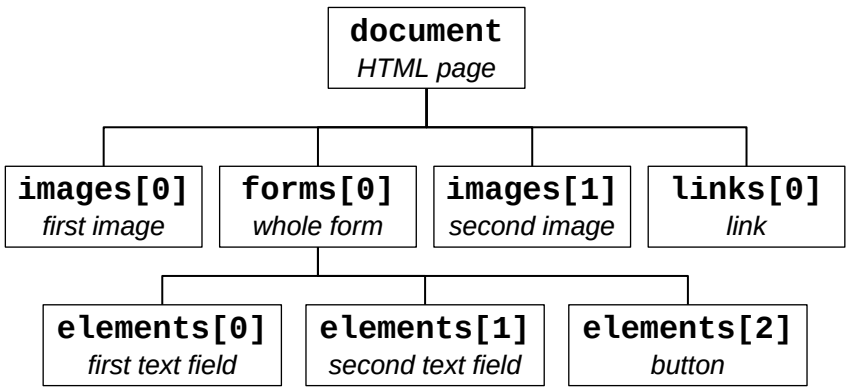

Figura 7.2: Gerarchia DOM della pagina in figura 7.1.

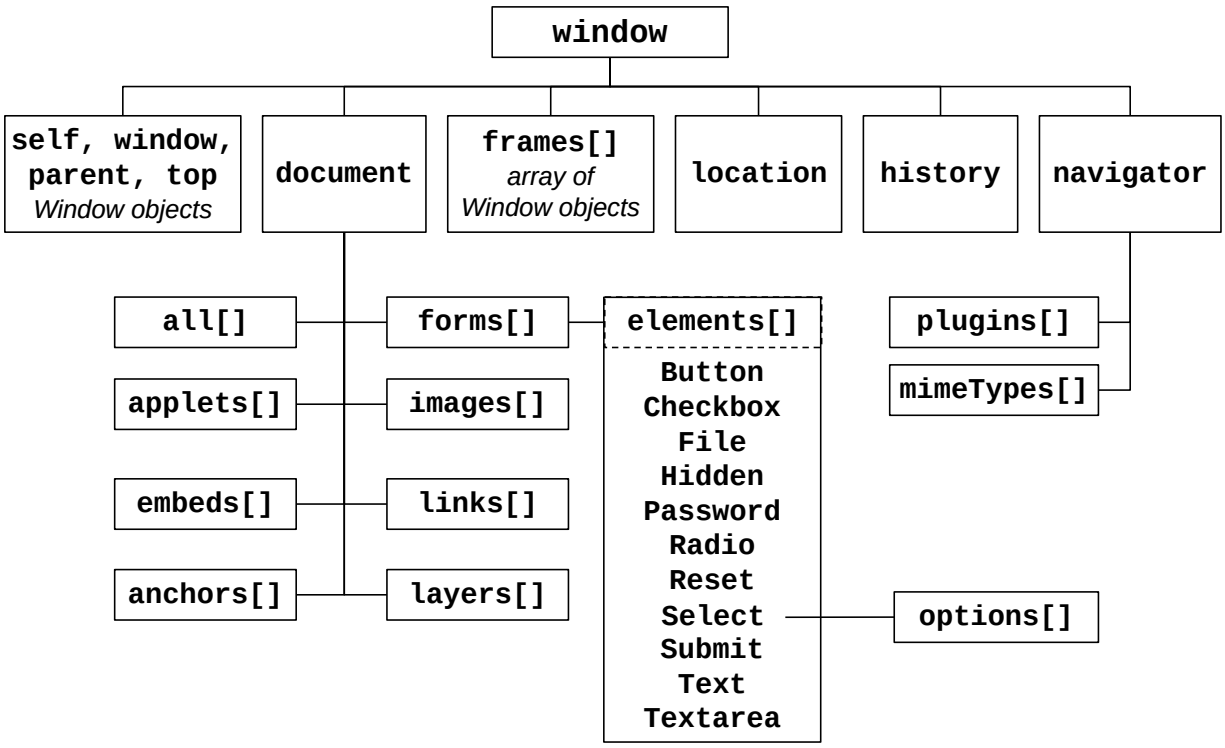

Figura 7.3: Gerarchia DOM (in JavaScript).

esso il modello permette di accedere ad altri elementi quali la `history` (cioè la cronologia di navigazione), l'indirizzo (tramite `window.location`) che può anche essere modificato per far visitare un'altra pagina, il `background` (al quale si può cambiare colore) e, se la pagina è organizzata in frame, si può usare ad esempio `window.parent` (per accedere al frame genitore), `window.self` (se stesso) e `window.top` (per accedere al frame più in alto).

7.2 Sicurezza degli script client-side

Poiché uno script client-side viene eseguito sul browser è necessario tenere in considerazione che l'utente ha la possibilità di accedervi e modificarlo; è bene pertanto non inserirvi dati riservati che l'utente non dovrebbe poter conoscere (ad esempio le risposte al test che gli si sta proponendo o la password necessaria per accadere ad una funzione riservata sul server).

Bisogna anche valutare l'eventualità che qualcuno tenti di usare uno script client-side per eseguire istruzioni maligne: a questo proposito va detto che, usando HTML 4, lo script può agire solo all'interno del browser, mentre con HTML 5 ha un accesso molto più ampio alle risorse del sistema. Tra le possibili insidie che si possono avere anche con HTML 4 si ha il rischio che vengano modificate delle impostazione del browser, la possibilità che vengano letti e trasmessi a terzi informazioni destinate ad altri, il pericolo di attacchi "denial of service" (tali cioè da rendere non più utilizzabile il sistema). La figura 7.4 contiene un esempio di codice maligno per l'esecuzione di un attacco DoS. Il programma Javascript ha comunque accesso ad un set di comandi ristretto; in particolare non può:

- leggere o modificare file sul sistema client;
- lanciare in esecuzione altri programmi;

```
<html>
  <!-- attacco.html -->
  <head>
    <title>DoS attack - apre infinite finestre </title>
  </head>
  <body>
    <script type="text/javascript">
      window.open ("attacco.html");
    </script>
  </body>
</html>
```

Figura 7.4: Esempio di pagina contenente codice maligno JS.

- usare primitive per collegamenti in rete (può però aprire nuove finestre del browser e far scaricare ad esso contenuti web);
- eseguire funzioni *multithreading* (per cui non può, ad esempio, far partire più copie di sé senza aprire altre finestre).

Capitolo 8

Il linguaggio CSS

8.1 HTML e stili

I file HTML, oltre a contenere il testo vero e proprio del documento, permettono di definire due aspetti distinti di un documento: la sua organizzazione logica (es. sezioni, paragrafi, liste) e la sua presentazione (es. colori, grassetto).

Inizialmente i tag erano stati pensati per indicare esclusivamente l'organizzazione logica, delegando al browser gli aspetti di presentazione: in questo modo però uno stesso documento poteva essere visualizzato in modo diverso su browser differenti, ad esempio per la dimensione dei caratteri ed il colore dei titoli. Per permettere al programmatore web di definire più puntualmente gli aspetti di presentazione del documento, vennero successivamente i tag dedicati alla formattazione, come il tag `<font>`. ma il risultato fu negativo: infatti il sito web è sviluppato e aggiornato da più persone, ognuna delle quali segue le proprie regole provocando così la perdità dell'uniformità dell'aspetto grafico del sito. Per far fronte a questa problematica si è tornati alle origini, eliminando dall'HTML i tag di formattazione delegando la formattazione a file esterni: i Cascading Style Sheet. Nell'HTML che usiamo oggi è possibile utilizzare sia il CSS, sia i tag HTML per la formattazione visuale. Tuttavia quest'ultimi sono deprecati e non esistono più in HTML 5 e XHTML.

8.2 CSS

8.2.1 Storia

La prima versione del CSS è del 1996. Recentemente, nel giugno 2011, è stato rilasciato CSS2.1 aggiungendo oltre 70 funzionalità alle 50 iniziali. Attualmente è in sviluppo il CSS3 che noi non trattiamo perché è ancora in fase di definizione.

8.2.2 Funzionalità

Le funzionalità citate precedentemente tengono in considerazione che le pagine web possano essere visualizzate non solo su personal computer ma anche su diverse periferiche (es. periferiche per la lettura braille) o possano essere stampate.

8.2.3 Formato ed integrazione con HTML

Il codice che serve a definire il formato e il layout della pagina, deve essere scritto in ASCII puro e solitamente viene salvato in un file di testo esterno alla pagina. Il file deve avere estensione `".css"`.

Il file CSS viene richiamato in una pagina tramite un `link` nell'header del file HTML, specificando il formato della risorsa che si importa (`MIME`). Per esempio:

```
<link rel="stylesheet" type="text/css" href="polito.css">
```

C'è la possibilità di specificare che certe proprietà vengano applicate solo se il file è visualizzato da una certa periferica usando il parametro `media`. Il media può essere specificato come parametro del link oppure possono esserne specificati tanti divisi da una virgola.

```
<link rel="stylesheet" type="text/css"
      href="polito_stampa_A4.css" media="print">
```

Lo stesso file CSS può essere caricato in diversi file HTML, rendendo uniforme l'aspetto visivo del sito e lasciando allo sviluppatore solo il compito di occuparsi del contenuto e delle logica delle singole pagine.

Nonostante sia deprecato, è tuttavia possibile inserire specifiche CSS all'interno di una singola pagina. Questo si può fare con il tag `style` al cui interno si possono inserire tutte le specifiche CSS desiderate, come nel seguente esempio:

```
<head>
   <style type="text/css">
      body { background-color : gray }
   </style>
   . . .
</head>
```

Inoltre il tag `<style>` è un'estensione. HTML 3 non prevede (dunque non capisce) questo tag. Quando nel codice è presente `<style>` viene saltato solo il tag ma vengono comunque lette le righe di CSS. Per evitare questo rischio, si può utilizzare dentro a style un commento dell'HTML: così se venisse saltato il tag, verrebbe commentato tutto il CSS (privo di tag). Se invece usiamo il file esterno, chi non riesce a leggerlo non lo include e si evitano inconvenienti.

E' deprecato l'uso della formattazione per i singoli tag (ad esempio XHTML non supporta più il tag `<b>`). In sostituzione è possibile utilizzare l'attributo `style` applicato al singolo tag, come nel seguente esempio:

```
<span style="font-weight:bold">Testo in grassetto</span>
```

Anche questa opzione è però sconsigliata perché (a) non fornisce informazioni logiche circa il motivo per cui un determinato testo viene presentato in grassetto e (b) non permette di cambiare rapidamente la formattazione di tutti questi elementi nel caso si decidesse di presentarli con uno specifico colore invece che in grassetto. Meglio sarebbe definire una `class` al nome e gestire la proprietà `bold` da foglio esterno assegnandola a tutti i testi assimilabili a quella classe.

Nel caso in cui vi siano definite regole non compatibili, la specifica sul tag ha priorità su ciò che c'è scritto nell'`head`, il quale a sua volta "vince" sulle specifiche del file esterno.

8.2.4 Sintassi

La sintassi CSS è costituita da tre componenti principali:

- il selettore che indica a quale elemento HTML è riferita la specifica;
- l'identificativo della proprietà;
- il valore della proprietà

secondo il seguente formato:

selettore `{` *proprietà* `:` *valore* `;` `...` `}`

Per esempio scrivendo:

```
h1 {color: green}
```

si richiede che tutti i titoli di primo livello siano presentati col colore verde.

Importazione CSS

Esiste la possibilità di usare la sintassi `@import` per importare le regole CSS contenute in un altro file, per esempio:

```
@import url("poli_general.css");
```

Inoltre è possibile specificare l'importazione condizionata ad uno specifico media-type indicandolo dopo la URL, come nel seguente esempio:

```
@import url("poli_stampa.css") print;
```

Commenti

In CSS i commenti si scrivono `/* */` (seguono la sintassi del linguaggio C). Possono essere inseriti anche all'interno di una specifica.

Selettori

L'HTML organizza il documento gerarchicamente: gli elementi contenuti nel `body` sono detti figli (e tra loro fratelli) (Figura 8.1).
Le proprietà assegnate dal CSS (per questo definito "a cascata") vengono applicate a tutti i suoi discendenti. Però se su un figlio viene applicata una nuova proprietà più specifica, quest'ultima ha la priorità. Un'eccezione è `background` che di default è trasparente.

Per definire le proprietà di un singolo elemento si può usare `#` seguito dal il nome (unico) dell'id:

```
#footer { font-style: italic }
```

Oppure è possibile assegnare una proprietà ad una classe utilizzando `.` seguito dal nome della classe:

```
...
<body>
  <h1>I Re di Roma</h1>
  <p>I primi tre Re di Roma:</p>
  <ul>
    <li>Romolo</li>
    <li>Numa Pompilio</li>
    <li>Tullo Ostilio</li>
  </ul>
</body>
```

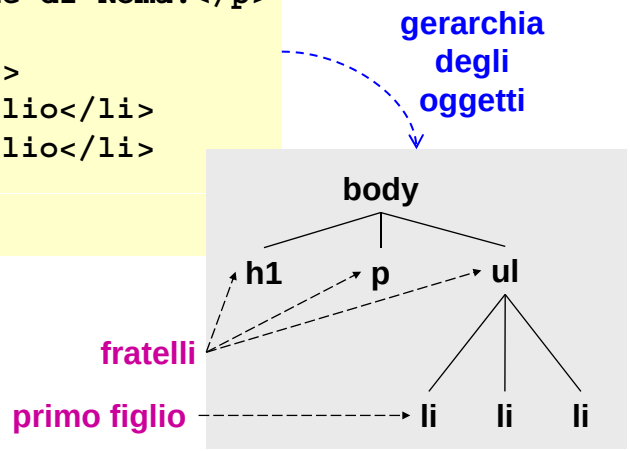

Figura 8.1: Esempio di gerarchia degli oggetti CSS.

selettore	*esempio*	*descrizione dell'esempio*
`:link`	`a:link`	seleziona i link mai visitati
`:visited`	`a:visited`	seleziona i link già visitati
`:active`	`a:active`	seleziona i link attivi
`:hover`	`a:hover`	seleziona i link su cui vi è sovrapposto il cursore
`:focus`	`input:focus`	seleziona il controllo di input di form che ha il focus
`:first-letter`	`p:first-letter`	Seleziona la prima lettera di ogni elemento `<p>`
`:first-line`	`p:first-line`	Seleziona la prima riga di ogni elemento `<p>`
`:first-child`	`p:first-child`	Seleziona ogni elemento `<p>` che è il primo figlio del suo genitore
`:before`	`p:before`	Inserisce del contenuto prima dell'elemento
`:after`	`p:after`	Inserisce del contenuto dopo l'elemento
`:lang(...)`	`p:lang(it)`	seleziona ogni elemento `<p>` con lingua uguale a `it`

Tabella 8.1: Elenco pseudo-classi CSS e relativi esempi.

```
.warning { color: red }
```

Il tag name può essere anche seguito da `:` e da una pseudo-classe. Le pseudo classi fanno riferimento a degli eventi: quando questi si manifestano, viene applicata la proprietà al selettore indicato. Nel seguente esempio, quando il cursore si sovrappone ad una qualunque àncora `<a>`, lo sfondo di tale area assumerà il colore definito come `aqua`:

```
a:hover { background: aqua }
```

La tabella 8.1 elenca le pseudo-classi CSS e presenta semplici esempi.

Per indicare più selettori, questi possono essere separati da una virgola e dunque verranno applicate a tutti le stesse proprietà.

Selettori gerarchici

...

Selettori condizionali

...

Specifica delle lunghezze

Nel CSS per indicare delle lunghezze abbiamo la possibilità di indicare delle lunghezze relative o delle lunghezze assolute (le prime indiacate per uno schermo, le seconde per la stampa). Le lunghezze relative possono essere espresse come:

- `em` = altezza della lettera M nel font corrente
- `exex` = altezza della lettera altezza della lettera xx nel nel font corrente font corrente
- `px` = dimensione di un pixel

Le lunghezze assolute:

- `in` = inch (25.4 mm)
- `cm` o `mm`
- `pt` = punto tipografico (1/72 di inch)
- `pc` = pica (12 pt)

Specifica dei colori

Per quanto riguarda i colori, si possono utilizzare:

- i colori predefiniti che erano validi anche per l'HTML:

  `Black, White, Gray, Silver, Yellow, Yellow, Red, Purple, Fuchsia, Maroon, Red, Purple, Fuchsia, Maroon, Green, Lime, Olive, Aqua, Teal, Blue, Navy`

- valori delle componenti RGB (assoluti o percentuali), ad esempio il colore `red` può anche essere indicato nei seguenti modi:

  ```
  rgb (255, 0, 0)}
  rgb (100%, 0%, 0%)
  ```

- valori delle componenti RGB in esadecimale (quando ci sono due cifre esadecimali uguali, se ne può scrivere una sola), ad esempio il colore `red` può anche essere indicato nei seguenti modi:

  ```
  #ff0000
  #f00 /* abbreviazione per due cifre hex uguali */
  ```

Si noti che la sintassi per i colori è diversa in CSS rispetto a HTML.

8.2.5 Validazione

La correttezza del file CSS deve essere validata. Nella seguente pagina è possibile usufruire di un servizio di validazione automatico:

http://jigsaw.w3.org/css-validator/

8.2.6 Lo sfondo (background)

Con la proprietà `background-color` possiamo scegliere il colore dello sfondo. I valori assegnabili sono:

- *colore* indica il colore dello sfondo fornendo il nome di un colore tra quelli predefiniti (vedi tabella XXX) oppure indicandone il codice RGB esadecimale, decimale o percentuale;
- `transparent` indica uno sfondo trasparente, ossia che lascia vedere lo sfondo dell'elemento sottostante; Risulta essere l'impostazione di default. Di fatto è come se non ci fosse alcun background.
- `inherit` richiede che il valore di questa proprietà sia uguale a quello del suo genitore.

Il valore `transparent` è quello di default.

Con `background-image` possiamo inserire un'immagine, fornendone l'URI con la seguente sintassi:

`url("`*URI-della-immagine* `")`

Quando si specifica un'immagine di sfondo è altamente consigliato specificare anche un colore di sfondo, da usarsi nel caso l'immagine non venga caricata (per problemi di rete o di formato) oppure come contorno (se l'immagine non riempie tutta la finestra) o per colorare le parti trasparenti dell'immegine (che mostrano appunto il colore sottostante).

Con `background-repeat` possiamo decidere se vogliamo che l'immagine venga ripetuta per coprire tutto lo sfondo della finestra, specificando uno dei seguenti valori:

- `repeat` richiede che l'immagine venga ripetuta sia in verticale sia in orizzontale, sino a coprire tutto lo sfondo;
- `repeat-x` richiede che l'immagine venga ripetuta solo in orizzontale;
- `repeat-y` richiede che l'immagine venga ripetuta solo in verticale;
- `no-repeat` richiede che l'immagine non venga ripetuta (ossia compaia una sola volta, posizionata secondo il valore indicato dalla proprietà `background-position`);
- `inherit` richiede che il valore di questa proprietà sia uguale a quello del suo genitore.

Con la proprietà `background-position` possiamo posizionare l'immagine in una determinata posizione specificando due valori, la posizione orizzontale e quella verticale:

`background-position:` *posizione-orizzontale posizione-verticale*

Ciascuna posizione può essere indicata tramite una distanza (misurata dall'angolo in alto a sinistra della pagina), una percentuale (della dimensione della pagina) oppure una keyword, scelta tra le seguenti:

- la posizione orizzontale si può specificare tramite `left`, `center` e `right` (corrispondenti rispettivamente a "sbandierata a sinistra", "centrato" e "sbandierata a destra")
- la posizione verticale si può specificare tramite `top`, `center` e `bottom` (corrispondenti rispettivamente a "appoggiata al bordo superiore", "centrata" e "appoggiata al bordo inferiore")

Con la proprietà `background-attachment` possiamo specificare se l'immagine rimarrà in posizione fissa o seguirà lo scorrimento del contenuto, specificando uno dei seguenti valori:

- `fixed` richiede che l'immagine rimanga ferma anche quando scorriamo il contenuto;
- `scroll` richiede che l'immagine segua il contenuto quando questo viene fatto scorrere;
- `inherit` richiede che il valore di questa proprietà sia uguale a quello del suo genitore.

È possibile richiedere tutti i valori dello sfondo in forma sintetica, specificandoli in qualsiasi ordine. Ad esempio se volessimo un'immagine centrata, non ripetuta, con sfondo grigio e che segua il contenuto, possiamo inserire:

```
background: url("polito.jpg") scroll center center gray no-repeat
```

8.2.7 Proprietà del testo (Text properties)

Con la proprietà `color` possiamo scegliere il colore del carattere con cui stiamo scrivendo. Possiamo fornire come valore il nome di un colore tra quelli predefiniti (vedi tabella XXX) oppure indicare il codice RGB esadecimale, decimale o percentuale;

Con la proprietà `text-align` possiamo scegliere come il nostro testo verra allineato rispetto al foglio. I valori assegnabili sono:

- `left` permette di allineare il testo lungo la sinistra del foglio/contenitore;
- `right` permette di allineare il testo lungo la destra del foglio/contenitore;
- `center` permette di scrivere il testo centrato nel foglio/contenitore;
- `justify` permette di scrivere il testo giustificato, ovvero allineato sia a destra che a sinistra con i margini del foglio/contenitore.

Con la proprietà `text-transform` possiamo trasformare la prima lettera di ogni parola o tutto il testo in maiuscolo o in minuscolo. I valori assegnabili sono:

- `none` lascia il testo invariato.è il valore di default;
- `capitalize` trasforma la prima lettera di ogni parola in maiuscola;
- `uppercase` trasforma tutti i caratteri in maiuscoli;

- `lowercase` trasforma tutte i caratteri in minuscoli;
- `inherit` richiede che il valore di questa proprietà sia uguale a quello del suo genitore.

Con la proprietà `text-decoration` possiamo possiamo applicare alcune modifiche allo stile del testo. I valori assegnabili sono:

- `none` lascia il testo invariato. È il valore di default;
- `underline` richiede una linea sotto il testo;
- `overline` richiede una linea sopra il testo;
- `linetrought` richiede una linea che tagli il testo in mezzo orizzontalmente (come se fosse cancellato);
- `blink` richiede un testo lampeggiante.

La proprietà `text-indent` specifica il rientro della prima riga. Il suo valore può essere espresso sia come una lunghezza sia come una percentuale della pagina/contenitore. Sono possibili anche valori negativi.

Con la proprietà `line-height` possiamo specificare l'altezza della linea. I valori assegnabili sono:

- `normal` lascia la dimensione della riga invariata. È il valore di default;
- *fattore moltiplicativo* con il quale è possibile specificare l'altezza della linea. Esso viene ereditato tale e quale dai figli;
- *lunghezza* con la quale possiamo richiedere una determinata altezza della riga, esprimendola con le unità di misure gia viste precedentemente;
- *percentuale* con la quale possiamo esprimere l'altezza della riga. Nel caso di valore percentuale viene ereditato il valore numerico risultante, con il difetto che nel caso un figlio abbia la dimensione del carattere che eccede rispetto alla dimensione della riga, quest'ultima non viene ridimensionata lasciando i caratteri tagliati.

8.2.8 Proprietà del carattere (Font properties)

Con la proprietà `font-style` possiamo possiamo applicare alcune modifiche al formato del testo. I valori assegnabili sono:

- `normal` lascia il testo normale. È il valore di default;
- `italic` richiede un testo scritto in itallico (leggermente inclinato);
- `oblique` sottilmente diverso dall'itallico.

Con la proprietà `font-weight` possiamo possiamo applicare alcune modifiche al formato del testo. I valori assegnabili sono:

- `normal` lascia il testo normale. È il valore di default;

- `bold` richiede un testo in grassetto;
- *valore numerico da 1 a 1000* permette di specificare la saturazione del nero. 400 è il valore normale, 700 e il grassetto di default.

Con la proprietà `font-variant` possiamo possiamo applicare lo stile "maiuscoletto". Presenta solo due valori assegnabili:

- `normal` lascia il testo normale. È il valore di default;
- `small-caps` richiede un testo in maiuscoletto;

Con la proprietà `font-stretch` possiamo rendere il testo più largo o più stretto. I valori assegnabili sono:

- `ultra/extra-condensed` rende il testo più stretto. Ultra avrà più effetto di extra;
- `condensed` ha un effetto identico al valore sopra, semplicemente l'effetto sarà meno marcato;
- `normal` lascia il testo invariato. È il valore di default;
- `expanded` rende il testo più largo.
- `ultra/extra-expanded` ha un effetto identico al valore sopra, ma ultra e extra ne amplificano gli effetti.

Questa proprietà è raramente disponibile su UA e nessun browser la supporta.

Con la proprietà `font-size` possiamo specificare le dimensioni del carattere. La dimensione può essere espresa come:

- *valore assoluto* utilizzando le unità di misura viste in precedenza (pt, mm...);
- *valore relativo* utilizziando valori relativi come em o ex;
- *percentuale* utilizzando valori percentuali;
- *scala assoluta* utilizzando una scala da 1 a 7 in cui la dimensione nomale è 3;
- *scala relativa* utilizzando un indicazione +/- N;
- *valori predefiniti* utilizzando valori di una scala predefinita (`xx-small, x-small, small, medium, large, x-large, xx-large`).

I valori relativi e percentuali si riferiscono al font dell'elemento genitore e sono da preferirsi rispetto ai valori assoluti.

Con la proprietà `font-family` possiamo scegliere il tipo di font (carattere) utilizzato. È possibile specificare una lista di font separati da una virgola, l'UA cercherà di utilizzare il primo disponibile partendo da sinistra. Qui sotto sono elencate le cinque famiglie di caratteri disponibili.

- `Serif`: sono i cosiddetti caratteri "con grazie", ovvero con la presenza di lievi abbellimenti alla fine dei tratti delle lettere. Questa famiglia di caratteri è sconsigliata per la visualizzazione su schermo per via della scarsa risoluzione di quest'ultimi. È invece consigliata nelle pagine pensate per essere stampate, vista l'alta risoluzione delle stampanti in grado di rappresentare correttamente i caratteri. Alcuni font appartenenti a questa famiglia: `Times, Times new roman, Palatino, Georgia` (progettato per il web), `Garamond`.

- `Sans-serif`: sono i cosiddetti caratteri "senza grazie", ovvero senza alcun tipo di abbellimento. Hanno un aspetto molto "pulito" e minimalista. Questi caratteri sono invece consigliati per la visualizzazione poichè essendo privi di abbellimenti vengono rappresentati perfettamente su qualsiasi tipo di schermo. Alcuni font appartenenti a questa famiglia: `Arial, Geneva, Lucida Sans, Helvetica, Verdana` (progettato per il web), `Trebuchet`.

- `Monospace`: sono i cosidetti caratteri "monospaziati", ovvero nei quali ogni carattere utilizza lo stesso spazio indifferentemente dalla sua dimensione (esattemente come nella macchina da scrivere). Alcuni font appartenenti a questa famiglia: `Courier, Courier new, Fixed, Lucida Console, Monaco`

- `Cursive`: vi appartengono tutti quei font che hanno uno stile simile alla scrittura a mano, solitamente sono inclinati e spesso hanno tratti che si estendono in ornamenti. Vista la grande varieta di stili presenti in questa famiglia è necessario prestare attenzione quando si utilizzano. Alcuni font appartenenti a questa famiglia: `Comic sans MS, Florence, Lucida Handwriting, Zapf Chancery`

- `Fantasy`: vi appartengono in generale tutti i caratteri non catalogabili nelle famiglie precedenti. I font appartenenti a questa famiglia hanno spesso uno stile grassetto esagerato o presentano ornamenti eccentrici. Questa famiglia è da utilizzare con parsimonia e mai per grandi porzioni di testo: risulta infatti spesso di difficile lettura e non sempre nel font sono presenti tutti i caratteri accentati. Alcuni font appartenenti a questa famiglia: `Impact` (in assoluto il più presente su Mac, Windows e Linux), `Oldtown, Copperplate, Desdemona`.

Si consiglia, come ultimo elemento della lista specificata in `font-family`, di inserire il nome di una famiglia in modo che se nessuno dei font richiesto è presente, l'UA possa utilizzarne uno che ha a disposizione. È inoltre consigliato inserire come alternative caratteri della stessa famiglia in modo da non cambiare radicalmente lo stile della pagina in base al font disponibile. Un possibile esempio potrebbe essere:

```
font-family: arial, helvetica, verdana, sans-serif
```

8.2.9 Contenitori

Ogni blocco viene visto come un contenitore secondo il modello illustrato in figura 8.2

Posizionamento e dimensioni

È possibile specificare le dimensioni del contenitore. I valori possono essere espressi mediante una lunghezza assoluta, una percentuale (riferita all'elemento genitore) oppure con il valore `inherit` che richiede che il valore di questa proprietà sia uguale a quello del suo genitore.

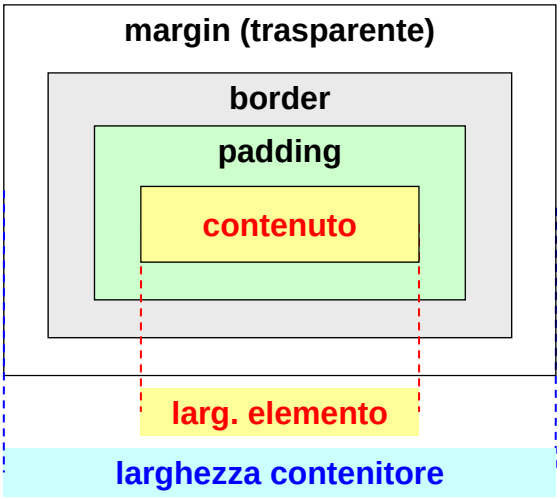

Figura 8.2: Modello dei contenitori in CSS.

- `width` specifica la larghezza del contenuto. Può anche essere impostato in maniera automatica con il valore `auto`;
- `min-width` specifica la larghezza minima del contenuto;
- `max-width` specifica la larghezza massima del contenuto. Di default questa proprietà assume il valore `none`;
- `height` specifica l'altezza del contenuto. Può anche essere impostato in maniera automatica con il valore `auto`;
- `min-height` specifica l'altezza minima del contenuto;
- `max-height` specifica l'altezza massima del contenuto. Di default questa proprietà assume il valore `none`;

Si consiglia di utilizzare anche in questo caso misure relative (in percentuale) e non assolute.

Le dimensioni del margine del contenitore possono essere specificate utilizzando le proprietà:

```
margin-left, margin-right, margin-top, margin-bottom
```

A ciascuna proprietà può essere assegnato un valore percentuale o una lunghezza assoluta. È anche possibile richiederne la regolazione automatica. con il valore `auto` (valore di default).

Analogamente alle dimensioni dei margini, le dimensioni relative al padding del contenitore possono essere specificate utilizzando le proprietà:

```
padding-left, padding-right, padding-top, padding-bottom
```

A ciascuna proprietà può essere assegnato un valore percentuale o una lunghezza assoluta. È anche possibile richiederne la regolazione automatica con il valore `auto` (valore di default).

Possiamo indicare l'allineamento di un contenitore rispetto al genitore utilizzando la proprietà `float`. Sono presenti tre possibili valori:

- `left` richiede un allineamento a sinistra;

- `right` richiede un allineamento a destra;
- `none` non richiede alcun allineamento. Valore di default.

Possiamo inoltre indicare quali lati di un contenitore non possono essere adiacenti ad altri contenitori con la proprietà `clear`. I valori assegnabili sono:

- `left` richiede che non vi siano altri contenitori adiacenti al lato sinistro;
- `right` richiede che non vi siano altri contenitori adiacenti al lato destro;
- `both` richiede che non vi siano altri contenitori adiacenti ad entrambi i lati;
- `none` non impone alcuna limitazione. Valore di default.

Bordi

Con le proprietà `border-top-style, border-right-style, border-bottom-style, border-left-sty` possiamo specificare le caratteritiche del bordo in alto, a destra, in basso e a sinistra. I valori assegnabili sono:

- `none` non inserisce alcun bordo. Valore di default;
- `hidden` il bordo è nascosto, esteticamente uguale a `none`. Trattato diversamente nella risoluzione di conflitti in quanto il bordo, anche se nascosto, è presente;
- `dotted` specifica un bordo punteggiato;
- `dashed` specifica un bordo tratteggiato;
- `solid` specifica un bordo pieno;
- `double` specifica un bordo doppio. La larghezza dei due bordi è uguale alla larghezza del singolo bordo specificata in `border-width`;
- `groove` specifica un bordo scanalato 3D;
- `ridge` specifica un bordo in rilievo scanalato 3D;
- `outset` richiede che la zona con il bordo sia in rilievo rispetto al piano della pagina;
- `inset` richiede che la zona con il bordo sia in dislivello rispetto al piano della pagina.

Con le proprietà `border-top-width, border-right-width, border-bottom-width, border-left-width` possiamo specificare la dimensione del bordo in alto, a destra, in basso e a sinistra. I valori assegnabili sono:

- `thin` specifica un bordo sottile;
- `medium` specifica un bordo medio. Valore di default;
- `thick` specifica un bordo spesso;
- *lunghezza* consente di specificare lo spessore del bordo con un valore assoluto.

Con le proprietà `border-top-color, border-right-color, border-bottom-color, border-left-color` possiamo specificare il colore del bordo in alto, a destra, in basso e a sinistra. I valori assegnabili sono:

- `transparent` specifica un bordo trasparente. Valore di default;
- *colore* consente di specificare il colore del bordo con i colori predefiniti o con un codice RGB;
- `inherit` richiede che il valore di questa proprietà sia uguale a quello del suo genitore.

Forme sintetiche

Esistono quattro forme sintetiche per specificare i margini, padding e le caratteristiche dei bordi del contenitore, utilizzabili come valori delle proprietà `margin`, `padding`, `border-style`, `border-width` e `border-color`.

`m_top_bottom_left_right`

`m_top_bottom m_left_right`

`m_top m_right_left m_bottom`

`m_top m_right m_bottom m_left`

Ad esempio

`border-style: m_dashed_dashed_dotted_dotted`

realizzerà un bordo tratteggiato sopra e sotto, punteggiato a destra e a sinistra.

`border-width: m_thin m_thick m_thin m_thick`

realizzerà un bordo sottile sopra e sotto, spesso a destra e a sinistra.

`border-color: m_red m_red m_transparent m_transparent`

realizzerà un bordo rosso sopra e a destra, trasparente a sotto e a sinistra.

Esiste un ulteriore metodo per specificare le caratteristiche di bordi in maniera sintetica. Possiamo, infatti, utilizzando la sintassi

`border-top/border-bottom/border-left/border-right:` *width style color*

specificare per il bordo alto/basso/sinitro/destro le proprietà spessore, stile e colore. Ad esempio scrivendo

`border-right: thin dotted red`

richiediamo un bordo destro rosso, sottile e punteggiato.

Infine con la sintassi

`border:` *width style color*

possiamo specificare in una sola volta spessore, stile e colore di tutti i bordi del contenitore.

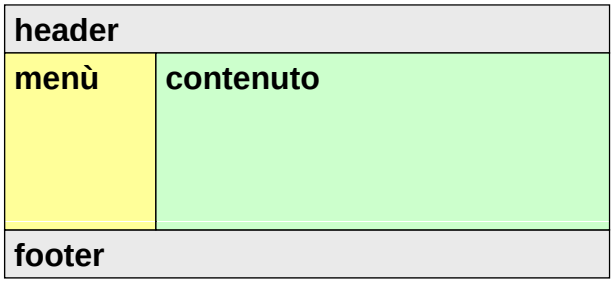

Figura 8.3: Esempio di layout grafico.

8.2.10 Proprietà dei link (Link properties)

Possiamo definire stili differenziati per i link, in base allo stato in cui si trovano.

- `:link` per indicare un link non ancora visitato;
- `:visited` per indicare un link già visitato;
- `:active` per indicare un link sul quale viene fatto click;
- `:hover` per indicare un link puntato dal mouse .

Ad esempio, se vogliamo che il link diventi verde quando il mouse ci passa sopra, dovremo scrivere nel file CSS di riferimento

```
a:hover {color:green}
```

8.2.11 Layout grafico

Utilizzando le proprietà `float` e `clear` è possibile organizzare i contenitori della pagina per creare un layout grafico. Innanzi tutto è necessario, nel file HTML, dividere la pagina in blocchi utilizzando i tag `<div>` e dando a ogni blocco un identificativo univoco (`id`) in modo che sia distinguibile dagli altri.:

```
<div id="header"> ... contenuto dell'intestazione ... </div>
<div id="navigation"> ... contenuto del menù ... </div>
<div id="content"> ... contenuto della pagina ... </div>
<div id="footer"> ... contenuto del footer ... </div>
```

In questo modo abbiamo creato quattro blocchi logici contenenti l'intestazione, il menù, il contenuto ed il footer di una pagina. Nel file CSS possiamo ora specificare il layout della pagina. Ad esempio, se si volesse realizzare un layout come quello illustrato in 8.3 si dovrà avere:

- il contenitore dell'header libero da entrambi i lati;
- il menù allineato e libero sul lato sinistro;
- il contenuto allineato sul lato sinistro;
- il footer libero su entrambi i lati senza alcun allineamento.

Il tutto si ottiene col seguente codice CSS:

```
#header {float:none; clear:both; .... }
#navigation {float:left; clear:left; ... }
#content { float:left; ... }
#menu { float:none; clear:both; ... }
```

Capitolo 9

Tecniche di buona progettazione di pagine web

Vista l'ampia gamma di stili possibili, qui di seguito verranno trattati alcuni dei temi fondamentali per la progettazione di una buona pagina web.

9.1 Scelta del font

Innanzitutto, indifferentemente dalla tipo di pagina che stiamo realizzando, in una singola pagina è buona norma non utilizzare più di tre o quattro font diversi. Per un motivo analogo è consigliato non cambiare carattere nel mezzo di una frase (a meno di avere una ragione valida), ma utilizzare piuttosto uno stile diverso (italico, grassetto, sottilineato...).

Per quanto riguarda la scelta della famiglia da utilizzare, dobbiamo tenere in considerazione che in generale i font privi di grazia (`sans-serif`) affaticano meno la vista (già molto sollecitata dall'uso intenso di schermi video) ed essendo privi di abbellimenti vengono resi molto bene anche su schermi video (che hanno tipicamente basse densità, circa 100 PPI).

I font con grazia (`serif`), invece, non vengono resi bene dagli schermi ma sono perfetti se la pagina deve essere stamapata: risultano più rilassanti per gli occhi e sono resi bene anche da stampanti di bassa categoria (che solitamente hanno almeno una densità di 300 PPI).

La famiglia `monospace` risulta molto utile per scrivere codice o imput da tastiera mentre la famiglia `fantasy` si presta bene per accentuare l'enfasi in un testo.

Infine, indifferentemente dalla scelta della famiglia, si sconsiglia di richiedere un font "raro" perché aumentiamo la possibilità che esso non sia dispondibile per l'UA che dovrà visualizzare la pagina.

9.2 Densità delle immagini

Vi sono due unità di misura per esprimere la densità, a seconda che si tratti di uno schermo o di una stampante:

- *DPI* (dot-per-inch) misura i punti per pollice, usato per indicare la qualità di una stampa o di una stampante;

- *PPI* (pixel-per-inch) misura i pixel per pollice, usato per le immagini e gli schermi video.

Esiste un semplice metodo per calcolare quanti PPI ha uno schermo. Innanzitutto si deve calcolare la diagonale dello schermo (in pixel) usando il teorema di Pitagora:

$$d_p = \sqrt{h_p^2 + w_p^2}$$

ove d_p, h_p e w_p sono rispettivamente la misura della diagonale, dell'altezza e della larghezza dello schermo espresse in pixel. Per ottenere la densità in PPI basterà ora dividere la diagonale miurata in pollici d_i (solitamente nota in uno schermo) per la diagonale in pixel calcolata:

$$D = \frac{d_p}{d_i}$$

Ad esempio un notebook $15.4''$ con risoluzione $1920 \times 1200\,\text{px}$ avrà una densità di soli $147\,\text{PPI}$, in base al seguente calcolo:

$$d_p = \sqrt{1920^2 + 1200^2} = \sqrt{5126400} = 2264\,\text{px}$$
$$D = \frac{2264\,\text{px}}{15.4\,\text{in}} = 147\,\text{PPI}$$

Si noti che una persona dotata di una vista normale può normalmente distinguere particolari sino a $300\,\text{DPI}$.

9.3 Leggibilità del testo

Per rendere il testo facilmente leggibile possiamo usare alcuni accorgimenti:

- *Contrasto:* un testo chiaro su uno sfondo scuro è più leggibile, ma è esteticamente più bello un testo scuro su uno sfondo chiaro (ad esempio nero su bianco).
- *Colori:* bisogna prestare attenzione al contrasto relativo, se mettiamo ad esempio su uno sfondo arancione un testo rosso o giallo, questo risulterà poco leggibile.
- *Dimensioni:* bisogna considerare che, a parità di dimensione, font diversi utilizzano spazi verticali diversi (ad esempio il Verdana occupa il 58% mentre il Times New Roman solo il 46%). In generale comunque si sconsiglia di scendere sotto i 10pt come grandezza, indifferentemente dal carattere. (11-12pt è un'ottima scelta).
- *Spaziatura tra righe dello stesso paragrafo:* andrebbe lasciato almeno un 25-30% per ottenere una migliore leggibilità.
- *Spaziatura tra le lettere:* una buona spaziatura (o chiaramente la famigilia monospace) aumentano la leggibilità del testo.
- *Font e stile:* è sconsigliato l'utilizzo di font condensed e si consiglia di limitare l'uso dell'italico a piccole parti in quanto poco leggibile.

Quando si stampa un documento conviene considerare i seguenti punti:

- evitare l'utilizzo di carta lucida in quanto il riflesso disturba la vista delle persone con la vista più debole;

- usare ampi margini di rilegatura in modo che si possa aprire le pagine in modo orizzontale completo;

- nel caso si abbiano più volumi di una stessa collana si consiglia di cambiare colore e stile della copertina, in modo da facilitarne la ricerca in una biblioteca.

Il web design è un'espressione utilizzata per indicare la struttura grafica di un sito web suggerendo delle tecniche di buona progettazione per ottenere delle pagine gradevoli e chiare. Quindi i fattori presi in esame sono le varie tipologie di font esistenti e i metodi per ottenere delle pagine web leggibili mediante un contrasto e una spaziatura adeguata con una giusta combinazione dei colori.

Altro aspetto di fondamentale importanza riguarda la stampa, ponendo l'attenzione sull'insieme di regole per creare le cosiddette pagine printer-friendly e sui vari modi di implementazione.

9.4 Caratteristiche del testo

Le principali regole prevedono di non usare più di tre o quattro font diversi per pagina, di non cambiare tipologia nel mezzo di una frase e non richiedere font rari poichè aumenta la possibilità che non siano disponibili o comunque diminuisce la portabilità del testo.

È possibile suddividere i tipi di carattere in due categorie principali: con o senza grazie (note anche con l'inglese serif); i caratteri graziati hanno delle particolari terminazioni alla fine dei tratti delle lettere.

9.4.1 Tipologie di font

I caratteri senza grazie sono diventati lo standard per il testo visualizzato sullo schermo poichè forniscono una resa pulita meglio leggibile rispetto ai caratteri con grazie e quindi affaticano meno la vista. Invece il testo stampato risulta più gradevole usando font con grazia resi bene dalla migliore qualità della stampa. Alcuni esempi di font serif sono Georgia, Times, Times New Roman mentre tra quelli sans-serif rientrano Arial, Verdana e Helvetica.

Inoltre esistono font minori tra cui quelli monospace per codice o input da tastiera (Courier, Monaco), fantasy (Desdemona, Impact) e cursive (Comic Sans MS); i font fantasy e cursive sono da usare con estrema parsimonia e mai per grandi porzioni di testo dato che risultano poco leggibili e spesso non includono caratteri accentati.

9.4.2 Leggibilità testo

Oltre al tipo di carattere utilizzato, altri fattori influiscono sulla buona leggibilità di una pagina web. Tra questi rivestono un ruolo importante:

- Contrasto: preferibile ed esteticamente più bello il testo scuro su sfondo chiaro ma anche accettato testo chiaro su sfondo scuro;

- Colori: giusta combinazione tra sfondo e testo (evitare ad esempio il rosso sull' arancione);

- Spaziatura: lasciare almeno 25-30 punti percentuali;
- Dimensione carattere: preferibilmente grande, dai 14 ai 18 punti, anche se ci possono essere variazioni a seconda del font scelto;
- Margini: applicarli nel caso di materiale soggetto a rilegatura.

9.5 Pagine web "printer-friendly"

Nel corso degli anni il web è diventato la principale fonte d'informazione e di pubblicità esistente. Questo ha aumentato la complessità delle pagine con immagini, annunci, link, simboli e logi utili in fase di navigazione ma che non ne agevolano la lettura a video; in alcuni casi risulta essere preferibile stampare la pagina per potervi prendere appunti o per semplici ragioni di comodità. Tuttavia la stampa diretta offre un pessimo risultato in termini di leggibilità e di spreco di risorse poichè, pubblicità o immagini non utili al fine desiderato, non vengono eliminate.

Per questo ragioni per la stampa si generano pagine chiamate "printer-friendly", termine che descrive una versione di una pagina web ottimizzata per la stampa seguendo un'approccio metodologico di buona progettazione, anche se in merito ci sono varie correnti di pensiero riguardo ciò che sia giusto mantenere e ciò che invece debba essere elimininato.

9.5.1 Regole da adottare

Innanzitutto, come nella realizzazione della pagina web a video, buona parte delle regole riguardano la formattazione del testo che coinvolge tipo di font, dimensione e colore; a questo segue la parte su quello che dev' essere eliminato e mantenuto. Alcuni sostengono che solo il contenuto dell'articolo e il titolo devono essere inclusi nella pagina. Altri sviluppatori affermano che è sufficiente rimuovere la parte laterale e in alto di navigazione o sostituirli con i collegamenti di testo in fondo all'articolo. Alcuni siti rimuovono la pubblicità mentre altri rimuovono solamente qualche annuncio o in alcuni casi nulla. Ecco alcuni suggerimenti:

- Cambiare i colori, testo nero su sfondo bianco.

  Se la pagina Web ha un colore di sfondo, oppure utilizza caratteri colorati, è necessario modificarla in fase di stampa. Preferito testo nero su sfondo bianco; questo perchè molti utenti utilizzano stampanti in bianco e nero.

- Modificare il font.

  La maggior parte delle pagine Web sono scritte con font sans-serif, perchè più leggibili a video. Tuttavia in fase di stampa conviene usare font serif dal momento in cui le stampanti hanno risoluzioni maggiori rispetto ai monitor. Come dimensione del testo consigliati 12 pt o superiore.

- Sottolineare i link.

  In questo modo si rende evidente che si tratta di un link; inoltre è possibile colorarli in blu per utenti che dispongono di stampanti a colori.

- Rimuovere le immagini non necessarie e le animazioni.

  Quali immagini siano essenziali dipende dallo sviluppatore e dal reparto marketing. In linea generale mantenere il logo dell'azienda e le immagini fondamentali per l'articolo.

- Rimuovere i menù / link per navigare le pagine.

  Utili in fase di navigazione ma nella stampa non servono, anzi ne ostacolano la comprensione.

- Eliminare tutta o in parte la pubblicità.

- Inserire la URL della pagina.

  Questo permette di mantenere un riferimento all'originale e di pubblicizzare il sito.

- Inserire una nota di copyright.

  Non impedisce la copiatura ma è un avviso legale.

9.5.2 Tipologie d'implementazione

Una volta fissate le regole generali per creare delle pagine web printer-friendly poniamo l'attenzione in merito alla loro implementazione.

Fondamentalmente esistono due possibili tecniche: la creazione di due pagine diverse, una per la visualizzazione a video e l'altra per la stampa, oppure utilizzare e definire appropriati CSS. Il primo metodo suggerito è attuabile solo per pagine poco complesse poichè necessita di un carico di lavoro elevato e per questo è poco utilizzato. Il secondo metodo è sicuramente quello più consigliato ed agevole, evitando la duplicazione della pagina. Per realizzare i relativi CSS esistono due possibili soluzioni:

- definire tre CSS diversi (base, monitor e stampa) e includerli in modo condizionale con l'attributo `media`

```
<link rel= "stylesheet" type= "text/css" href= "base.css">
<link rel= "stylesheet" type= "text/css" href= "screen.css"
   media="screen">
<link rel= "stylesheet" type= "text/css" href= "print.css"
   media="print">
```

- definire un unico CSS con sezioni condizionali tramite `@media`

```
/* parte comune a qualunque media */
h1, h2, h3 { font-style : italic }
/* solo per stampa */
@media print {
   body { font-size : 12pt; }
   . . .
}
/* solo per schermo */
@media screen {
   body { font-size : 100%; }
   . . .
}
```

In entrambi i casi è importante assegnare identificativi logici e significativi ai vari elementi del layout (es. non "`box1`" e "`box2`" ma "`advertising`" e "`navigation`" rispettivamente

```
body {
   color : black;
   background : white;
   font-family : "Times New Roman", Times, serif;
   font-size : 12pt;
}
a {
   text-decoration : underline;
   color : blue;
}
#navigation, #advertising, #other { display : none; }
```

Figura 9.1: Esempio di CSS per la stampa di una pagina HTML.

per l'elemento che contiene i banner pubblicitari e per il menù). Questi identificativi saranno poi usati per nascondere, tramite la direttiva "`display: none`" i box inutili per un certo media.

Per concludere questo argomento, la figura 9.1 contiene un esempio di CSS adatto per la stampa di una pagina HTML.

9.5.3 Panoramica sui colori

La scelta dei colori risulta essere di fondamentale importanza per gli sviluppatori di pagine web poichè influisce sulla leggibilità del testo e quindi sulla sua facilità di comprensione.

Per la percezione umana è utile considerare le coordinate *HSL* (Hue, Saturation, Lightness) che descrivono nel modo migliore la percezione dei colori per l'occhio umano variando luminosità e saturazione.

Il parametro Hue è quello che determina il colore base e in base ad esso è definito il cerchio dei colori: la figura 9.2 presenta un cerchio base ed uno più complesso in cui sono idnetificati i colori primari, secondari (combinazione di due primari) e terziari (combinazione di un primario ed un secondario).

Per ottenere un buon contrasto bisogna scegliere tonalità con Hue distante, meglio se diametralmente opposte sul cerchio dei colori (come ad esempio giallo e blu), evitare sfumature che non tutti gli schermi sono in grado di riprodurre ed usare, per quanto possibile, colori standard.

9.6 Riferimenti ed approfondimenti

Il testo presentato in questo capitolo è basato sulle seguenti fonti, utili anche per approfondire gli argomenti trattati:

- font
  http://webdesign.about.com/od/fonts/a/aa080204.htm
- pagine web printer-friendly
  http://webdesign.about.com/od/printerfriendly/

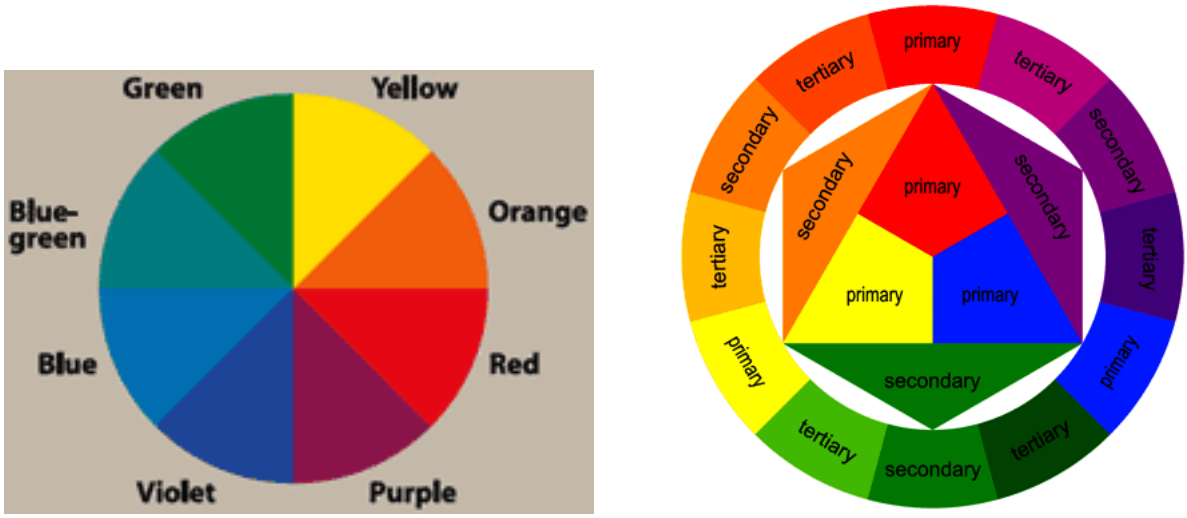

Figura 9.2: Il cerchio dei colori in base al parametro Hue.

- leggibilità del testo
  http://www.lighthouse.org/print_leg.htm
- teoria dei colori http://www.artyfactory.com/color_theory/color_theory.htm
- contrasto dei colori http://www.lighthouse.org/color_contrast.htm
- suggerimenti per la qualità dei siti web
  http://www.w3.org/QA/Tips/

Capitolo 10

Il protocollo HTTP/1.1

10.1 Perché una nuova versione?

Di solito è consigliabile non aggiornare la versione di un protocollo perché più la versione è stabile e meno problemi di compatibilità si creano. La versione 1.1 di HTTP è stata creata per risolvere una serie di problemi presenti nella versione 1.0.

HTTP/1.0 era stato progettato per oggetti statici e con dimensione nota, ovvero con l'assunzione che le risorse web corrispondessero ad un file memorizzato sul server e con dimensione fissa. Se la dimensione non è nota a priori, non esistendo nessun terminatore che segnala la fine della trasmissione, è possibile un troncamento dei dati nel caso che si verifichi L'errore dovuto a problemi con la rete infatti non è distinguibile dalla chiusura del canale da parte del server; questo vuol dire che in caso di errore nella trasmissione, dobbiamo ricominciare a scaricare il file dall'inizio mentre, per ottimizzare il traffico di rete, bisognerebbe poter riprendere il file dal punto in cui si è interrotto.

Il fatto che l'http 1.0 sia anche stateless comporta il bisogno di aprire una nuova connessione TCP per ogni oggetto che dobbiamo scaricare. L'apertura di una connessione TCP comporta un TCP setup (three-way-handshake) e una perdita di tempo dovuta all'algoritmo di slow start (partenza lenta quindi velocità limitata) mentre la chiusura del canale comporta la perdita di tempo dovuta al four-way-handshake e la perdita delle informazioni relative alla finestra TCP.

Lo gestione della cache era elementare (on off), potevo infatti solamente decidere se attivarla o no (pragma: no cache).

10.2 Migliorie dell'HTTP/1.1

La descrizione dell'http 1.1 è presente nell'RFC-2616. Nell'http 1.1 le connessioni non vengono chiuse ogni volta ma al contrario, può essere scelta una connessione persistente; in questo tipo di connessioni il canale rimane aperto fino a quando, o il client o il server, non decidono di chiuderlo. E' possibile anche decidere di sfruttare il pipelining che, insieme alla connessione persistente, serve a migliorare la velocità poiché nello stesso canale TCP possiamo effettuare più transazioni.

Altri miglioramenti sono relativi alla trasmissione del body; è possibile negoziare il tipo di dati che vengono trasmessi e la lingua in cui dev'essere trasmesso il contenuto del body. Nel caso di pagine dinamiche, siccome non è nota la dimensione della pagina, è possibile

spezzarla in frammenti di dimensione fissa. In questo modo siamo certi di trasmettere tutti i pezzi in cui è divisa la pagina indicando ogni volta la dimensione del pezzo che stiamo trasmettendo (chanched encoding). Grazie al chanched encoding il body può essere anche trasmesso solo in parti. Se mi si interrompe una trasmissione posso chiedere di trasmettere solo il pezzo che mi manca, cosa non possibile nell'http 1.0.

La gestione della cache è più raffinata, si possono scegliere varie modalità e sono previste gerarchie di proxy fra il client e l'origin server.

E' possibile avere server virtuali, cioè associare ad un unico indirizzo IP diversi server logici.

Sono stati aggiunti i metodi Put, delete, trace, options e connect ed è stato introdotto un nuovo metodo di autenticazione, basata su digest, direttamente a livello di trasporto dei dati.

10.3 Virtual Host

Con HTTP/1.0 occorreva un indirizzo IP per ogni server web ospitato su un nodo; per ospitare due server sulla stessa macchina bisognava avere due schede di rete o una stessa scheda di rete con due diversi indirizzi IP (multihomed). Con HTTP/1.1 non è più necessario perché questo protocollo prevede che allo stesso IP possono essere associati più server. Questo è dovuto al fatto che i server sono ottenibili tramite alias (Cname o nome canonico), ma soprattutto perché il client deve indicare il virtual host desiderato tramite il suo FQDN (Fully Qualified Domain Name).

Per indicare il FQDN è stato infatti aggiunto l'header Host e opzionalmente la porta su cui volete comunicare con quel server. Se non viene usato l'header host, dato che possono corrispondere più server allo stesso indirizzo, il protocollo non funziona.

Per comprendere meglio lo scopo ed il funzionamento dei virtual hosts consideriamo come esempio il caso di un ipotetico ISP con dominio `provider.it` che si è dotato di un server (denominato `host.provider.it` con indirizzo 10.1.1.1) per ospitare i server web dei suoi clienti.

DNS

A livello di DNS è stato creato un record (classe internet di tipo address) che associa al calcolatore host.provider.it l'indirizzo IP 10.1.1.1 Vengono acquisiti due clienti, www.musica.it e www.libri.it, e vengono aggiunti altri due record al dns (classe internet di tipo Cname) che segnalano che il nome canonico, cioè il vero nome, è host.provider.it e quindi che l'indirizzo IP associato ai due clienti sarà il 10.1.1.1.

Http

Per richiedere l'home page del sito www.musica.it ci si dovrà collegare all'IP 10.1.1.1, inviare una richiesta della pagina con metodo get, ad esempio get /index.html, utilizzare il protocollo http 1.1 e specificare host : www.musica.it;

Nel caso in cui volessimo l'home page del sito www.libri.it bisognerebbe inserire nel campo host : `www.libri.it` mentre il tipo di richiesta e l'IP a cui collegarsi resterebbero invariati.

Grazie all'header `Host` posso quindi distinguere quale virtual server contattare tra i tanti ospitati su uno stesso indirizzo IP.

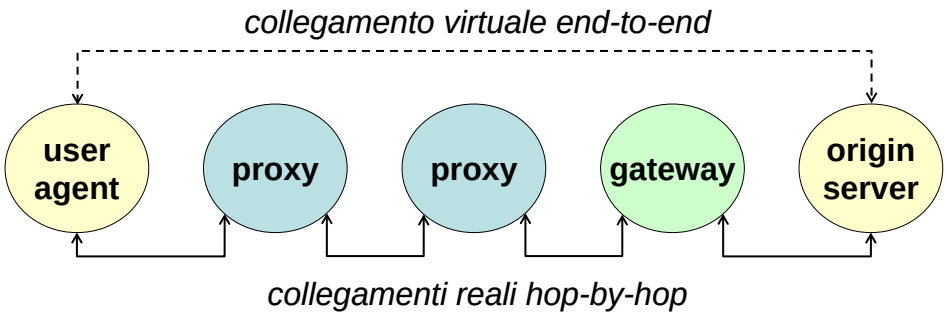

Figura 10.1: Collegamenti fra User Agent ed Origin Server.

10.4 Connessioni persistenti

Tramite le connessioni persistenti eliminiamo i problemi relativi al setup del canale TCP e dello slow start. In una connessione persistente le richieste e le risposte vengono inviate su un unico canale TCP; il canale verrà chiuso solamente quando o l'utente o il server inseriranno l'header `Connection: close`; che in pratica serve a tornare ad una connessione di tipo HTTP/1.0.

Con HTTP/1.1 non abbiamo più un metodo end-to-end ma hop-by-hop; l'header non interessa quindi solamente UserAgent-OriginServer ma le singole coppie, ed ognuno di questi collegamenti sarà indipendente dalle altre coppie. Nella maggior parte dei casi il nostro browser non è collegato direttamente al server ma ad un proxy, che a sua volta può essere collegato ad un proxy di livello superiore o al gateway, come nella figura 10.1 dove si evidenzia anche la differenza fra il collegamento end-to-end (unico) ed i collegamenti hop-by-hop (potenzialmente multipli). Possiamo quindi decidere di tenere chiuso il collegamento fra proxy e gateway ma tenere aperto quello fra gateway ed origin server. Ad esempio avere un canale persistente fra gateway e origin server (il canale più utilizzato) ed un canale chiuso fra gateway e proxy potrebbe servire a risparmiare tempo e nello stesso a non rischiare un sovraccarico del gateway in quanto i canali TCP non utilizzati verrebbero chiusi.

10.5 Esempio di connessioni persistenti

Un host vuole collegarsi all'home page del sito www.polito.it. Dopo aver aperto un canale TCP (TCP setup) l'host invia una richiesta con metodo get, specificando come tipo di connessione l'http 1.1 e come host www.polito.it. Il server risponde inviando la pagina richiesta ma, al contrario dell'http 1.0, non chiude il canale. A questo punto, dopo aver ricevuto la risposta, l'host può mandare un'altra richiesta o chiudere la connessione. Nel caso in cui il client voglia terminare la trasmissione, potrà aggiungere l'header `Connection: close`. In questo caso il server risponderà aggiungendo a sua volta l'header `Connection: close`. Il canale TCP verrà chiuso (TCP teardown).

L'header Connection close è relativo ad ogni singolo comando e se aggiunto sarà il vostro ultimo comando all'interno del canale.

Alcuni server possono avere un timeout, un canale che rimane inattivo può essere chiuso dal server anche senza la ricezione dell'header connection close. Il timeout è utilizzato per non tenere impegnata una risorsa di rete senza comunicazioni per troppo tempo.

10.6 Vantaggi e svantaggi delle connessioni persistenti

Alcuni dei vantaggi apportati dall'utilizzo di connessioni persistenti sono i seguenti:

Il numero di overhead in apertura e chiusura del canale è minore.

Il canale di chiusura (4-way-handshake) non ha solo un numero di pacchetti da inviare, ma anche un time out; rimangono infatti aperti dei canali per circa 2 minuti e mezzo. Grazie alle connessioni persistenti riusciamo a minimizzare questo problema.

Mantenendo la window TCP per tutta la durata del trasferimento riusciamo a migliorare la gestione della congestione della rete.

Il client può fare pipeline delle richieste, cioè inviare le richieste in modo sequenziale; il server deve fornire le risposte nello stesso ordine in cui il client le ha richieste.

Le migliorie introdotte dalle connessioni persistenti fanno si che tutti i nodi della catena che collegano l'user agent all'origin server lavorino di meno. Questo comporta un risparmio della CPU; positivo anche per il green computing (sistemi di calcolo ecologici).

Evoluzione dolce a nuove versioni di HTTP; se o il client o il server non supportano http 1.1 è possibile tornare all'http 1.0.

Le connessioni persistenti però portano anche alcuni svantaggi, alcuni di questi sono:

Maggior sovraccarico del server. Il principio di fondo delle connessioni persistenti è opposto a quello che era stato il punto di forza dell'http 1.0, dove il server chiudeva sempre il canale per permettere a tutti i client di contattarlo ed evitare che alcuni lo monopolizzassero.

Tenendo aperto un canale inutilizzato si rischia una possibile saturazione delle risorse. Le connessioni persistenti potrebbero essere utilizzate per quello che in sicurezza è chiamato denial-of-service[1] (negazione del servizio).

10.7 Come funziona il Pipeline

Il pipeline è la possibilità di inviare più richieste senza attendere le risposte, questo ottimizza la TCP ed è molto utile quando si richiedono in un colpo solo più elementi (di una stessa risorsa). Le richieste sono sequenziali e non parallele; le risposte saranno ricevute nello stesso ordine delle richieste. In seguito verrà presentato un esempio di un canale http 1.1 con pipeline. In caso di errore, però, potrebbe essere necessario ripetere tutto il comando perché il client può non essere in grado di capire a quale pacchetto di risposta fa riferimento l'errore.

10.8 Connessioni HTTP/1.0 e 1.1 a confronto

Ipotizziamo che un client desideri caricare una pagina contenente un testo HTML, un logo PNG, una libreria JavaScript, un footer. In totale si tratta di cinque oggeti, di cui gli ultimi quattro saranno noti solo dopo che è stato ricevuto il primo (la pagina HTML che contiene non solo il testo da visualizzare ma anche i riferimenti agli altri componenti della pagina). Studiamo ora le differenze nel caricamento della pagina usando il protocollo HTTP nelle versioni 1.0 e 1.1 con diversi accorgimenti

[1] E' un attacco che non mira a rubare dei dati ma a rendere il servizio inutilizzabile.

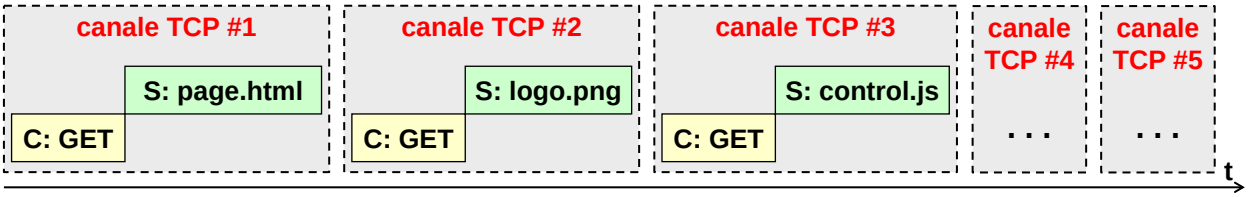

Figura 10.2: Diagramma temporale del trasferimento con HTTP/1.0.

La figura 10.2 illustra il caso è stato usato HTTP/1.0: il browser ha inviato una richiesta GET per la pagina e, dopo averla ricevuta, ha chiuso il canale. Stessa cosa ha fatto per la ricezione dei quattro ulteriori componenti della pagina. In totale avrà usato cinque diversi canali TCP[2]. Si notino le interruzioni tra un canale ed il successivo, nonché i tempi di set-up e tear-down di ciascun canale TCP indicati dalla durata del canale TCP (area grigia) maggiore della somma dei tempi di trasferimento del client (area gialla) e del server (area verde).

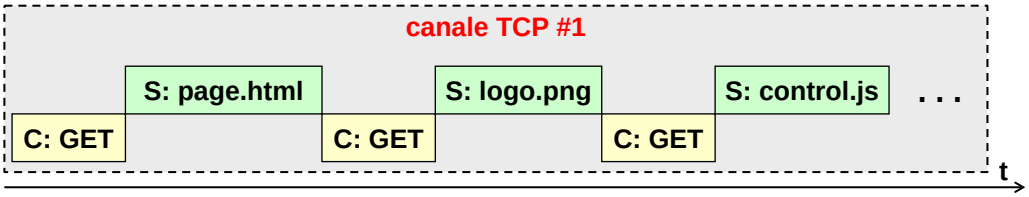

Figura 10.3: Diagramma temporale del trasferimento con HTTP/1.1.

Nella figura 10.3 è stato usato HTTP/1.1: il browser invierà la stessa sequenza di richieste ma con un unico canale. In questo modo non dovrà effettuare un solo setup e teardown TCP e lo slow start avrà luogo una sola volta.

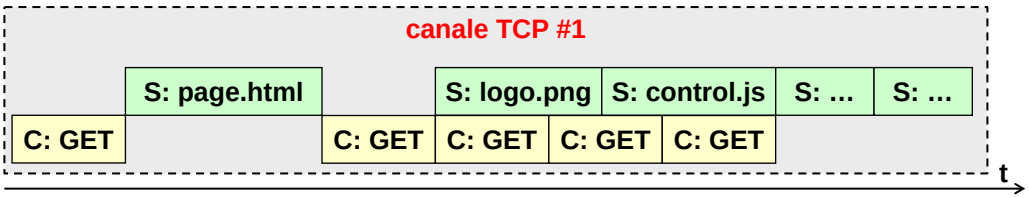

Figura 10.4: Diagramma temporale del trasferimento con HTTP/1.1 e pipeline.

Nella figura 10.4 è stato usato HTTP/1.1 con pipeline: dopo aver mandato il primo GET e ricevuto la pagina, il browser richiede gli oggetti in modo sequenziale, senza attendere la risposta ed usando lo stesso canale. Questo metodo è più veloce rispetto a quello senza pipeline perché occupa maggiormente i segmenti TCP (ossia i pacchetti di rete viaggiano più "pieni" invece che mezzi vuoti). In caso di errore nella risposta, dovendo chiedere nuovamente tutti gli oggetti, risulterebbe più lento.

Nella figura 10.5 è stato usato HTTP/1.0 ma con richieste in parallelo anziché in sequenza: il browser, dopo aver ricevuto il codice HTML della pagina ed aver chiuso il canale, aprirà un numero di canali TCP pari al numero degli oggetti da caricare. Questo metodo può risultare il più veloce, a scapito di un sovraccarico della rete per il numero di canali che sono aperti simultaneamente.

[2]Per esigenze di impaginazione nella figura sono mostrati solo i primi tre canali TCP.

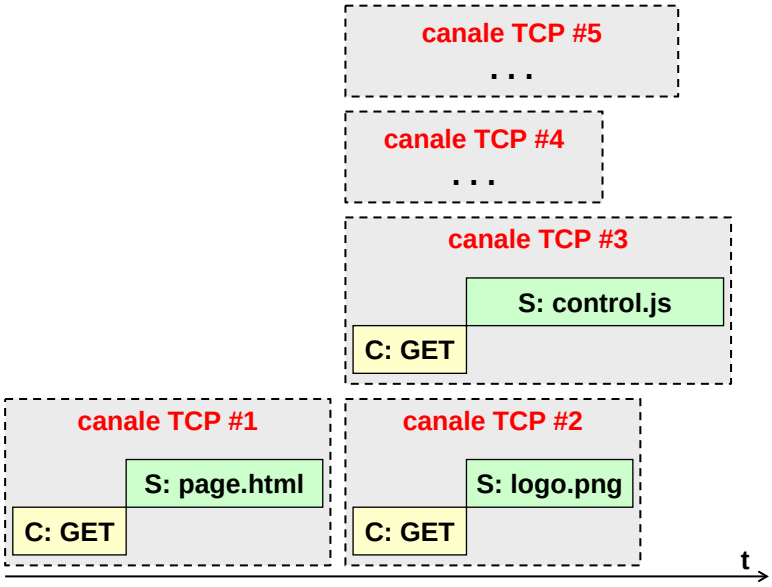

Figura 10.5: Diagramma temporale del trasferimento con HTTP/1.0 e canali paralleli.

Anche in HTTP/1.1 si posso usare canali paralleli ma così facendo si perderebbe il vantaggio di avere un solo canale TCP.

Ma quanti canali in parallelo conviene aprire? Aprire troppi canali non fornisce un vantaggio ma un sovraccarico. I browser infatti hanno tipicamente un numero predefinito di canali paralleli da aprire. E' chiaro che questo è un comportamento egoista perché non rispetta l'idea di un canale per un client; io voglio lavorare in fretta e quindi apro un numero di canali superiori, non è un metodo rispettoso della banda e degli altri utenti.

10.9 Compressione dei file

Un ulteriore vantaggio in termine di trasmissione sta nel fatto che nell'http 1.1 è possibile effettuare la compressione. Comprimere i dati vuol dire diminuire il numero di byte da inviare, che comporta un risparmio di tempo (come percepito dall'utente finale).

Per comprimere i dati il server sarà costretto ad utilizzare la CPU; per effettuare la compressione infatti vengono utilizzati degli algoritmi matematici. Anche il client sarà costretto ad utilizzare la CPU, in quanto dovrà decomprimere i dati prima di poterli visualizzare.

Nel caso in cui la pagina da inviare fosse statica, il server potrebbe effettuare la compressione dei dati soltanto la prima volta e quindi salvarla localmente, in modo da poterla inviare le volte successive senza effettuare nuovamente la compressione. Per le pagine dinamiche invece sarà costretto a comprimere i dati al momento della richiesta (compressione al volo) ed inviarli.

La risorsa scarsa, fino a poco tempo fa, era la banda: comprimere i pacchetti e inviare meno byte era quindi sicuramente una cosa positiva. Negli ultimi anni però, col crescente di dispositivi mobili lato client (es. tablet, smartphone), la soluzione non è più cosi immediata in quanto bisogna tener conto anche del consumo di energia, soprattutto quando questa è fornita da una batteria che è necessariamente limitata.

Utilizzando la CPU per decomprimere avrò un consumo energetico maggiore, però dati compressi significa impiegare meno tempo per la loro ricezione, ossia meno energia necessaria

per la parte di comunicazione wireless. Bisogna quindi chiedersi se un dispositivo mobile consuma di più usando la CPU o l'antenna. La risposta dipende dal tipo di mobile che il client utilizza. Questa potrebbe anche dipendere dal tipo di connessione che il mobile sta utilizzando, vi è infatti differenza fra un WiFi a banda larga o una connessione tramite UMTS. La risposta non è quindi cosi semplice.

Per quanto riguarda il consumo di batteria sui mobile, si è scoperto che la cosa che consuma di più è il GPS (80% circa di batteria). Stanno infatti sviluppando nuove versioni di software per i dispositivi mobili che spegne il GPS ogni volta che non serve.

Capitolo 11

La tecnologia ASP

11.1 IIS

IIS (Internet Information Service) è il server web di Microsoft

11.2 ISAPI

L'Internet Server Application Programming Interface (ISAPI) è un N-tier API di Internet Information Services (IIS). ISAPI è una interfaccia di programmazione (API) messa a disposizione da Microsoft per gli sviluppatori che intendono interagire ed ampliare le funzionalità del server web Internet Information Services (IIS). Ogni applicazione è una DLL caricata in memoria alla prima richiesta e mantenuta per soddisfare le successive, solo il sistemista può eliminarla. L'applicazione deve necessariamente rispecchiare il concetto di Thread Safety, in quanto è di fondamentale importanza che i vari threads possano avere accesso alle stesse informazioni condivise, ma che queste siano accessibili solo da un thread alla volta.

ISAPI è ideato come alternativa Microsoft a CGI, tecnologia invece orientata verso la creazione di un processo per ogni richiesta web quindi molto robusta in caso di crash, ma svantaggiosa in termini di utilizzo e consumo di risorse (CPU e RAM). In termini di prestazioni, ISAPI risulta migliore grazie all'implementazione tramite thread, ma per lo stesso motivo risulta meno robusto.

Le applicazioni ISAPI possono essere di due tipi: estensioni e filtri. In entrambi i casi si tratta di programmi sviluppati sotto forma di libreria dinamica (file `.dll`) e registrati con IIS per essere eseguiti sul server web sotto specifiche condizioni.

11.2.1 Filtri ISAPI

I filtri vengono utilizzati per modificare o migliorare la funzionalità fornita da IIS. Vengono programmati per esaminare e modificare se necessario, gli stream di dati sia in input sia in output, ovvero per effettuare pre/post processing della domanda e della risposta. Alcuni

slide numero 5 del gruppo di slide -ISAPI eASP-

Figura 11.1: Filtro ISAPI.

esempi di DLL possono essere `compfilt.dll` per la compressione HTTP o `md5filt.dll` per la digest authentication. I filtri posso ricoprire funzioni di load balancing, bilanciando il carico di domanda tra più server, o di sicurezza negli accessi e nei log on oppure possono riadattare la risposta alle capacità dei client.

11.2.2 Estensioni ISAPI

Le estensioni sono vere e proprie applicazioni eseguibili su IIS ed hanno accesso a tutte le funzionalità fornite da ISS. Le estensioni sono implementate come DLL caricate in un processo controllato da IIS. Il client può accedervi, come se stesse accedendo ad una pagina HTML statica.

L'estensione ISAPI è associata ad una pagina con una specifica estensione, la quale verrà elaborata restituendo in uscita il codice HTML risultante. Alcuni esempi sono `asp.dll` (per pagine ASP) e `ssinc.dll` (SSI).

Per ottimizzare le prestazioni del server è necessario regolare varie impostazioni associate alle estensioni ISAPI (DLL) in esecuzione sul server. La registrazione dei dati di utilizzo, quali il numero di richieste e l'ora in cui queste sono state inoltrate, può fornire indicazioni utili per effettuare tali regolazioni. IIS non risulta installato in windows e quando lo si installa per la prima volta, IIS è bloccato, ovvero è abilitata solo la gestione delle richieste di pagine Web statiche ed è installato solo il servizio Pubblicazione sul Web (servizio WWW). Attivazione IIS:

```
Pannello di controllo > Installazione Applicazioni >
Installazione componenti di Windows > Internet Information Services (IIS)
```

Nessuna delle funzionalità disponibili in IIS, ad esempio ASP, ASP.NET, esecuzione script CGI è abilitata. Se queste funzionalità non vengono abilitate, IIS restituisce l'errore 404. È possibile abilitarle utilizzando il nodo Abilitazione delle estensioni del servizio Web in Gestione IIS. Per eseguire le procedure di attivazione si usa MMC per gestire una virtual directory di IIS. Si possono associare estensioni, applicazioni, comandi HTTP accettati o addirittura pagine web specifiche per ovviare a vari errori applicativi attraverso il seguente percorso:

```
Properties > Home Directory > Application Settings > Configuration > Mappings
```

11.3 Introduzione ad ASP

ASP, acronimo di Active Server Pages, è un'estensione ISAPI e va sottolineato che esso non è un linguaggio di programmazione o di scripting, ma è una tecnologia di Microsoft che permette l'interpretazione degli script server-side. Le pagine ASP sono pagine web contenenti, oltre al puro codice HTML, degli script eseguibili dal server per generare runtime il codice HTML da inviare al browser del client. Così facendo si generano contenuti dinamici, il tutto senza dover inviare del codice eseguibile all'utente finale, al quale verrà inviato solo il risultato finale risparmiando in tempi e in banda.

I linguaggi utilizzabili sono VBScript e JavaScript per l'ambiente ASP. Grazie a questi linguaggi il sistema dinamico può comunicare lato server con tutti gli oggetti presenti sul sistema, rendendo così possibile lo sviluppo di siti dinamici basati sulle informazioni contenute nel database. ASP è indipendente dal linguaggio di scripting utilizzato.

slide numero 12 del gruppo di slide -ISAPI eASP-

Figura 11.2: Architettura ASP.

slide numero 13 del gruppo di slide -ISAPI eASP-

Figura 11.3: Accesso dati ASP.

Siccome esistono varie versioni del server IIS – e corrispondentemente della tecnologia ASP – si noti che in questo testo si fa riferimento alla versione 6.0 di IIS, le cui tecnologie dovrebbero essere supportata anche dalle versioni successive.

11.4 Architettura ASP

Come raffigurato in figura 11.2, il motore ASP è l'estensione ISAPI del server web (IIS) ASP.DLL, mentre il file con nome `x.asp` è un file di testo con estensione .asp costituito da HTML standard con linguaggio di script racchiuso tra i caratteri speciali `<%` e `%>`.

Sempre in figura 11.2 possiamo analizzare come avviene la richiesta e la ricezione di una pagina ASP. Il browser dopo aver attivato il canale HTTP (su trasporto TCP) con il server web, invia un messaggio HTTP di tipo GET della pagina `x.asp`, il server web ricevuta la richiesta legge da disco il file corrispondente e lo passa all'estensione ISAPI ASP.DLL che attraverso un interprete di script, esegue il codice accedendo se necessario a dati o oggetti presenti su disco o in database. Compilata la pagina di puro codice HTML, ASP.DLL la invia al browser del client che la tradurrà. Per specificare l'interprete da usare va inserito nel codice il comando `<%@ LANGUAGE="NomeInterprete" %>` IIS interpreta JScript/JavaScript o VBScript che è il linguaggio di default, inoltre è possibile specificarne anche altri come PearlScript o Python.

11.5 Accesso ai dati ASP

È possibile interfacciare le pagine ASP con qualsiasi tipo di database attraverso degli oggetti ADO (Activex Data Objects), come ad esempio Oracole, Sybase, MySQL, DB2 e tanti altri, o con dei server data o file system attraverso oggetti interni (built-in). Infine è possibile inoltre ricorrere ad oggetti proprietari attraverso delle applicazioni sviluppate con linguaggi di programmazione orientate agli oggetti come C++ o java.

11.6 Oggetto Enumerator

L'oggetto `Enumerator` è necessario se si vuole ciclare su una collection Microsoft, esiste solo su piattaforma MS Internet Explorer per client-side e IIS per server side.

slide numero 17 del gruppo di slide -ISAPI eASP-

Figura 11.4: Enumerator.

Come descritto in figura 11.4 per ciclare su una collection, prima viene creato un oggetto `Enumerator` che attraverso il metodo `moveFirst()` punta l'elemento corrente al primo elemento e poi viene creato un ciclo `while` (o `for`) fino al raggiungimento della condizione di uscita espressa mediante il metodo `atEnd()` che ritorna un valore booleano che indica se si è alla fine della collection. Il metodo `item()` ritorna l'elemento corrente, mentre la funzione `Response.write()` permette di scrivere il valore di una variabile, o di altri contenuti come del testo o del codice HTML (lato client `Document.write()`), per abbreviazione la funzione può essere sostituita dal simbolo `=` seguito da una stringa tra doppi apici o da una variabile. L'aggiornamento del ciclo viene implementato grazie al metodo `moveNext()` che sposta l'elemento corrente al successivo nella collection. Una possibile collection potrebbe essere `Request.ServerVariables`.

11.7 Oggetti interni ASP

Per facilitare la gestione dell'input e output il server ASP mette a disposizione diversi oggetti interni non istanziati:

- l'oggetto `Request` viene usato per recuperare informazioni passate dal client al server;
- l'oggetto `Response` viene usato per inviare dati dal server al client, sia nel body (tipicamente codice HTML o JS) sia nell'header HTTP (ad esempio valori dei cookie);
- l'oggetto `Application` viene usato per condividere informazioni fra diversi client che esplorano lo stesso gruppo di pagine (quelle che costituiscono una stessa applicazione ASP);
- l'oggetto `Session` permette di memorizzare temporaneamente informazioni relative ad un singolo client (ad esempio il carrello dei prodotti selezionati) ed è quindi utile per trasmettere informazioni utili a più pagine ASP;
- l'oggetto `Server` offre la possibilità di gestire alcune funzionalità del server ASP.

Si noti che tutti questo sono oggetti ASP, disponibili sia in JavaScript sia in VBScript, ma purtroppo la documentazione MS è quasi esclusivamente per VBScript.

11.7.1 L'oggetto Request

L'oggetto `Request` rappresenta la richiesta HTTP inviata dal client al server IIS; come tale, permette di conoscere il valore degli header HTTP e di estrarre il valore dei parametri di un form, siano essi nell'header o nel body.

Si presentano qui nel seguito le principali caratteristiche (proprietà, metodi e collezioni) dell'oggetto `Request`. Per un elenco completo consultare la relativa pagina su MSDN:

`http://msdn.microsoft.com/en-us/library/ms524948(v=vs.90).aspx`

Request collection

La collection `ClientCertificate` estrae i valori delle estensioni di un certificato digitale X.509 inviato dal client, utilizzata perché in genere il sistema username e password non risulta molto affidabile in termini di sicurezza.

Esempio trasmissione con GET	Esempio trasmissione con POST

lato client
 	
`<form method="GET" ...>`
`cognome?`
`<input type="text" name="cognome">`
 	
lato server
 	
`Egr. Ing.`
`<% =Request.QueryString("cognome") %>`
	lato client
	
	`<form method="POST" ...>`
	`cognome?`
	`<input type="text" name="cognome">`
	
	lato server
	
	`Egr. Ing.`
	`<% =Request.Form("cognome") %>`

Figura 11.5: Esempio di lettura parametri trasmessi con GET o POST.

Le collection `QueryString` e `Form` forniscono i valori dei parametri di un form inviati dal client rispettivamente mediante i metodi GET e POST:

```
<% val = Request.QueryString("nome o ID del controllo") %>
<% val = Request.Form("nome o ID del controllo") %>}
```

Da sottolineare la fondamentale corrispondenza che bisogna mantere tra il nome (o ID) del controllo nel form HTML e l'identificativo usato in ASP. La figura 11.5 presenta i due casi di trasmissione dei valori di un form tramite GET e POST e corrispondente lettura in ASP.

La collection `Cookies` estrae i valori dei cookie applicativi ed è usata per impostare o leggere i valori dei cookie. Se un cookie non esiste, il valore estratto sarà ???. Ad esempio, ipotizzando che in una precedente interazione il server abbia assegnato al client un cookie denominato `utentePolito`, il valore corrispondente a tale cookie per questo client si ottiene con:

```
<% user = Request.Cookies("utentePolito") %>
```

La collection `ServerVariables` permette di leggere i valori presenti nell'header HTTP della richiesta. Ad esempio:

```
<% bro = Request.ServerVariables("HTTP_USER_AGENT") %>     (il tipo di browser)
<% srv = Request.ServerVariables("HTTP_HOST") %>           (il nome DNS del server)
```

Va sottolineato che i campi dei form estratti server-side tramite queste collection non sono stringhe, come capita leggendoli client-side, ma sono oggetti ASP che non sempre vengono convertiti automaticamente nel tipo adatto all'operazione. Quindi conviene sempre convertire esplicitamente i valori al tipo di oggetto desiderato.

Proprietà dell'oggetto Request

`int TotalBytes`
E' una proprietà a sola lettura e fornisce il numero di byte inviati dal client nel body della

richiesta. Risulta utile quando il server deve leggere esplicitamente tutti i dati presenti nel body HTTP della richiesta, come nel caso di upload di un file.

```
<% bytecount = Request.TotalBytes %>
```

Metodi dell'oggetto Request

`BinaryRead (Int count)`
Questo metodo leggere dal body HTTP della richiesta il numero di byte specificato dal parametro `count` e restituisce un Array contenente tutti byte letti. Dopo l'esecuzione di questo metodo, il parametro `count` contiene il numero di byte che non si è riusciti a leggere (dovrebe sempre essere zero a meno di errori). Questo metodo è utile per leggere tutto il body HTTP di una richiesta, come è necessario nel caso di upload di un file.

11.7.2 Response

L'oggetto ASP `Response` è usato per manipolare la risposta HTTP inviata al client, inserendo dati nel body o modificando l'header. L'uso più frequente di questo metodo è per scrivere in modo dinamico parte del codice HTML inviato al client e per inviare dati tramite i cookie.

Si presentano qui nel seguito le principali caratteristiche (proprietà, metodi e collezioni) dell'oggetto `Response`. Per un elenco completo consultare la relativa pagina su MSDN:

http://msdn.microsoft.com/en-us/library/ms525405(v=vs.90).aspx

Proprietà dell'oggetto Response

`Boolean Buffer`
Specifica se la risposta HTTP deve essere inviata subito al client (`Response.Buffer=false`, è il default) o se deve essere accumulata in un buffer (`Response.Buffer=true`). Nel caso si scelga la seconda opzione, la risposta verrà inviata al client solo quando termina l'esecuzione dello script o quando il programmatore decide esplicitamente di trasmetterla (con `Response.Flush`). Ciò permette di ottimizzare la trasmissione di rete (segmenti più grossi) a discapito della rapidità (dati parziali inviati non appena generati), lasciando spazio anche ad eventuali correzioni (nella forma di cancellazione totale del contenuto del buffer e scrittura di una nuova risposta).

`String ContentType`
Imposta il content-type della risposta, configurando il corrispondente header HTTP. Se questa proprietà non viene impostata, il valore di default è `text/html`.

`String Charset`
Indica la codifica dei caratteri da usare nell'interpretare la parte di testo HTML della risposta. In pratica modifica l'header `content-type` della risposta HTTP. Non è necessario impostare questa proprietà se la pagina HTML generata contiene il corrispondente meta tag.

`Int Expires`
Configura il tempo (espresso in minuti) di validità della pagina nella cache del client prima che scada. Usando il valore -1 si ottiene scadenza immediata, ossia una pagina effimera.

`Data ExpiresAbsolute`
Imposta la data ed ora di scadenza della pagina generata. Se si imposta solo la data, la pagina scadrà alla mezzanotte della data indicata. Se questa proprietà non viene impostata, la pagina scade immediatamente dopo essere stata ricevuta dal client, ossia è una pagina effimera.

`String Status`
Specifica il valore del codice di stato HTTP da restituire al client. Il valore impostato deve comprendere sia il codice numerico sia il commento, ad esempio:

```
Response.Status = "401 Unauthorized"
```

Metodi dell'oggetto Response

`AddHeader (String headerName, String headerValue)`
Aggiunge alla risposta HTTP un nuovo header, con nome `headerName` e valore `headerValue`.

`AppendToLog (String logText)`
Aggiunge la stringa `logText` in coda al log del server IIS.

`BinaryWrite (Array data)`
Inserisce nel body della risposta HTTP la sequenza di byte `data` senza effettuare nessuna manipolazione (utile per trasmettere il contenuto di file binari, come un file JPEG o MS-Word).

`Clear`
Cancella dal buffer della risposta qualunque dato ivi presente e non ancora trasmesso al client.

`End`
Termina l'esecuzione dello script, inviando al client tutti i dati presenti nel buffer della risposta.

`Flush`
Invia al client tutti i dati presenti nel buffer della risposta.

`Redirect (String URI)`
Reindirizza il client alla URI indicata come parametro del metodo.

`Write (data)`
Scrive nella risposta HTTP i dati passati come parametro del metodo. Si noti che i dati non possono contenere la sequenza `%>` (perché simboleggia la fine della script ASP) che va sostituita con `%\>`. Ad esempio, non si può scrivere:

```
<% Response.Write("<hr width=80%>") %>
```

ma si deve scrivere

```
<% Response.Write("<hr width=80%\>") %>
```

Il server IIS provvederà ad eliminare il carattere supplementare inserendo nella risposta HTML solo il seguente testo:

```
<hr width=80%>
```