Progettazione di servizi web e reti di calcolatori

appunti trascritti dagli studenti (ed integrati dal docente)

Prof.Antonio Lioy

Politecnico di Torino
Dip. Automatica e Informatica

v 1.09 - 17 giugno 2015

Indice

Versioni

versione	*data*	*commento*
1.00	19/3/2013	versione iniziale
1.01	24/3/2013	aggiunto capitolo 2 (TCP e UDP), basato sul contributo di Valentina Panzica
1.02	2/4/2013	aggiunto capitolo 3 (DNS), basato sul contributo di Gabriele Masseroni
1.03	2/4/2013	aggiunta parte iniziale del capitolo 4 (e-mail), basato sul contributo di Blinio Marco
1.04	2/4/2013	completato il capitolo 4 (e-mail) col contributo di Punzi Francesca
1.05	3/4/2013	aggiornato il capitolo 5 (architetture distribuite) del precedente anno accademico con le modifiche proposte da Barroero Fulvio
1.06	4/4/2013	aggiunti i capitoli 6 (web) e 7 (HTML) del precedente anno accademico
1.07	5/4/2013	aggiunti i capitoli 8 (CSS) e 9 (web design) del precedente anno accademico
1.08	21/3/2015	aggiunte due parti del capitolo 14 (PHP) in base ai contributi di Marco Gaido e Fabio Ballati
1.09	17/6/2015	aggiunto il capitolo 16 (sicurezza) in base al contributo di Benedetta Cacioppo e modificato il capitolo 9 in base al contributo di Alessio Perri

Capitolo 1

Introduzione

1.1 Premessa

Questo testo raccoglie gli appunti del corso "Progettazione di servizi web e reti di calcolatori" tenuto nell'anno accademico 2012-2013 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 Ingegneneria Gestionale, classe L-9, ex CDL in Organizzazione dell'Impresa) i riferimenti tecnologici per la valutazione, la progettazione e la gestione di servizi di rete basati paradigma web.

A questo fine, dopo la spiegazione del livello di trasporto ed applicativo nell'architettura TCP/IP, viene fornita un'introduzione generale alle diverse architetture distribuite di sistemi informatici (con valutazione qualitativa e quantitativa delle caratteristiche e delle prestazioni), esaminando specificamente i vari tipi di architetture client-server a più livelli.

contributo	*studenti*
capitolo 2	Valentina Panzica
capitolo 3	Gabriele Masseroni
capitolo 4	Blinio Marco
capitolo 4	Punzi Francesca
capitolo 5 (aggiornamento)	Barroero Fulvio

Tabella 1.1: Studenti dell'AA 2012-13 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 PHP 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) sulla sicurezza dei sistemi web.

Capitolo 2

TCP e UDP: il livello trasporto in TCP/IP

2.1 Introduzione al livello trasporto

Rispetto al modello OSI, in cui abbiamo 7 livelli (teorici, perché nella pratica i livelli presentation e session sono raramente implementati), nel modello dell'architettura TCP/ IP si hanno solamente 3 livelli (Fig. 2.1). Sotto IP non sono specificati i due livelli più bassi (data link e physical) che servono tipicamente per le reti locali. Il livello 3 IP, che significa Internetwork Protocol, si occupa di costruire un pacchetto (un insieme di bit) in grado di essere trasportato da una parte all'altra di una rete costituita da tanti nodi grazie al routing.

Partendo dall'alto del modello TCP/IP, vi sono le applicazioni che richiedono dei servizi, come ad esempio trasferire dei dati. Una specifica applicazione si deve far carico di definire:

- il *formato dei dati*, per far sì che tutti i nodi possano interpretarli allo stesso modo, perché computer diversi hanno formati di dati diversi e i dati devono essere universalmente leggibili;
- la *logica di funzionamento*, ovvero le domande e le risposte che possono essere trasmesse;
- il *protocollo* di comunicazione, ossia l'ordine corretto di trasmissione delle domande, delle risposte e dei dati.

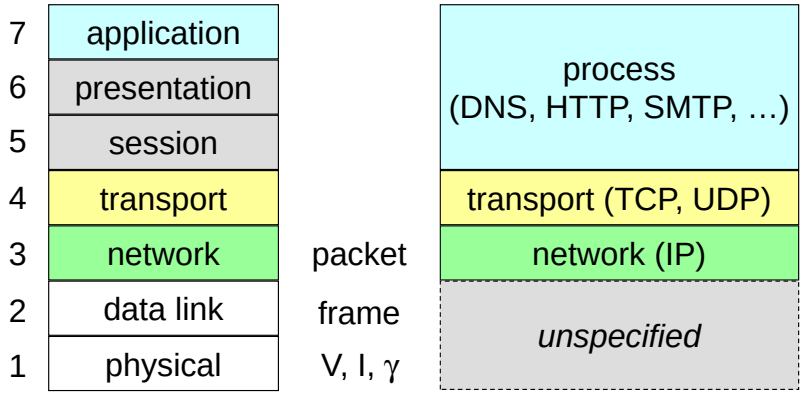

Figura 2.1: confronto tra stack OSI e TCP/IP.

Nei livelli più bassi sono definiti i protocolli, ossia come i bit vengono trasferiti da una parte all'altra della rete attraverso i pacchetti che vengono instradati sui router (protocolli di routing), allo stesso modo anche le applicazioni definiscono i loro protocolli, detti protocolli applicativi. I protocolli applicativi specificano, ad esempio, come un client deve fare una domanda ad un server e come il server invia la risposta (protocollo client-server). Semplificando un po' (ma neanche poi tanto) si può dire che i livelli 1 e 2 sono quelli di dominio delle telecomunicazioni, i livelli 3 e 4 sono pertinenza delle reti di calcolatori mentre i livelli 5, 6 e 7 sono tipicamente definiti dagli sviluppatori applicativi.

Dovendo trasmettere dei dati si potrebbe pensare di spezzarli in tanti pacchetti IP da dare alla rete che li porterebbe a destinazione, ma le reti IP hanno una serie di limitazioni. Un pacchetto IP, ad esempio, non può contenere un file da 1 GB, in quanto deve rispettare una dimensione massima indicata nel campo lunghezza (compresi gli header si arriva ad un massimo a 64 kB). Successivamente lo stesso pacchetto IP deve essere trasmesso all'interno di una trama di livello 2, che tipicamente, nel caso di Ethernet, ha una dimensione massima di 1500 B. Dovrebbe essere, dunque, il programmatore a suddividere i dati in tanti pacchettini IP.

Vi è, inoltre, anche il problema dell'affidabilità: la rete IP non è affidabile, perché non assicura che un pacchetto arrivi a destinazione. Quando un router è sovraccarico, può decidere di "buttar via" i pacchetti in più e l'applicazione mittente non sa se i dati arrivati sono completi o se mancano dei pacchetti.

Un'altra limitazione della rete IP è quella dei possibili pacchetti duplicati: se non c'è risposta da parte del destinatario, il mittente può decidere di rispedire il pacchetto, che può essere quindi ricevuto più volte.

Infine, la rete IP non rispetta l'ordinamento: se vengono inviati dei pacchetti in ordine, non è detto che verranno ricevuti nello stesso ordine, perché ogni pacchetto è instradato in modo differente da altri e a seconda del traffico che incontra, arriverà in tempi diversi in modo variabile. Non è presente un concetto di sequenza di pacchetto all'interno di IP.

Riassumendo i principali problemi della rete IP sono:

- non affidabilità
- pacchetti duplicati
- pacchetti fuori sequenza
- limite di dimensione dei pacchetti

Per far fronte alle limitazioni della rete IP, è stato creato a fattor comune il livello di trasporto (livello 4 della pila OSI), un livello logico end-to-end sopra la rete IP. Se due nodi vogliono parlare tra di loro nella rete, il livello 4 crea un canale di comunicazione.

Poiché applicazioni diverse hanno esigenze diverse, si sono creati due principali protocolli di trasporto: TCP e UDP. Il canale offerto dal protocollo TCP è un canale che privilegia l'affidabilità, la lentezza del canale è il prezzo da pagare se vogliamo che il file arrivi in modo completo, non spezzettato e in ordine. Il canale offerto dal protocollo UDP, al contrario, è un canale che privilegia la latenza a scapito dell'affidabilità. Ad esempio, giochi on-line o telefonate VoIP richiedono che l'informazione arrivi subito.

2.1.1 TCP (Transmission Control Protocol)

Il protocollo TCP offre un controllo della trasmissione e crea un canale end-to-end di tipo stream[1] bi-direzionale di byte tra due componenti distribuite (Fig. 2.2).

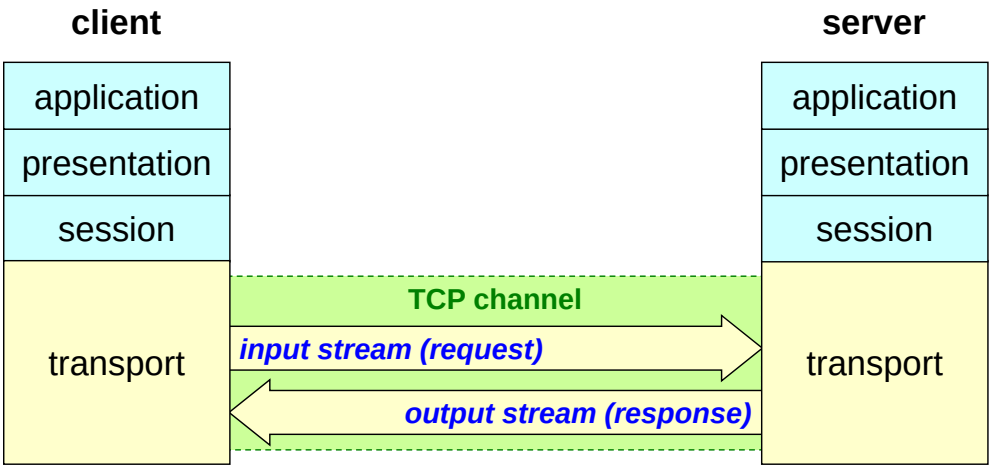

Figura 2.2: canale TCP.

Un singolo canale TCP tra due nodi A e B permette al nodo A di mandare dati al nodo B e viceversa, con trasmissioni simultanee, come una chiamata telefonica in cui si hanno due flussi vocali contemporanei. L'unità di trasmissione è pari a 1 byte (alla volta).

Il protocollo TCP offre anche *buffering* sia in trasmissione sia in ricezione, disaccoppiando così la velocità dei due nodi, altrimenti potrebbero comunicare solo nodi con la stessa velocità di trasmissione. Il buffer è una sorta di tampone o zona di "parcheggio" in cui i dati pronti per la trasmissione vengono accumulati sinche non è possibile trasmetterli in rete ed analogamente i dati ricevuti vengono accumulati sinché l'applicazione che deve usarli non è pronta a farlo.

Per comprendere meglio questo concetto immaginiamo che il server di YouTube trasmetta un filmato ad uno smartphone (Fig. 2.3): la potenza di calcolo e la velocità di trasmissione di un server non sono affatto uguali a quelle di uno smartphone e se il server di YouTube trasmettesse dati alla sua massima velocità lo smartphone probabilmente non riuscirebbe a gestire in tempo utile e molti verrebbero persi. Il buffer di trasmissione inizia a mandare dei bit al destinatario, il quale avrà un suo buffer di ricezione in cui riceve i dati, li elabora e fa vedere il risultato sullo smartphone.

TCP opera in questo modo per permettere che ogni nodo operi alla propria velocità, il trasmettitore non può trasmettere fino a quando il ricevitore non comunica di avere spazio libero per ricevere i dati, questo sicuramente rallenta il server che potrebbe trasmettere più velocemente, ma i dati in quel caso andrebbero persi.

Per tutte queste sue proprietà il protocollo TCP è il più usato dalle applicazioni Internet (es. web, posta elettronica, trasferimento file).

2.1.2 UDP (User Datagram Protocol)

Il protocollo UDP è un'alternativa a TCP perché offre come servizio l'invio di un *datagramma*, ossia permette alle componenti in comunicazione di scambiarsi messaggi contenenti una

[1] Un *data stream* è una sequenza ordinata di dati.

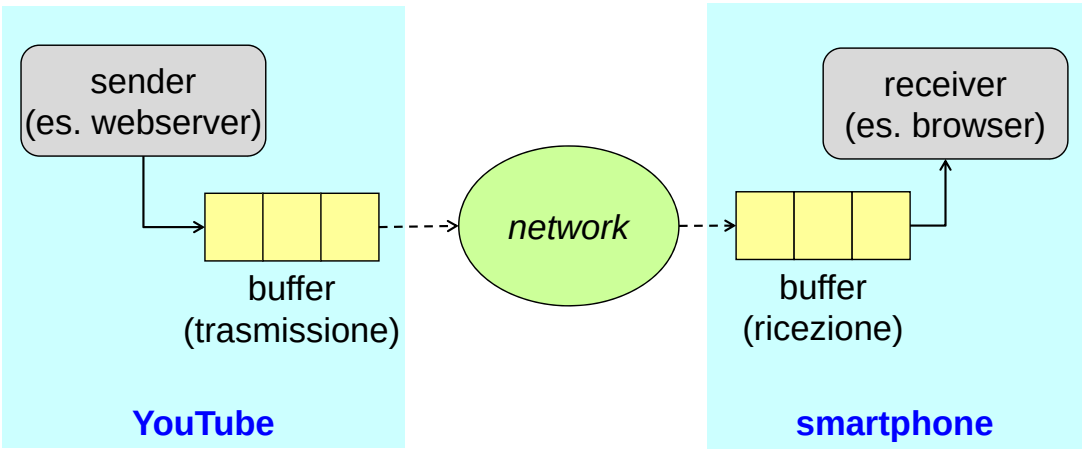

Figura 2.3: buffer TCP in trasmissione e ricezione.

sequenza di byte. UDP non ha il concetto di canale ma solo di singoli datagrammi ed i flussi di messaggi nei due sensi sono due comunicazioni indipendenti, che possono avvenire simultaneamente o in tempo diversi, in entrambe le direzioni o in una sola (Fig. 2.4).

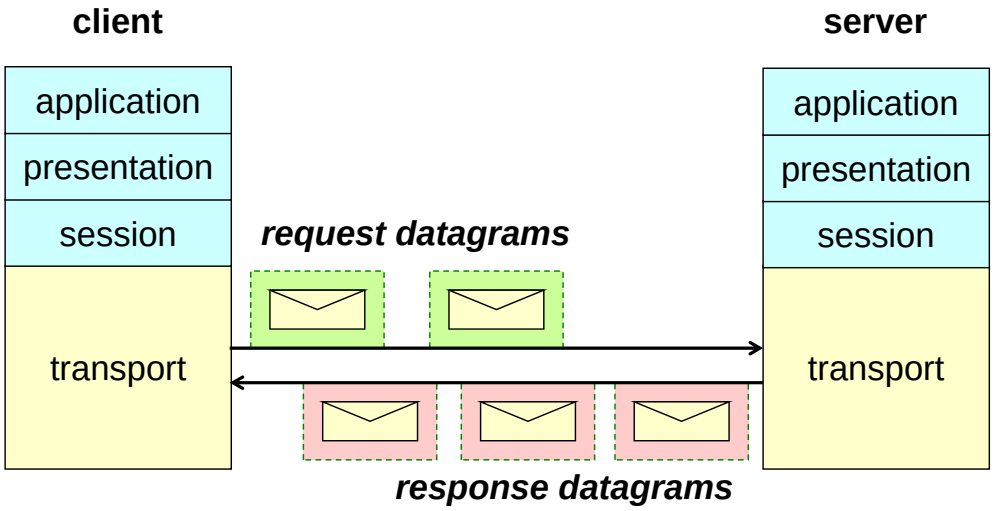

Figura 2.4: scambio di datagrammi UDP.

In TCP avevamo un canale in cui trasmettere a raffica le informazioni in bit, in UDP invece la trasmissione viene organizzata in singoli messaggi senza più controllo di trasmissione. UDP è un protocollo inaffidabile, in quanto non si preoccupa minimamente che i datagrammi arrivino a destinazione, ma compensa con il fatto che è estremamente veloce. La lunghezza del messaggio o datagramma è limitata a massimo 64 kB. L'accodamento in un buffer avviene *solo* nel nodo destinatario (se quest'ultimo è dotato di un buffer proprio), mentre al contrario in TCP vi è un doppio accodamento, buffer di trasmissione e di ricezione. Con UDP si corre il rischio di perdere dei pacchetti se il buffer di ricezione del destinatario è pieno. In generale UDP viene usato da applicazioni in cui la ritrasmissione o la perdita di un pacchetto non è un problema, ad esempio per applicazioni DNS, NTP o nel gaming on-line.

2.1.3 Le porte TCP e UDP

In generale TCP e UDP sono due protocolli di trasporto alternativi che realizzano funzionalità comuni a tutti gli applicativi. Poiché entrambi i protocolli sono usabili simultaneamente da applicativi diversi che risiedono sullo stesso nodo, deve esistere un modo per distinguere i dati generati da o destinati ad una specifica applicazione su un determinato nodo. A questo scopo si usa il concetto di *porta*, che dal punto di vista tecnico è associato al concetto di *multiplexing* e *demultiplexing* (fig. 2.5).

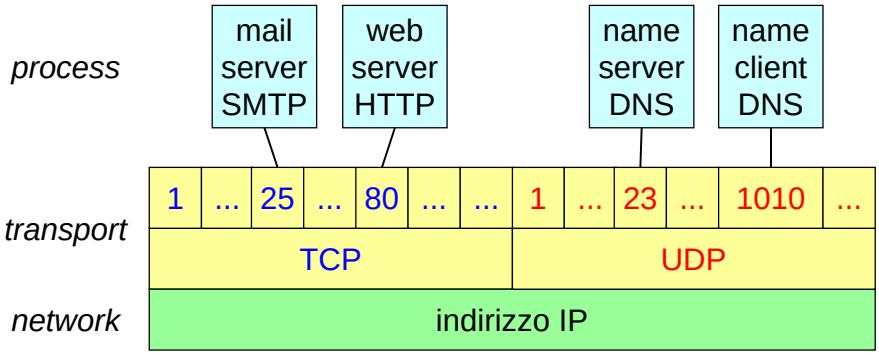

Figura 2.5: multiplexing di un stesso indirizzo IP tramite le porte TCP e UDP.

Ad esempio, quando un browser vuole connettersi ad un server web, deve indicare:

- l'indirizzo IP del server web (questo riguarda il flusso IP);
- il protocollo di trasporto (in questo caso TCP, perché sopra il flusso IP vogliamo il trasporto in un canale logico di tipo stream affidabile);
- il numero della porta associata al servizio richiesto (in questo caso servizio web, porta 80), perché un server offre tanti servizi, ognuno associato non solo all'indirizzo IP del server ma anche ad uno specifico numero di porta. (Analogia col mondo reale: le porte sono paragonabili ai diversi sportelli delle poste, specifici per un particolare servizio, ad esempio spedizioni e operazioni sui conti correnti postali).

Un indirizzo IP di un server pur essendo di per sé uno solo, associandogli un certo numero di porte è come se venissero moltiplicati gli indirizzi: questo è il concetto multiplexing.

Le porte TCP e UDP sono identificate da un numero intero su 16 bit, questo vuol dire che su un host ci possono essere fino ad un massimo di 65536 applicazioni (o meglio processi) diverse che in un certo istante stanno parlando in rete. Bisogna ricordare, però, che le porte in realtà non sono 65536 bensì 131072, perché ne abbiamo a disposizione 2^{16} per il protocollo TCP e altre 2^{16} per quello UDP. In contemporanea al massimo si potranno avere 65536 processi che usano UDP e altri 65536 che usano TCP.

Uno specifico processo che comunica in rete è identificato da tre dati: se l'indirizzo IP è fisso ci sono due protocolli tra cui poter scegliere e per ogni protocollo ci sono 2^{16} porte disponibili.

Le porte sono raggruppate in classi:

- le porte $0 \ldots 1023$ sono dette *porte privilegiate* e sono usabili solo da processi di sistema;
- le porte $1024 \ldots 65535$ sono dette *porte utente* esono usabili da qualunque processo.

Si definiscono inoltre *porte statiche* quelle dove un server è in ascolto, ossia in attesa di richieste. Al contrario, le *porte dinamiche* (anche dette *porte effimere*) sono quelle usate per completare una richiesta di connessione e svolgere un lavoro.

Quando due nodi stanno comunicando, indipendentemente dal fatto che usino TCP o UDP, hanno instaurato una comunicazione che è identificata sempre da una quintupla i cui elementi sono:

- il protocollo di trasporto scelto per la comunicazione (TCP o UDP);
- l'indirizzo IP (32 bit) e la porta (16 bit) del client;
- l'indirizzo IP (32 bit) e la porta (16 bit) del server.

Ad esempio, per un collegamento HTTP tra un browser attivo sul nodo con indirizzo 14.2.20.3 ed un server ospitato all'indirizzo 130.192.3.27 la quintupla potrebbe essere così identificata:

(TCP, 14.2.20.3, 1040, 130.192.3.27, 80)

Si noti che il browser usa una porta effimera (in questo caso la 1040) per stabilire la connessione col server. Alla prossima connessione (con lo stesso server o con uno diverso) il browser potrebbe usare una porta diversa, perché gli viene fornita automaticamente dal sistema operativo è non è importante che sia sempre la stessa, come invece deve essere per un server.

2.2 UDP (User Datagram Protocol)

Rispetto a TCP, UDP è un procollo più semplice, mentre rispetto al protocollo IP, UDP ci fornisce in più la possibilità di inviare messaggi di una dimensione maggiore di quella del semplice pacchetto IP.

UDP è un protocollo di trasporto orientato ai messaggi: non viene creato un canale, ma c'è un mittente che invia datagrammi, per tanto si dice che è un protocollo non connesso, non c'è un collegamento permanente o temporaneo tra i due nodi della rete che stanno comunicando. Questo significa che non sappiamo se il destinatario riceve o non riceve il messaggio, lo stato del destinatario è ignoto, perché non vi è nessun accordo preliminare per la trasmissione (in TCP c'è invece l'handshake). In questo senso UDP è un protocollo non affidabile perché i datagrammi possono essere persi, duplicati o anche fuori sequenza.

Il protocollo UDP aggiunge due funzionalità rispetto a quelle di IP: il multiplexing delle informazioni tra le varie applicazioni (tramite il concetto di porta) e la `checksum` (opzionale, perché già fatto ai livelli 2 e 3) per verificare l'integrità dei dati. L'uso di una checksum è utile contro errori di trasmissione, ma non contro attacchi alla sicurezza dei dati.

Il datagramma UDP si chiama anche PDU, ossia Protocol Data Unit, l'unità elementare trasmessa dal protocollo.

Ogni riga della PDU è composta da 32 bit (= 4 B). Nella figura è evidenziato in verde l'header, l'intestazione del pacchetto, costituito da 8 B. L'header è poi seguito da un numero variabile di dati trasportati (il payload UDP) inseriti dal livello superiore. I primi 4 byte della PDU rappresentano la porta mittente (source port, 16 bit) e la porta destinazione (destination port, 16 bit). Conosciamo, dunque, già tre dati della quintupla: due porte e il protocollo scelto per la comunicazione.

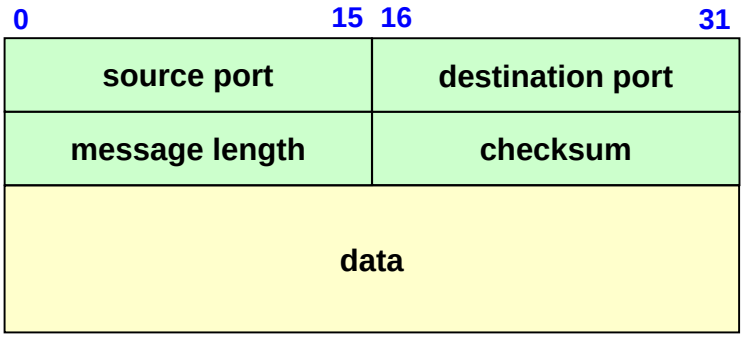

Figura 2.6: struttura di una PDU (datagram) UDP.

L'indirizzo IP del mittente e del destinatario non sono presenti nel datagramma UDP, poiché nel sistema OSI vi è il concetto di incapsulamento, il datagramma UDP (di livello 4) viene incapsulato come payload di un pacchetto IP di livello 3 per essere trasportato in rete. Questo pacchetto IP avrà un suo header, al cui interno ci saranno gli indirizzi IP del destinatario e del mittente. L'header di livello 3, infatti, serve per il routing.

Un altro campo dell'header UDP è il message length, anch'esso di 16 bit, che indica la lunghezza totale della PDU (ossia header più payload) e quindi anche la sua dimensione massima, pari a $2^{16} - 1$byte, ossia 65.535 byte.

Volendo calcolare quanto spazio è disponibile per i dati applicativi in un singolo datagramma UDP, bisogna sottrarre alla sua dimensione massima sia lo spazio occupato dall'header UDP (8 byte) sia quello richiesto dall'header IP (minimo 20 byte) perché un singolo datagramma UDP deve essere trasportato all'interno di un solo pacchetto IP, il quale ha anch'esso una dimensione massima complessiva di 65.535 byte. Risulta quindi che lo spazio massimo disponibile per il payload applicativo in UDP è pari a:

$$65.535\,\mathrm{B} - 8\,\mathrm{B} - 20\,\mathrm{B} = 65.507\,\mathrm{B}$$

Infine, l'ultimo campo dell'header della PDU è la checksum, anch'esso su 16 bit, i quali sono tutti a zero se non viene usata. La checksum contiene un codice di errore che protegge sia l'header sia il payload.

2.2.1 UDP: applicabilità

E' utile usare UDP in quattro casi concettuali:

1. Se si opera su rete affidabile, ad esempio LAN soprattutto quelle di tipo cablato oppure su una rete punto-punto.
2. Se una singola PDU può contenere tutti i dati applicativi che si vogliono mandare, ossia se due applicazioni si stanno mandando dei dati di lunghezza inferiore a 64 kB.
3. Se non importa che esattamente tutti i dati arrivino a destinazione.
4. Se è l'applicazione stessa che gestisce meccanismi di ritrasmissione

Tuttavia, il principale problema di UDP è il controllo di congestione: UDP è *connectionless* e quindi non sa che cosa sta accadendo all'altro capo della rete.

Immaginiamo di essere in un sistema in cui il destinatario si occupa di dire se i pacchetti sono stati ricevuti e supponiamo che ci sia troppa gente che sta usando la rete; se mandiamo, ad esempio un certo numero di pacchetti, ci si aspetta che il destinatario ci dia conferma di averli ricevuti. Tuttavia, ci può sembrare che il destinatario non risponda, perché magari si è persa la sua risposta. In questo caso allora si provvede a rimandare tutti i pacchetti causando un ulteriore intasamento della rete. Infatti, in UDP il mittente mantiene il proprio tasso di trasmissione (elevato) anche se la rete è intasata, contribuendo ad intasarla maggiormente.

Per risolvere il problema del controllo di congestione assente in UDP era stato proposto il protocollo DDCP (Datagram Congestion Control Protocol) che però nella pratica non è mai stato utilizzato.

2.2.2 UDP: applicazioni

Le principali applicazioni che usano UDP sono:

- **DNS (Domain Name System)** è il protocollo che si occupa di traduzioni nomi in indirizzi IP e viceversa.

  È l'applicazione principe per UDP, perché il DNS è logicamente un protocollo a messaggi, per il quale non serve un canale permanente. Dovendo fare una domanda a un server DNS per risolvere un indirizzo se non arriva la risposta non è poi così grave, perché si rifà la domanda finché non si ottiene una risposta.

- **NFS (Network File System)** protocollo per i dischi di rete (in ambiente Unix) Anche in questo caso si tratta di ripetere una domanda al server finché non si ottenga risposta.

- **SNMP (Simple Network Management Protocol)** si occupa della gestione apparecchiature di rete (router, switch, ...). Tramite questo protocollo le apparecchiature si scambiano tra loro informazioni statistiche sulla rete (ad esempio informazioni sul traffico) e non è importante la velocità dello scambio di informazioni ma è importante parlare in continuazione per essere costantemente aggiornati sulla situazione della rete. I pacchetti persi o duplicati non sono un problema perché si rimanda l'ordine, o la richiesta.

- Molte applicazioni di streaming audio e video, in cui è importante la bassa latenza (ci interessa che ad esempio le parole arrivino in fretta al destinatario) oppure in cui è accettabile la perdita di alcuni dati e non si perde tempo a ritrasmettere.

2.3 TCP: Transmission Control Protocol

TCP è un protocollo di trasporto che costruisce sopra IP un'astrazione logica del canale di connessione che permane tra due nodi della rete per un certo tempo, su cui si può inviare un flusso, o sequenza, di bit (in inglese *stream*).

Principali caratteristiche di TCP:

- byte-stream-oriented

- è un protocollo orientato alla connessione, o semplicemente *connesso*: i due nodi sanno l'un l'altro qual è il reciproco stato

- è affidabile, non ci sono problemi di pacchetti persi o duplicati.
- latenza alta, throughput minore

TCP è usato da applicativi che richiedono la trasmissione affidabile dell'informazione, come ad esempio:

telnet terminale virtuale

FTP (File Transfer Protocol) trasferimento file

SMTP (Simple Mail Transfer Protocol) trasmissione e-mail

HTTP (Hyper-Text Transfer Protocol) scambio dati tra browser e server web

TCP offre diverse funzionalità:

- *supporto della connessione tramite circuiti virtuali*: non vi è un vero circuito fisico ma è come se vi fosse un canale che collega i due nodi che parlano TCP (al di sotto, TCP usa i pacchetti IP quindi i pacchetti non seguono un cammino preciso, ma seguono un cammino che dipende dagli algoritmi di routing)
- *controllo di errore*
- *controllo di flusso*: viene controllato l'ordine dei pacchetti
- *multiplazione e de-multiplazione*: permette a più applicazioni di ricevere e di trasmettere contemporaneamente, ovviamente su canali TCP diversi
- *controllo di stato e di sincronizzazione*: se il destinatario non può ricevere, TCP ferma la trasmissione

Caratteristica più importante è che, in generale TCP, *garantisce la consegna del pacchetto*, al contrario di UDP. L'acknowledge (conferma di ricezione) è il meccanismo con cui TCP, implementa la garanzia di ricezione.

2.3.1 Campi dell'header TCP

Come per UDP, si ha la porta mittente (source port, 16 bit) e la porta del destinatario (destination port, 16 bit) all'inizio dell'header di un segmento TCP.

Poiché si necessita che i dati arrivino nello stesso ordine con cui son stati spediti, c'è bisogno del sequence number (32 bit), per numerare i segmenti in cui il file è stato spezzettato. Quando il flag del segmento corrente, corrispondente al flag SYN, è *vero* (cioè SYN=1) allora questo sequence number è il valore iniziale corrisponde al primo segmento di una trasmissione, tipicamente ha il valore 0. Quando SYN=0 (cioè *falso*) allora quel sequence number corrisponde alla posizione nello stream complessivo del primo data byte. I valori di questi flag sono presenti nel campo control.

Il campo acknowledgment number (32 bit) serve per dare conferma che certi dati sono stati ricevuti a seconda del valore del flag corrispondente (ACK). Se ACK=1, l'acknowledgment number indica la posizione nello stream del primo data byte da ricevere (tutti quelli precedenti sono stati ricevuti correttamente, dice sostanzialmente quale data byte si aspetta di ricevere). Se invece ACK=0, l'acknowledgment number non è significativo e viene ignorato.

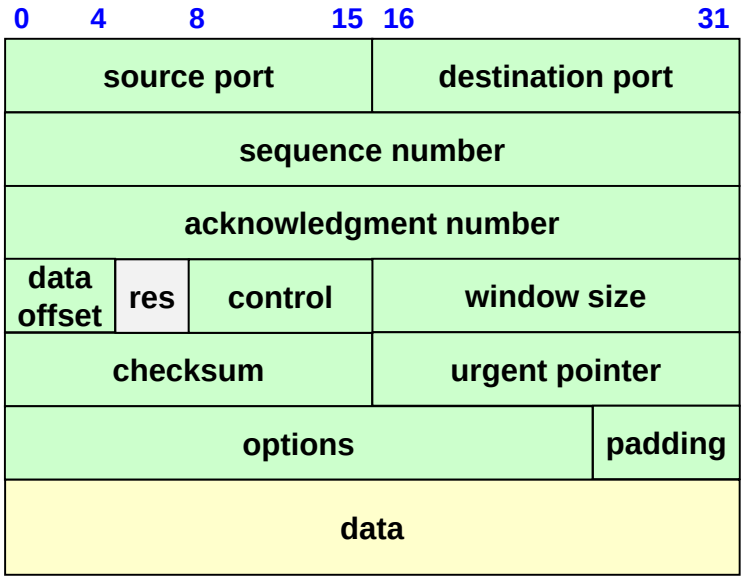

Figura 2.7: struttura di una PDU (segmento) TCP.

Il campo data offset (4 bit) indica la lunghezza dell'header TCP misurata in word da 32 bit e il suo valore varia da 5 a 15 (ossia 20 . . . 60 byte, con un massimo di 40 byte di opzioni).

Il campo control contiene un insieme di flag[2] ed è formato da 9 bit, guardati singolarmente:

- SYN serve per sincronizzare i sequence number ed è il flag usato all'inizio della trasmissione perché bisogna stabilire il punto si sta partendo. Ad esempio se SYN =0 e sequence number =57, vuol dire che questo primo byte di dato va messo in posizione 57 del flusso di dati complessivo (i successivi vanno messi dopo perché questa è una sequenza).
- ACK sta a significare che il campo Acknowledgment Number è valido.
- FIN indica la fine della trasmissione e non verranno trasmessi altri dati.
- RST è un flag che chiede il reset della connessione, è diverso dal FIN perché interrompe la trasmissione brutalmente per qualche motivo anche se non sono stati trasmessi tutti i dati (`abort`)
- PSH è una richiesta di inviare i dati presenti nel buffer di ricezione all'applicazione.
- URG indica se il campo Urgent Pointer valido o meno;
- NS, CWR, ECE servono per il controllo avanzato di congestione e non vengono considerati nella presente trattazione.

Il campo window size (16 bit) indica che chi ha trasmesso questo pacchetto ha ancora posto per un certo numero di byte, ossia segnala quanto spazio è ancora disponibile nella *receive window*.

Se si riceve un pacchetto con window size pari a zero vuol dire che non c'è più spazio e si deve interrompere la trasmissione. Il mittente può mandare al massimo il numero di byte indicati dal window size prima di attendere un ACK ed una nuova WIN: esso aspetta fino

[2]Si ricorda che un flag è un singolo bit usato per indicare vero o falso.

allo scadere del *timeout*, poi riprova a ritrasmettere oppure dopo una serie di ritrasmissioni non andate a buon fine si decreta che il collegamento è scaduto.

Il campo options ha dimensione variabile da 0 a 320 bit (in multipli di byte) e abilita varie opzioni, quali Timestamp, Selective Acknowledgment ed altri.

L'opzione Selective Acknowledgement (SACK) può essere utile nei casi in cui sia stato perso uno specifico segmento ma quelli successivi siano stati ricevuti correttamente (Fig. 2.8). In questa condizione, se si risponde con un acknowledgement si può al massimo dichiarare "*ho ricevuto tutti i segmenti fino al 50*", anche se in realtà si sono già ricevuto i segmenti da 100 a 200 ma non se ne può dare conferma perché il campo acknowledgement dice qual è *l'ultimo byte ricevuto correttamente per poter chiedere di ricevere tutti i successivi*, compresi quelli che avremmo già ricevuto. Se indicassimo come ultimo byte ricevuto correttamente il 200, non ci verranno mai re-inviati i precedenti byte mancanti. L'acknowledgement funziona bene, ma rischia di far ritrasmettere dei dati già ricevuti quando ci sono dei buchi di byte mancanti nel mezzo della trasmissione. Il selective acknowledgement si occupa di risolvere proprio questo problema, permettendo di mandare solo i byte che mancano (raffinamento di TCP).

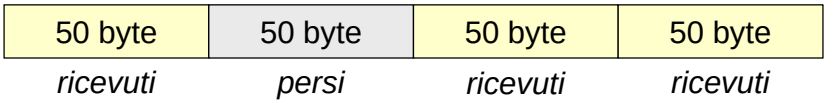

Figura 2.8: ACK normale e SACK in caso di perdita di un segmento.

Nel campo padding si mettono byte a zero per rendere l'header un multiplo di 32 bit come nel caso in cui le opzioni siano un multiplo intero di byte, ma non di 4 byte.

Come in UDP, il campo checksum (16 bit, tutti a zero se non usata) contiene un codice per rivelare errori di trasmissione e protegge sia l'header sia il payload.

2.3.2 Urgent Pointer TCP

Nel caso in cui il campo URG=1, indica che nel segmento ci sono uno o più byte urgenti, che tipicamente sono associati ad eventi improvvisi asincroni (ad esempio, `interrupt`). Questi byte possono essere byte da trattare prima di quelli già ricevuti ma ancora nel buffer.

L'uso dell'Urgent Pointer è un meccanismo per "saltare la coda" sul ricevente, ma ciò va bene quando questo segmento è già arrivato al destinatario. Tuttavia se il percorso di rete è lento non ha alcun effetto sulle code dei vari router in rete sui grossi trasferimenti di dati, non velocizza il trasferimento perché il segmento segue il normale indirizzamento. Soltanto nel momento in cui il segmento arriverà al destinatario, sarà un segnale di attenzione per il sistema operativo di destinazione che lo avvisa di avere un pacchetto urgente tra quelli nel buffer di ricezione. Quindi come tale molto spesso è praticamente inutile, è utile sui programmi interattivi in cui non c'è una grossa coda.

2.3.3 Apertura di un canbale TCP

Per creare un canale TCP c'è un meccanismo ben preciso che si chiama *TCP three-way handshake*. Durante questa procedura si parla di *active open* e *passive open*, ossia uno dei due nodi prende l'iniziativa di richiedere la connessione, tipicamente sempre il client.

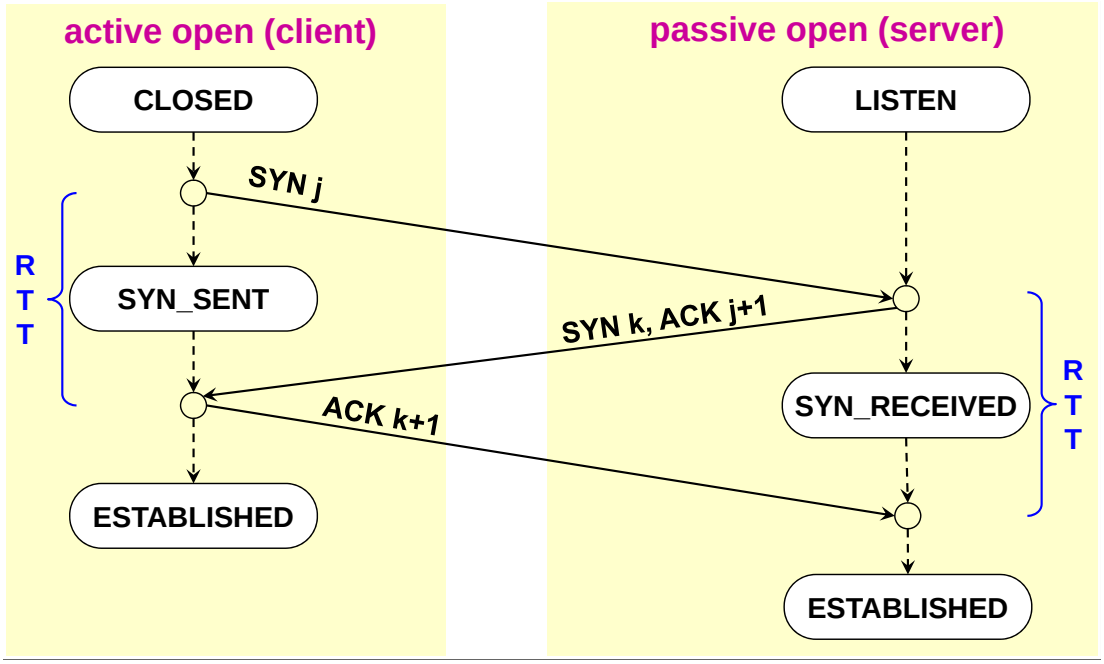

Figura 2.9: TCP three-way handshake.

È importante ricordare che TCP è un protocollo a stati che passa da uno stato ad un altro a seconda dei messaggi che riceve.

Sul lato client lo stato iniziale è `CLOSED`, sul server invece è `LISTEN`, esso è, infatti, in ascolto dei client che vogliono contattarlo. Dallo stato `CLOSED` il client manda un primo pacchetto TCP che contiene solo due informazioni importanti: il flag SYN=1 e il numero di sequenza `j` (`j` = *valore iniziale del numero di sequenza*), questa è una richiesta da parte del client al server di sincronizzare i loro numeri di sequenza.

Quando il SYN viene ricevuto dal server, lo sblocca dallo stato `LISTEN` e fa sì che risponda al client inviandogli il suo numero di sequenza `k` (diverso da quello del client), insieme con il campo `acknowledge number=j+1`, per indicare che il numero di sequenza j è stato ricevuto correttamente e il prossimo che si aspetta è `j+1`, cioè il successivo. A questo punto il server è andato allo stato SYN_RECEIVED, viceversa il client era già passato allo stato SYN_SENT una volta inviato il primo SYN.

È importante notare che il payload di questi segmenti TCP è vuoto, non si stanno ancora trasportando dati. Questo è un esempio di sovraccarico di TCP, perché stiamo usando pacchetti di rete ma senza trasportare dati informativi.

Quando il client riceve il pacchetto con SYN=k e ACK=j+1, risponde con un pacchetto che contiene solo ACK=k+1 (= "*ho ricevuto il tuo pacchetto etichettato a k, quindi il primo posto libero sarà k+1* "), una volta ricevuto l'ACK dal server, quest'ultimo si sblocca dallo stato SYN_RECEIVED ed entrambi i nodi vanno nello stato ESTABLISHED, *collegamento stabilito*.

Il *RTT (Round Trip Time* è il tempo che serve perché un pacchetto faccia andata e ritorno dai due nodi. Come minimo al pacchetto occorre un RTT e come massimo 2 RTT (i pacchetti possono essere persi per via della rete guasta).

Inoltre associato ad ogni stato c'è un *timeout* (tipicamente 200 ms per TCP): se il client è, ad esempio, nello stato SYN_SENT e dopo 200 ms non è stato ancora ricevuto il SYN-ACK, esso ritrasmetterà il SYN.

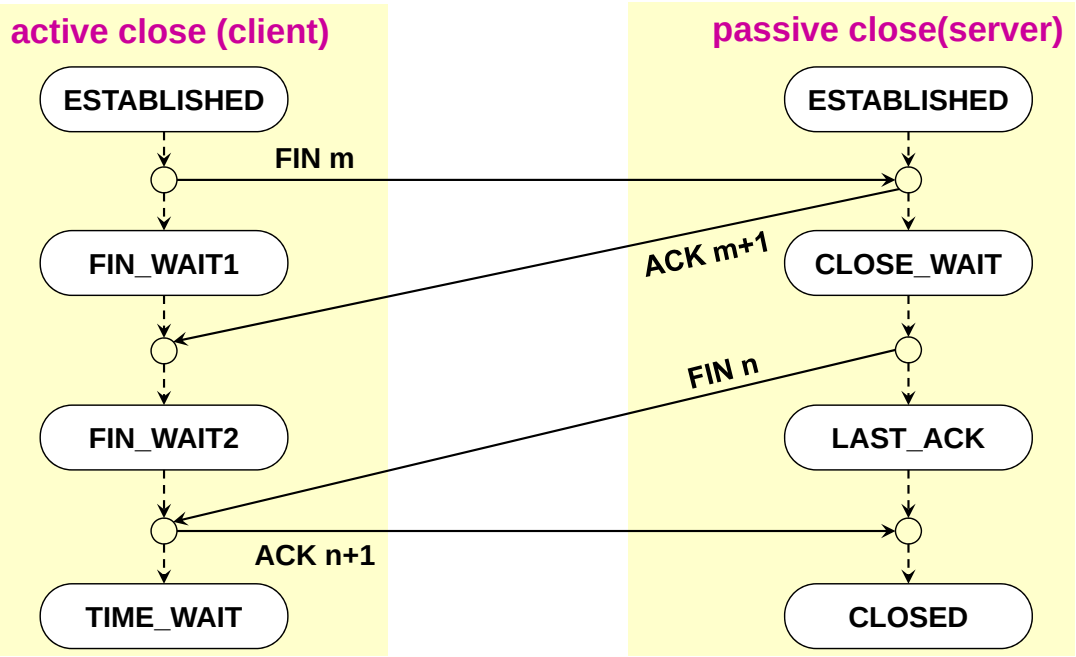

Figura 2.10: TCP four-way teardown.

2.3.4 Chiusura di un canale TCP

Duale dell'apertura del canale si ha quando si deve chiudere il canale: anche in questo caso uno dei due nodi prende l'iniziativa di chiudere la connessione (tipicamente il client applicativo, ma non è sempre così) e si parla di *active close (client)* e *passive close (server)*. La procedura prende il nome di *TCP four-way teardown*.

All'inizio entrambi i nodi si trovano nello stato ESTABLISHED.

Quando il client vuole chiudere la comunicazione, manda un segmento vuoto che contiene solo il flag di FIN, con il numero m, ossia il numero finale di sequenza: dopo questo numero di sequenza non arriverà più nulla da parte del client. Una volta mandato il FIN m il client passa allo stato FIN_WAIT1.

Non appena il server riceve il FIN m, risponde con un ACK m+1, che sta a significare "*ok ho ricevuto il tuo messaggio, mi aspetterei un m+1 ma non arriverà mai perché hai detto di aver finito*" e passa allo stato CLOSE_WAIT.

A questo punto il client, ricevuto l'ACK m+1, cambia il suo stato a FIN_WAIT2 e non fa niente fino a che non riceve il FIN n mandato successivamente dal server: quando ciò accade, il client sblocca lo stato FIN_WAIT2, manda un ACK che conferma di essere d'accordo sulla chiusura del canale e cambia il suo stato in TIME_WAIT.

Una volta mandato il FIN n, il server passa allo stato LAST_ACK e ricevuto l'ultimo ACK n+1 dal client chiude la connessione e va nello stato CLOSED.

Come si può notare, la chiusura del canale è più lenta dell'apertura: sono, infatti, necessari 4 segmenti anziché 3. Il client, inoltre, non ha chiuso subito il canale, ma rimane nello stato TIME_WAIT da cui esce solo per timeout. Il timeout ha una durata pari a 2 MSL, ove il parametro *MSL (Max Segment Lifetime)* corrisponde al massimo tempo che un segmento può rimanere in giro per la rete prima di arrivare a destinazione (perché non tutti i pacchetti seguono la stessa strada). Quindi il timeout va da un minimo di 1 ad un massimo di 4 minuti, a seconda del sistema operativo.

Lo TIME_WAIT esiste per risolvere due problemi:

1. implementare la chiusura TCP full-duplex

   L'ultimo ACK potrebbe venir perso ed il client ricevere un nuovo FIN. Infatti, se il segmento ACK n+1 mandato dal client viene perso, il server che era nello stato LAST_ACK in attesa proprio di quel segmento, non ricevendolo, dopo un certo tempo ritrasmette un nuovo FIN ed in questo caso il client deve essere pronto a mandare di nuovo un ACK di risposta al server per permettergli di chiudere la connessione.

   Nello stato TIME_WAIT, dunque, il client non fa nulla ma se riceve un FIN manda nuovamente un ACK.

2. permettere a pacchetti duplicati di "spirare"

   Se il collegamento venisse chiuso subito e si ricevesse un altro pacchetto dal server, esso potrebbe essere interpretato come il segmento di un nuovo collegamento, mentre in realtà appartiene ad una comunicazione vecchia.

   Segmenti di questo tipo potrebbero essere interpretati come parte di una nuova incarnazione della connessione sulla stessa porta: supponiamo che il pacchetto fosse destinato alla porta 54, se un'altra applicazione sta usando quella stessa porta dopo che è stato chiuso il canale, allora quel pacchetto duplicato verrebbe dato ad un'altra applicazione che non se ne fa nulla.

2.3.5 TCP Maximum Segment Size (MSS)

Come fatto per UDP, calcoliamo la massima dimensione di un segmento TCP:

$$576\,\mathrm{B} - 20\,\mathrm{B} - 20\,\mathrm{B} = 536\,\mathrm{B}$$

`TCP Maximum Segment Size (MSS)` massima quantità di dati (payload) presenti in un segmento TCP

Al massimo numero di byte per pacchetto IP, trattabile da tutti viene sottratto il minimo numero di byte per l'header IP e il minimo numero di byte per l'header TCP.

2.3.6 TCP sliding window

Il protocollo TCP utilizza sliding window per comunicare quanto spazio c'è ancora a disposizione. In generale il problema si pone sul ricevente se questo ha un buffer limitato.

Lo spazio disponibile varia in base al numero di ACK inviati e ai dati ricevuti e prelevati dal buffer da parte dell'applicazione. Per evitare troppe trasmissioni di pacchetti, il ricevente non manda ACK per singoli byte, ma cerca di raggruppare più byte ricevuti, dando poi la conferma complessiva di ricezione con successo.

Inoltre, si tende a non mandare un segmento solo per un singolo ACK, ma si usa fare "*piggyback*" su dati da inviare. Piggyback significa letteralmente "*farsi trasportare sulle spalle di un altro*", ad esempio si fa piggyback quando si manda un `SYN-ACK`: viene mandato il sequence number e contemporaneamente una conferma dei dati ricevuti precedentemente.

La dimensione della sliding window è quindi sempre pari alla dimensione dei dati ricevibili.

TCP riceve un byte-stream dal livello superiore e deve decidere quando inviare un segmento e con quanti byte, per fare ciò vi è un meccanismo di accumulo in entrambi i due casi seguenti:

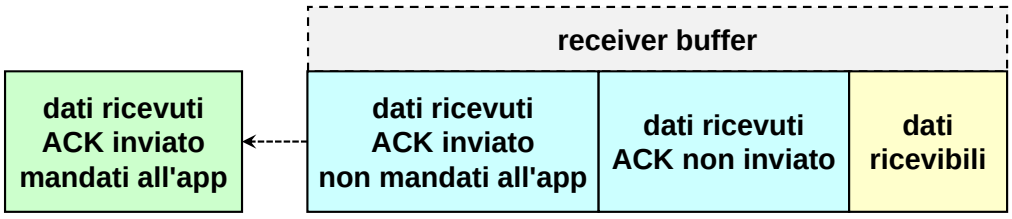

Figura 2.11: TCP: sliding window

- invia quando è pieno un segmento massimo (MSS) cioè se l'applicazione manda tanti byte tali da riempire un segmento (almeno 536 B = MSS)
- invia alla scadenza di un timeout (200 ms)

Altrimenti:

- invia quando si riceve un comando esplicito (`flush`) dal livello superiore (ad esempio una domanda, che necessita di una risposta affinché si possa andare avanti con l'elaborazione, deve essere mandata subito).
- quando, invece, non vogliamo che sia l'applicazione a dare il comando MANDA, si invia non appena ci sono dati disponibili, marcando la connessione come TCP_NODELAY: in questo modo TCP non applica l'algoritmo di accumulo per minimizzare il numero di segmenti da inviare. Con TCP_NODELAY abbiamo bassa latenza ma throughput inferiore, perché per mandare i dati è necessario mettere comunque un intero header (per inviare, ad esempio, un solo byte se ne devono necessariamente usare 21, col risultato che il traffico utile è pari a 1/21). In generale con TCP_NODELAY si privilegia la latenza a scapito delle prestazioni perché il canale apparentemente va più lento e si spreca banda.

I segmenti di grossa dimensione hanno maggiore latenza, migliore throughput (viceversa per segmenti piccoli).

Esempi: Per il download non interessa la latenza, si vogliono segmenti grossi Al contrario per i giochi on-line si vogliono risposte immediate.

2.4 TCP: slow start

Le prime versioni di TCP quando andavano in timeout ritrasmettevano l'intera window, questo poteva causare gravi congestioni della rete. Ad esempio, nell'ottobre 1986, Arpanet fu bloccata da una congestione: normalmente funzionava a 32 kbit/s e scese a 40 bit/s facendo diventare la rete 1000 volte più lenta.

Per evitare le congestioni, venne introdotto l'algoritmo *slow-start* il quale prevede che, quando si verifica un timeout, non si ritrasmette un segmento con la stessa dimensione, ma si riparte dal primo più piccolo possibile e la window viene reinizializzata al valore minimo e fatta crescere lentamente, per evitare nuove congestioni.

Questo è diventato un problema rilevante per le connessioni wireless, rispetto a quelle cablate, perché le prime sono più soggette a interferenze che possono far perdere i segmenti con le risposte TCP, cosicché quando si arriva al timeout TCP pensa che ci sia una congestione e riparte da zero. Con questo algoritmo, si ha un possibile collegamento a singhiozzo per le reti wireless e si perde moltissimo in prestazioni.

Capitolo 3

Il DNS (Domain Name System)

3.1 Funzione DNS

Il *DNS* (Domain Name System) è il sistema adottato da Internet per mantenere la corrispondenza tra *nomi logici (DNS)* (es. www.polito.it) e indirizzi IP (es. 130.192.182.33). Per svolgere la sua funzione, mantiene i dati in due "domini" separati:

- *Dominio Diretto*: fornito un nome logico, ci restituisce l'indirizzo IP corrispondente;
- *Dominio Inverso*: fornito un indirizzo IP, ci restituisce il nome logico corrispondente.

E' importante sottolineare che non sempre si otterrà una sola risposta dal server DNS contattato, bensì si otterranno da 0 a N risposte, a seconda del numero di schede di rete installate sul server che sto ricercando.

3.2 Sistema DNS

Il DNS è un sistema gerarchico e distribuito. Il sapere, ovvero il Database di corrispondenze nomi logici - indirizzi IP, è suddiviso tra i vari domini, ognuno dei quali mantiene il proprio Database di conoscenze su un proprio *nameserver*.

Per interagire con gli utenti, i nameserver utilizzano la porta 53/udp, attraverso la quale gestiscono query e risposte. DNS è, infatti, un sistema che si basa sul protocollo UDP, in quanto non necessita di particolari condizioni di affidabilità, quanto di un pronto e rapido accesso ai dati, appunto offerto dal protocollo UDP. Ad esempio, se un host che richiede a chi appartiene un dato indirizzo IP od un nome logico non ottiene una risposta da qualche nameserver entro un certo timeout, allora riformulerà la domanda.

Viceversa, per scambiare dati tra essi, i nameserver necessitano di un'alta affidabilità. Come tale, per trasferimenti dati "bulk", essi faranno riferimento alla porta 53/tcp. Ciò, in quanto questi trasferimenti necessitano di una garanzia di ricezione che solo il TCP può offrire.

3.3 Architettura DNS

La figura 3.1 riporta uno schema esemplificato dell'architettura DNS. Per capire come essa è strutturata, è opportuno fare un esempio pratico, ponendosi nei panni del client. Supponiamo

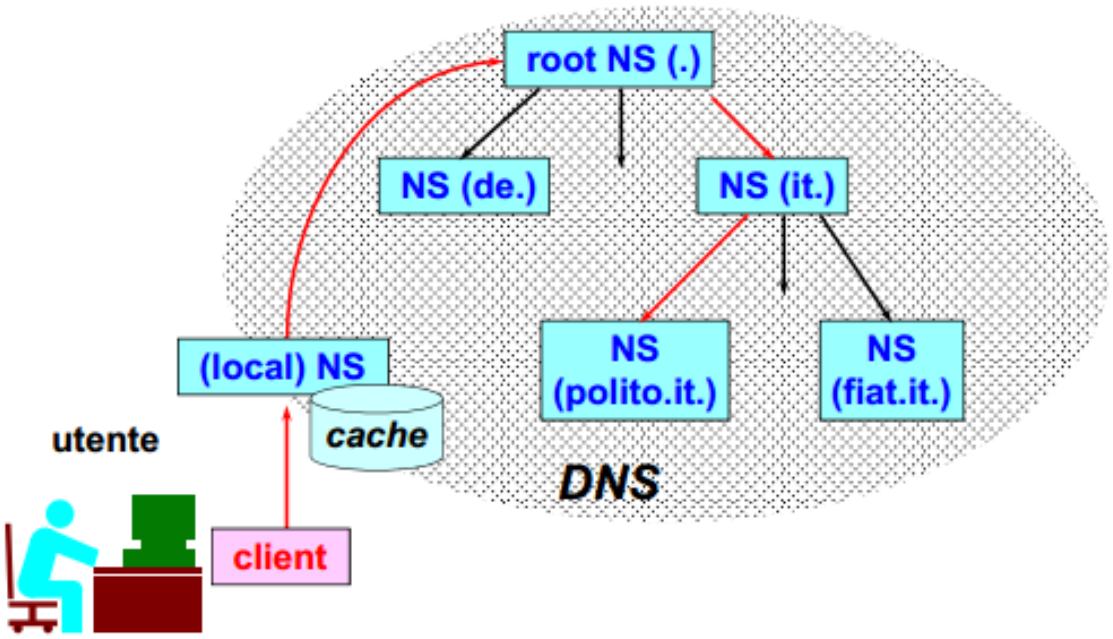

Figura 3.1: architettura DNS.

che questo client voglia conoscere l'indirizzo IP del server `www.polito.it`. Come primo passo, il client contatterà il nameserver che esso conosce, ovvero il "*(local) Nameserver*", noto perché fornito dal protocollo DHCP (Dynamic Host Configuration Protocol) oppure per configurazione manuale da parte del sistemista.

Il (local) Nameserver è un nameserver situato nello stesso dominio logico del client e che conosce tutti i nomi e gli indirizzi di tale dominio. Il motivo di una tale collocazione risulta dal fatto che, spesso, il nostro client potrebbe voler comunicare con gli altri host della suo stesso dominio, anziché uscire all'esterno. E' dunque un modo per ridurre il carico di rete. Il (local) Nameserver potrebbe già conoscere la risposta alla query del client. Se così è, fornirà la risposta, altrimenti inoltrerà la richiesta ad uno dei *Root Nameserver* mondiali.

I Root Nameserver gestiscono il dominio "." , ovvero un punto sottinteso (spesso dimenticato) rappresentante il mondo intero. In effetti, il reale indirizzo web non è "www.polito.it", bensì "www.polito.it.". Ovviamente, i Root Nameserver non conoscono tutte le corrispondenze esistenti. Avere un database di tali dimensioni sovraccaricherebbe ogni macchina. Infatti, come già detto,il DNS è un sistema gerarchico e distribuito e, come tale, il sapere è suddiviso tra i vari domini in maniera sempre più specifica a mano a mano che si scende nella gerarchia. Proprio per questo, i Root Nameserver non avranno risposta alla query del client, inoltrata dal suo (local) nameserver. Tuttavia, nel nome logico passato individueranno il dominio di primo livello "`it`" ed inoltreranno la query ai nameserver sotto di loro, con conoscenze più specifiche. Questi nameserver riceveranno la query a loro inoltrata ed individueranno in essa il dominio di secondo livello "`polito.it`". A quel punto, potranno o fornire la risposta al client, oppure inoltrare la query a qualche server sotto di essi. Alla fine, il client otterrà una risposta alla propria query e potrà contattare il server desiderato.

E' importante sottolineare che ogni nameserver usa salvare le query in una memoria cache per un tempo di default di 15 giorni (modificabile dai sistemisti), in modo da non dover inoltrare ogni volta la stessa richiesta al sistema DNS e ridurre il carico di rete.

3.4 Record DNS

Le risposte dei nameserver contengono le informazioni richieste sotto forma di uno o più "*Record DNS*", che possono essere dei seguenti tipi:

- *A* (Address) fornisce l'indirizzo IPv4 corrispondente ad un nome DNS;
- *AAAA* (Address) fornisce l'indirizzo IPv6 corrispondente ad un nome DNS;
- *PTR* (PoinTeR) fornisce il nome DNS corrispondente ad un indirizzo IP passato;
- *CNAME* (Canonical NAME) permette di collegare un nome DNS ad un altro. La risoluzione continuerà con il nuovo nome indicato dal record CNAME;
- *NS* (NameServer) indica un nameserver delegato per uno specifico dominio DNS;
- *SOA* (Start Of Authority) indica il nameserver primario per uno specifico dominio, assieme ad informazioni ausiliare quali l'indirizzo mail dell'amministratore, il numero seriale del dominio (utile per sapere se i dati del dominio sono stati variati) e diversi timer che regolano la frequenza di trasferimento e la durata di validità dei record;
- *MX* (Mail Exchanger) indica i server di posta elettronica delegati a ricevere la posta in ingresso per un dominio, indicando anche l'ordine con cui contattare i server (i server con maggiore priorità sono quelli con indice numerico minore);
- *HINFO* (Host Information) fornisce informazioni sull'host e sul suo sistema operativo;
- *TXT* (TeXT) trasporta informazioni testuali in formato libero;
- *WKS* (Well Known Services) fornisce l'elenco dei servizi attivi nel nodo.

3.5 Nameserver

Si è osservato come esistano più tipi di nameserver DNS, distinti in base al loro livello gerarchico ed alle loro funzioni. Di seguito ne è riportata una breve descrizione. Si tenga conto del fatto che ogni nameserver può assumere al contempo uno o più di questi ruoli.

3.5.1 Root Nameserver

Come già menzionato, i *Root Nameserver* gestiscono i "domini di primo livello". Sono 13 server logici mondiali (10 in USA, 2 in Europa e 1 in Giappone), con nomi da `a.root-servers.net` a `m.root-servers.net`. Questi server logici corrispondono a 356 server fisici (al 3 marzo 2013) sparsi per tutto il mondo. La pagina `http://root-servers.org/` contiene l'elenco completo ed aggiornato.

3.5.2 Primary Nameserver

I "*Primary Nameserver*" sono i nameserver master per un dato dominio, ossia possiedono la copia ufficiale dei dati del dominio, quella su cui è possibile eseguire modifiche, aggiunte e cancellazioni. Di regola, è obbligatorio che ogni Primary abbia almeno un "Secondary Nameserver". Il consiglio è sempre quello di avere due Secondary, di cui almeno uno off-site, di modo che uno di essi possa rimanere attivo anche se l'altro va in crash o si spegne a causa di malfunzionamento o blackout.

Record SOA

Come già accennato, il Record SOA (Start Of Authority) è il record DNS che mi fornisce l'identità del Primary Nameserver per un dato dominio. Di seguito, se ne riporta il formato:

```
nome [TTL] classe SOA
origine; hostname del NS
riferimento; email del gestore del NS (con "." al posto di "@")
aggiornamento; informazioni temporali
```

Si noti che il campo "classe" corrisponderà sempre a "IN", in quanto si opererà solamente nel campo di Internet.

Il campo aggiornamento, invece, riporta, tra parentesi tonde, cinque valori numerici interi, quali dei quali rappresentano intervalli di tempo in secondi:

- `serial`, numero seriale della versione dei dati, incrementato ad ogni modifica;
- `refresh`, intervallo di tempo prima di controllare il numero seriale (al reboot dei Secondary);
- `retry`, tempo di attesa prima di riprovare un trasferimento fallito;
- `expire`, tempo prima della scadenza dei dati, al termine del quale essi vanno aggiornati;
- *default*, TTL di default assegnato ai dati.

Di seguito, viene riportato un esempio di record SOA (quello del dominio `polito.it.`):

```
primary name server = leonardo.polito.it
responsible mail addr = root.leonardo.polito.it
serial  = 2015061601
refresh = 10800 (3 hours)
retry   = 1800 (30 minutes)
expire  = 1209600 (14 days)
default TTL = 86400 (1 day)
```

3.5.3 Secondary Nameserver

I *Secondary Nameserver* rappresentano le copie in sola lettura dei dati presenti sul Primary Nameserver del dominio. In pratica sono gli slave dello stesso, utili ad una corretta ripartizione delle query, evitando il sovraccarico del Primary.

Le copie dei dati che essi contengono sono aggiornate periodicamente allo scadere dei vari tempi del record SOA, oppure immediatamente a seguito di una modifica sul Primary. Quest'ultima opzione, però, è possibile sfruttarla soltanto tramite un'azione manuale del sistemista. Non esiste, dunque, un aggiornamento automatico immediato. È dunque possibile che, a seguito di modifiche sul Database del Primary, i dati presenti nei Secondary non siano stati ancora aggiornati e dunque non siano più validi.

Il trasferimento dei dati dai Primary del dominio ai Secondary dello stesso è eseguito tramite il trasferimento "bulk" affidabile sulla porta 53/tcp.

3.5.4 Forwarding Nameserver

I *Forwarding Nameserver* sono quelli che inoltrano le query alla gerarchia DNS per conto di altri nameserver che non possono (o non vogliono) agganciarsi direttamente al DNS. In pratica sono dei DNS proxy.

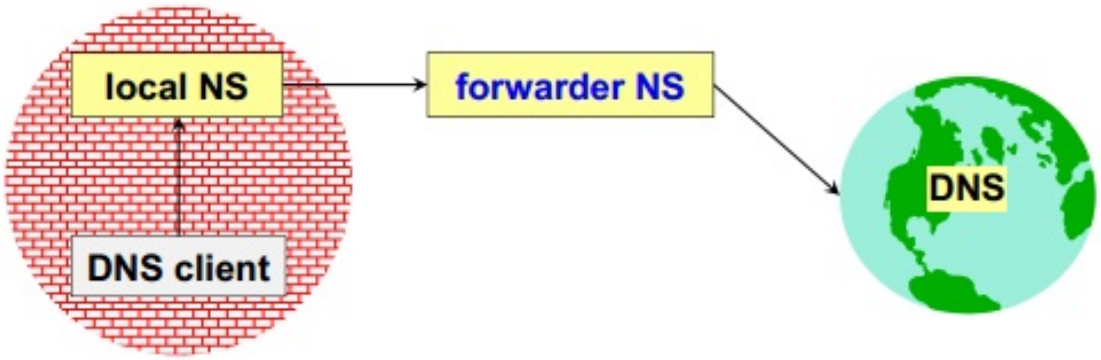

Figura 3.2: Schema del funzionamento di un Forwarding NS (fonte: slides prof. Lioy).

3.5.5 Caching Nameserver

I *Caching Nameserver*, se non assumono al contempo altri ruoli, non mantengono alcun dato permanente. Sono solamente usati per le loro funzionalità di caching, ovvero come server locali fittizi per ridurre il carico di rete.

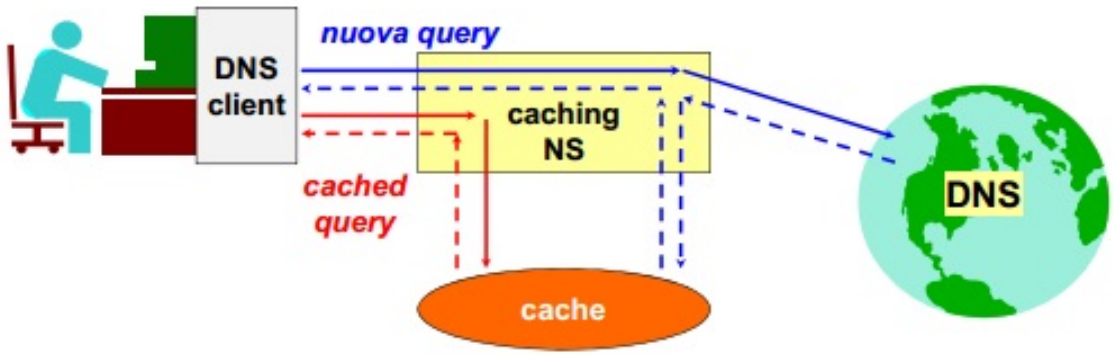

Figura 3.3: Schema del funzionamento di un Caching NS (fonte: slides prof. Lioy).

Le risposte fornite dal DNS possono essere marcate come "*authoritative*" o "*non authoritative*":

- sono authoritative le risposte ottenute direttamente da un nameserver primario;
- sono non authoritative le risposte ottenute tramite una cache (l'informazione originale potrebbe anche essere diversa).

3.6 Esempio di risoluzione di un indirizzo

Supponiamo di voler conoscere l'indirizzo IP del server `www.polito.it` e di volere emulare a mano il comportamento del local nameserver quando riceve questa query.

Il primo passo consisterà nel contattare uno dei root nameserver. Per farlo abbiamo bisogno anche noi di conoscere l'indirizzo di uno di essi e possiamo farlo consultando la pagina http://root-servers.org/. Volendo ottimizzare le prestazioni, notiamo che tre server sono localizzati a Torino (quelli gestiti da ICANN, ISC e Verisign) e sono quindi ottimi candidati come nostro root nameserver. Una rapida verifica con traceroute fa però vedere che, in base agli accordi di peering, il nameserver che dista meno hop dalla nostra rete è quello gestito da ISC ed ospitato presso TOPIX: `f.root-servers.net` ovvero 192.5.5.241. Siamo quindi pronti a compiere il primo passo:

```
$ nslookup -q=NS www.polito.it. 192.5.5.241
Server:         192.5.5.241
Address:        192.5.5.241#53

Non-authoritative answer:
*** Can't find www.polito.it.: No answer

Authoritative answers can be found from:
it      nameserver = a.dns.it.
it      nameserver = c.dns.it.
it      nameserver = m.dns.it.
it      nameserver = dns.nic.it.
it      nameserver = r.dns.it.
it      nameserver = nameserver.cnr.it.
a.dns.it        internet address = 194.0.16.215
c.dns.it        internet address = 194.0.1.22
m.dns.it        internet address = 217.29.76.4
r.dns.it        internet address = 193.206.141.46
dns.nic.it      internet address = 192.12.192.5
nameserver.cnr.it       internet address = 194.119.192.34
a.dns.it        has AAAA address 2001:678:12:0:194:0:16:215
c.dns.it        has AAAA address 2001:678:4::16
m.dns.it        has AAAA address 2001:1ac0:0:200:0:a5d1:6004:2
r.dns.it        has AAAA address 2001:760:ffff:ffff::ca
dns.nic.it      has AAAA address 2a00:d40:1:1::5
nameserver.cnr.it       has AAAA address 2a00:1620:c0:220:194:119:192:34
```

Conosciamo ora gli indirizzi dei nameserver che gestiscono il dominio `it` e possiamo quindi ripetere la stessa domanda ad uno di essi:

```
$ nslookup -q=NS www.polito.it. 192.12.192.5
Server:         192.12.192.5
Address:        192.12.192.5#53

Non-authoritative answer:
*** Can't find www.polito.it.: No answer

Authoritative answers can be found from:
```

```
polito.it       nameserver = leonardo.polito.it.
polito.it       nameserver = giove.polito.it.
polito.it       nameserver = ns1.garr.net.
giove.polito.it internet address = 130.192.3.24
leonardo.polito.it      internet address = 130.192.3.21
```

Conosciamo ora gli indirizzi dei nameserver che gestiscono il dominio `polito.it` e possiamo quindi chiedere ad uno di essi l'indirizzo del server che vogliamo contattare:

```
$ nslookup -q=A www.polito.it. 130.192.3.24
Server:         130.192.3.24
Address:        130.192.3.24#53

www.polito.it   canonical name = webfarm.polito.it.
Name:   webfarm.polito.it
Address: 130.192.182.33
```

Si noti che la risposta indica che `www.polito.it` è un alias: il vero nome del server è `webfarm.polito.it` e può essere contattato all'indirizzo 130.192.182.33.

Tutta questa procedura non verrà più ripetuta da un nameserver finché il TTL dell'informazione presente nella cache non sarà scaduto.

3.7 Caching DNS

Come si è già detto in precedenza, i nameserver usano memorizzare per un certo periodo, di minimo 15 giorni, le query effettuate, di modo da non dover ricercare ogni volta le stesse risposte all'interno della gerarchia, rallentando l'esecuzione delle applicazioni e aumentandone la latenza.

Questo lungo TTL delle informazioni ricercate, trova la sua spiegazione nel fatto che il DNS occupa una gran fetta del carico di rete. Si pensi solamente ai siti web che vengono visitati ogni giorno ed alla frequenza con cui gli utenti accedono ad essi. Se i nameserver fossero costretti ad inoltrare in ogni momento query alla gerarchia DNS, la rete sarebbe perennemente intasata e a rischio di crash. Memorizzando le query in una cache del nameserver locale per un certo periodo, ogni volta che un client della sottorete necessita di una risposta, allora sarà il (local) nameserver stesso a rispondere, evitando di intasare la rete con continui inoltri alla gerarchia.

Tuttavia, questo TTL costituisce anche un grosso problema per chiunque detenga un sito web. In effetti, capita molto spesso di avere la necessità di modificare gli indirizzi IP dei server sui quali è distribuito un sito web. Si pensi ad un'azienda che, stufa di avere troppi server nei propri stabilimenti, decide di dare in outsourcing la gestione del proprio sito web. In queste condizioni, chi fosse acceduto al sito sino a poco prima della variazione degli indirizzi IP dei server, si ritroverebbe ad avere nella cache del proprio (local) Nameserver un'informazione non più valida, con la conseguente impossibilità di accedere nuovamente al sito per un tempo uguale o poco minore di 15 giorni.

Su questo problema, purtroppo, non si è in grado di intervenire. Proprio per questo, l'unico modo per evitare simili disagi agli utenti consiste nel dovere, da parte di chi andrà a modificare gli indirizzi IP dei server, di mantenere ancora attivi anche i vecchi indirizzi per un tempo almeno pari al TTL dei dati nelle cache, prima di poterli rilasciare definitivamente.

3.8 Carico di rete

Si è già detto come il sistema DNS generi un carico estremamente alto per la rete e per i nameserver stessi, provocando ingenti aumenti di latenza nelle applicazioni. A tal proposito, si è osservato come i meccanismi di caching risolvessero un poco il problema. Tuttavia, per gestire in modo migliore il tutto ed evitare il sovraccarico della rete e dei server, occorre anche tener conto di come si organizzano i nameserver per la propria Intranet. A tale scopo, vengono presentate quattro possibili soluzioni:

1. primary aziendale gestito in outsourcing dall'ISP (ciò, però, è causa di continue query in rete);
2. primary aziendale all'ISP e caching in azienda;
3. primary aziendale all'ISP e secondary in azienda;
4. primary in azienda e secondary all'ISP.

3.9 AS112

Il problema già discusso del carico di rete generato dal traffico DNS, trova incremento in un numero molto alto di query che i nameserver ricevono, relativa alla risoluzione di indirizzi privati, ovvero di quegli indirizzi relativi a reti private, non instradati all'esterno dai router di bordo delle sottoreti, quali:

`10.0.0.0/8` `169.254.0.0/16` `172.16.0.0/12` `192.168.0.0/16`

Ovviamente, i nameserver DNS rispondono dicendo che non conoscono tali indirizzi, tuttavia tali richieste provocano un ulteriore aumento di traffico sulla rete e devono essere interrotte. Per fare ciò, è possibile marcare come "authoritative" il proprio (local) nameserver per tali reti, oppure seguire le istruzioni del progetto "*AS112*", reperibili al sito `http://public.as112.net/`.

3.10 Server "paranoici"

Una problematica molto recente, purtroppo in larga diffusione, riguarda la cosiddetta "black-Internet" (l'Internet nera). I "cattivi" utilizzano sempre più indirizzi illegali e fittizi, sfruttati senza il consenso del reale proprietario, per svolgere i loro sporchi affari. Sono indirizzi con un tempo vita estremamente breve, di poche ore al massimo, i quali vengono utilizzati per trattare di terrorismo, pedo-pornografia, droga ed altro.

Per far fronte a questo spiacevole problema da qualche tempo i sistemisti si sono attivati, introducendo un meccanismo di sicurezza che rende i server, per così dire, "paranoici". Questi server sono programmati di modo che, non appena ricevono una richiesta di accesso da parte di un indirizzo, verifichino la sua autenticità tramite due passaggi consecutivi:

1. richiedere, tramite record PTR, il nome logico corrispondente all'indirizzo IP che lo ha contattato;

2. ottenuto il nome logico, verificarne la sua correttezza richiedendo al DNS di risolvere tale nome e confrontando l'indirizzo IP ottenuto con quello del client che ha contattato il server.

Se i due indirizzi IP risultano identici, allora il server permette al client di accedere ai propri servizi, in caso contrario giudicherà l'indirizzo IP come illegale e rifiuterà l'accesso ai servizi.

Questa verifica sembra una cosa ovvia e semplice, ma bisogna tener conto che così non è, in quanto:

- molte aziende registrano i loro host solo sul dominio diretto e non su quello inverso;
- dominio diretto e inverso sono spesso gestiti separatamente e quindi sono possibili disallineamenti ed errori.

Proprio per questi motivi, un comune indirizzo IP valido potrebbe essere marcato come illegale da un server paranoico anche se non lo è.

Capitolo 4

La posta elettronica

4.1 Applicazioni di rete

La posta elettronica è un servizio Internet grazie al quale ogni utente abilitato è in grado di inviare e ricevere messaggi, utilizzando un computer connesso alla rete attraverso un proprio account di posta, registrato presso un provider del servizio. Lo scopo del servizio è lo scambio di messaggi tra due o più utenti, questo si effettuata tramite l'applicazione di rete ovvero un programma che consente la comunicazione tra due nodi.

L'applicazione di rete è composta da:

Indirizzi logici utilizzati per identificare gli end-point applicativi i quali consentono l'individuazione della persona nelle rete;

Il protocollo di rete usato per la comunicazione tra due end-point, che contengono il formato delle risposte e delle domande alle quali un server può rispondere;

Il formato dei dati che deve essere usato durante la comunicazione.

Inoltre per ciascun punto è necessario definire: il lessico (i caratteri ammessi nella comunicazione), la sintassi (la modalità con cui debbono essere composte le richieste e le risposte) e la semantica (il significato che assumono le domande e le risposte).

4.2 Indirizzi di posta elettronica

Ciascun utente può possedere una o più caselle di posta elettronica, sulle quali può ricevere i messaggi di posta elettronica. Quando lo desidera l'utente può consultare il contenuto della casella, scaricare la posta, eventualmente cancellarla e inviare messaggi agli altri utenti. L'accesso alla casella è normalmente controllato tramite id-utente e password. La modalità di accesso del servizio è quindi asincrona, poiché per la trasmissione non è necessario che mittente e destinatario siano contemporaneamente attivi o collegati. Inoltre la consegna dei messaggi non è garantita, nel caso in cui il server non sia stato in grado di consegnare il messaggio, il mittente non ne avrà un riscontro. Il mittente può anche richiedere una conferma di consegna o di letture, ma il destinatario potrà nuovamente decidere se inviare tale notifica.

Le specifiche per la rappresentazione dei messaggi di posta elettronica sono definite nel RFC-822. In particolare gli indirizzi di posta elettronica devono essere nella forma:

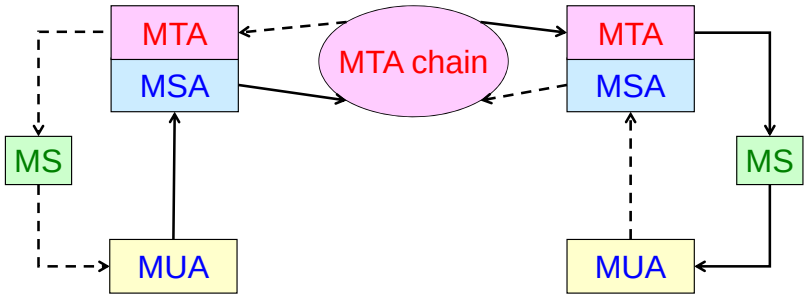

Figura 4.1: l'architettura MHS.

casella_postale `@` *dominio_postale*

ove la *casella_postale* è un identificativo dell'utente all'interno del dominio specificato dal *dominio_postale*. Si noti che la casella può corrispondere ad uno username associato ad un login oppure essere un identificativo virtuale, così come il dominio può essere un nome DNS o l'indirizzo IP di un host oppure essere un dominio logico che non corrisponde ad alcun host fisico.

Un indirizzo RFC-822 può essere scritto puro o essere preceduto o seguito da un commento, come nei seguenti esempi:

- indirizzo puro – `lioy@polito.it`
- indirizzo preceduto da un commento – `"Antonio Lioy" <lioy@polito.it>`
- indirizzo seguito da un commento – `lioy@polito.it (Antonio Lioy)`

La possibilità di inserire un commento viene spesso usata per trarre in inganno il destinatario circa la reale identità del mittente, come nel seguente caso:

`"Antonio Lioy" <al86@yahoo.it>`

Il dominio postale è un concetto logico, ma per trasmettere la posta con SMTP occorre identificare un host fisico a cui è assegnato un preciso indirizzo IP. Interrogando il DNS è possibile sapere il dominio coincide con un host oppure se è un dominio logico con associato specifici server (MX, Mail eXchanger) che ne gestiscono la posta in arrivo:

- (dominio coincide con un host?) `nslookup -q=A` *dominio_postale*
- (dominio ha degli MX?) `nslookup -q=MX` *dominio_postale*

Se entrambe le query danno risposta negativa, allora il dominio non è valido per l'invio di posta elettronica.

4.3 L'architettura MHS

Per la trasmissione e la lettura dei messaggi di posta elettronica TCP/IP viene usata l'architettura *MHS (Message Handling System)* caratterizzata da quattro componenti logici che interagiscono fra di loro: MUA, MSA, MTA, MS (Fig. 4.1).

Il *MUA (Message User Agent)* è l'applicazione che si interfaccia con l'utente per permettergli di spedire e ricevere posta elettronica.

Il MUA non invia direttamente il messaggio al destinatario ma si appoggia ad un server di posta locale, noto come *MSA (Message Submission Agent)*, il quale gestisce tutti gli eventuali problemi agendo come primo anello della catena di trasmissione.

Questa catena è composta da uno o più *MTA (Message Transfer Agent)* che ricevono un messaggio e lo inoltrano al prossimo server in base all'instradamento appropriato per il destinatario. In questo gli MTA sono simili ai router perché gli uni instradano messaggi e gli altri pacchetti di rete.

L'ultimo MTA della catena consegna il messaggio al server finale, denominato *MS (Message Store)* perché gestisce le caselle di posta degli utenti. Il destinatario userà il suo MUA per sapere se c'è della nuova posta nella sua casella, leggerla ed eventualmente cancellarla.

Si noti che, se il destinatario decide di rispondere al mittente, non è detto che il messaggio di risposta segua la stessa strada di quello ricevuto.

Questo metodo di trasferimento dei messaggi di posta elettronica mima il trasferimento della posta cartacea: cassette di posta pubbliche (MSA), centri di smistamento a vari livelli – cittadino, regionale, nazionale e internazionale (MTA) – e buca delle lettere personale (MS).

MSA e MTA implementano un sistema *store-and-forward* caratterizzato da ricezione di un messaggio, sua memorizzazione (temporanea) e quindi invio al prossimo server. Ciò permette di gestire eventuali problemi di congestione o guasti sulle linee di trasmissione: nel caso che il prossimo MTA non sia raggiungibile, un MTA ritenterà l'invio per i tre giorni successivi e solo dopo tale lasso di tempo eliminerà il messaggio dalla propria coda, eventualmente segnalando il problema al mittente.

Si noti che i messaggi di posta non vengono trasmessi direttamente dal MUA del mittente a quello del destinatario ma sono salvati sui nodi intermedi che attraversano. Ciò comporta notevoli problemi di sicurezza in relazione sia alla privacy sia all'integrità dei messaggi che possono essere letti e modificati sui server intermedi.

Per la trasmissione della posta elettronica sono usati diversi protocolli di basso livello, così detti perché si occupano solo di trasportare i messaggi e non entrano nel merito del loro contenuto. Essi sono:

- SMTP (Simple Mail Transfer Protocol), con porta di default 25/tcp per gli MTA e 587/tcp per gli MSA, usato per le comunicazioni MUA-MSA, MSA-MTA e MTA-MTA;
- POP (Post Office Protocol), con porta di default 110/tcp per il MS, usato per la comunicazione MUA-MS;
- IMAP (Internet Message Access Protocol), con porta di default 143/tcp per il MS, usato per la comunicazione MUA-MS.

Il protocollo SMTP è di tipo push perché permette di inviare un messaggio ad un server mentre POP e IMAP sono di tipo pull perché permettono di contattare un server per gestire gli eventuali messaggi presenti su di esso. Maggior dettagli su questi protocolli sono forniti nelle sioni ad essi dedicate.

Il MUA può essere realizzato secondo due diverse filosofie: client-server e webmail.

Un MUA client-server (Fig. 4.2) ...

Un MUA di tipo webmail è un'applicazione web che consente di gestire uno o più account di posta elettronica attraverso un web browser. Questo servizio ci consente di leggere

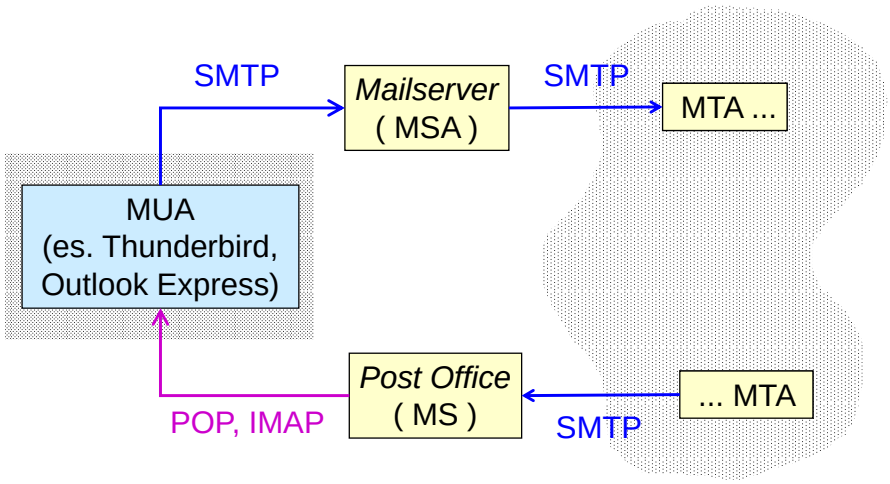

Figura 4.2: MUA collegato in modalità client-server.

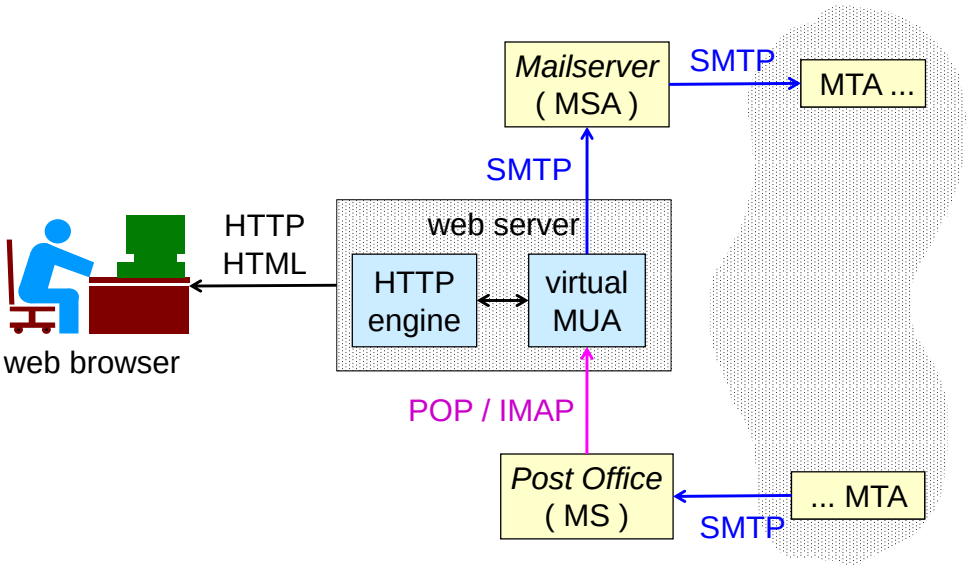

Figura 4.3: MUA realizzato tramite “webmail”.

i messaggi anche su computer differenti, attraverso la creazione di un virtual MUA (condiviso da tutti gli utenti), ma con l'introduzione di questo sistema di lettura si aggiungono altri problemi legati alla sicurezza, inoltre la protezione delle informazioni dell'utente risulta ancora più difficile.

4.4 Il formato RFC-822

Nel documento RFC-822 sono contenute le informazioni che definiscono un messaggio di posta elettronica:

- formato di un messaggio (header + body);
- body di puro testo (contenente solo parole).

In seguito alle necessità dell'utente il formato RCF-822 si evolse nel MIME:

- estensione multimediale di RCF-822;
- boby può includere anche dati non di testo;
- alcuni campi aggiuntivi nell'header;

Formato RFC-5322 (ex RFC-822):

- solo caratteri US-ASCII codifica a 7 bit (128 caratteri disponibili), 1 bit utilizzato per il controllo dell'errore o forzato a zero;
- righe terminate con <CR><LF>;
- messaggi composti da header+body;

Un messaggio RFC-822 inizia con una parte di intestazione (header) costituita da righe che iniziano tutte con una parola chiave, seguita dal carattere ":" e quindi Se una riga inizia con uno o più spazi allora è una continuazione del contenuto della riga precedente (che probabilmente superava il limite di 76 caratteri).

Il cosiddetto body di un messaggio RFC-822 è separato dall'header da una riga vuota (ossia contenente solo la coppia di caratteri CR-LF) e contiene il messaggio vero e proprio.

La tabella in figura 4.4 elenca alcuni delle principali parole chiave usate negli header RFC-822. I campi che identificano i destinatari (To, CC, Bcc) possono comparire più volte e/o contenere elenchi di destinatari separati da virgole. I destinatari specificati in Bcc riceveranno una copia del messaggio ma non compariranno tra i destinatari dello stesso. I campi Received vengono inseriti dai vari nodi intermedi attraversati (MSA, MTA, MS) e forniscono una sorta di traccia del percorso fatto dal messaggio nella sua strada dal mittente al destinatario. La ricevuta di ritorno (se richiesta) può essere inviata dal MS (all'atto della ricezione oppure della lettura del messaggio da parte dell'utente) oppure dal destinatario stesso (tramite il suo MUA), ma potrebbe anche non essere inviata del tutto.

Nota pratica: si presti particolare attenzione a non lasciare vuoto il campo Subject perché aumenta la probabilità che il messaggio sia considerato "spam" e scartato automaticamente dal MS o MUA del ricevente.

keyword	*informazione fornita*
From:	mittente (logico)
Sender:	mittente (operativo)
Organization:	organizzazione del mittente
To:	destinatario
Subject:	argomento
Date:	data e ora di spedizione da parte del MUA
Received:	informazioni relative al passaggio su un nodo intermedio
Message-Id:	identificativo di spedizione (inserito da MSA)
CC:	(Carbon Copy) in copia a
Bcc:	(Blind Carbon Copy) in copia nascosta a
Return-Receipt-To:	indirizzo a cui spedire la ricevuta di ritorno

Figura 4.4: i principali header RFC-822.

4.5 Il protocollo SMTP

4.5.1 Comandi di base:

`HELO` *hostname* (identifica l'host mittente tramite FQDN o indirizzo IP);

`MAIL FROM:` *return-path* (indirizzo mail del mittente);

RCPT TO: forward-path (identifica il destinatario);

DATA (tutte le righe successive contengono un messaggio RFC-822 sino alla riga esclusa che contiene <CR><LF>);

QUIT (termina la trasmissione).

Altri comandi:

RSET (annulla comandi sinora impartiti);

VRFY indirizzo -postale (verifica se l'indirizzo è valido);

EXPN indirizzo-di-una-lista-postale (elenca iscritti ad una lista);

HELP (fornisce aiuto);

NOOP (operazione nulla).

Comandi obsoleti:

SEND FROM: return-path (identifica mittente);

SOML FROM: return-path (send-or-mail);

SAML FROM: return-path (send-and-mail).

4.5.2 Codici di stato SMTP

Ogni risposta del server inizia con un codice di stato numerico a tre cifre `XYZ` La prima cifra fornisce il major status dell'azione richiesta:

- `X = 1` positive preliminary (non sono presenti errore per il momento);
- `X = 2` positive completion (una fase del protocollo si è conclusa con esito positivo);
- `X = 3` positive intermediate;
- `X = 4` transient negative completion (errore temporaneo, es. la memoria del disco risulta piena);
- `X = 5` permanent negative completion (errore persistente, es. indirizzo o comando errato).

La seconda cifra indica la tipologia di errore:

- Y = 0 syntax (errore di sintassi);
- Y = 1 information (errore nelle informazioni);
- Y = 2 connections (errore nella connessione);
- Y = 5 mail system (errore nel sistema di posta).

Infine la terza cifra fornisce un modo per distinguere codici di risposta appartenenti alla stessa classe.

4.5.3 Limiti SMTP e RFC-822

Sono presenti dei limiti ben definiti nell'utilizzo del protocollo SMTP poiché è necessario dimensionare la memoria necessaria a immagazzinare le informazioni e per l'implementazione del programma altrimenti sorgerebbero problemi di compatibilità.

Limiti quantitativi:

- mailbox inferiore a 64 caratteri (indirizzo prima della @);
- domain inferiore 64 caratteri (indirizzo dopo la @);
- reverse/forward-path inferiore a 256 caratteri (percorso tra due destinazioni);
- comando/risposta inferiore a 512 caratteri;
- linea di testo inferiore a 1000 caratteri (Deve);
- linea di testo inferiore 80 caratteri (Dovrebbe/è consigliabile);
- numero di destinatari inferiore a 100 (limite imposto automaticamente dal sistema);
- dimensione totale del messaggio di posta, compresi allegati, è di 64 kB (MTA possono decidere di eliminare i messaggi più grandi, senza alcuna segnalazione di errore).

4.5.4 Esempio SMTP/ RFC-822:

```
S: 220 duke.colorado.edu ...
C:    HELO leonardo.polito.it
S: 250 Hello leonardo.polito.it ... Nice to meet you!
C:    MAIL FROM: cat
S: 250 cat ... Sender ok
C:    RCPT TO: franz
S: 250 franz ... Recipient ok
C:    DATA
S: 354 Enter mail, end with "." on a line by itself
C:    From: cat@athena.polito.it (Antonio Lioy)
C:    To: franz@duke.colorado.edu
C:    Subject: vacanze
C:
C:    Ciao Francesco,
C:    ti rinnovo l'invito a venirmi a trovare nelle tue
C:    prossime vacanze in Italia. Fammi sapere
C:    quando arrivi.
C:    Antonio
C:    .
S: 250 Ok
C:    QUIT
S: 221 duke.colorado.edu closing connection
```

4.6 Il protocollo ESMTP

Il nuovo protocollo *ESMTP (Extended SMTP)*, definito in RFC-1869 e quindi incorporato in RFC-5321, è un'evoluzione di SMTP. I client ESMTP devono presentarsi con il comando EHLO (invece che HELO), ad indicare che è in grado di parlare la nuova versione. Nel caso in cui il server ricevente parli ESMTP, deve dichiarare le estensioni che supporta, una per riga, nella sua risposta all'EHLO, altrimenti segnalerà errore (in quanto il comando EHLO è errato in SMTP) ed il dialogo proseguirà con SMTP.

4.6.1 Estensione comandi standard:

SEND (Invia come mail);

SOML (Invia come mail o terminale);

SAML (Invia come mail e terminale);

EXPN (Espandi la mailing list);

HELP (Informazioni di aiuto);

TURN;

BITMIME (RFC-1652 nel corpo del comando data può ricevere dati da 8 bit);

SIZE dimensione;

MAIL FROM: address SIZE=dimensione (RFC-1870 dichiara la massima dimensione accettabile dal server o la dimensione del messaggio da inviare);

PIPELINE (RFC-1854 invio di più comandi senza attendere risposta per ognuno di essi).

4.6.2 Estensione DSN (Delivery Status Notification)

Estende il comando RCTP con:

- NOTIFY=notify-list
  valori possibili: NEVER, SUCCESS (Successo), FAILURE (Fallimento), DELAY(Ritardo Store and Forward);

- ORCPT=original-recipient
  specifica il destinatario originale.

Estende il comando MAIL con

- RET=returned-message
  valori possibili: FULL (Rimanda indietro tutto il messaggio), HDRS (Rimanda solo l'intestazione);

- ENVID=sender-id
  identificativo creato dal mittente.

4.6.3 Esempi ESMTP

Un esempio di server ESMTP che capisce solo il protocollo base ma non supporta nessuna estensione è dato dalla seguente transazione:

```
S: 220 mail.polito.it - SMTP service ready
C:    EHLO mailer.x.com
S: 250 Hello mailer.x.com - nice to meet you!
```

Invece in questa transazione il server supporta due estensioni ESMTP:

```
S: 220 mail.polito.it - SMTP service ready
C:   EHLO mailer.x.com
S: 250-Hello mailer.x.com - nice to meet you!
S: 250-SIZE 26214400
S: 250 8BITMIME
```

Infine ecco il caso di un server che non conosce il protocollo ESMTP:

```
S: 220 mail.polito.it - SMTP service ready
C:    EHLO mailer.x.com
S: 500 Command not recognized: EHLO
```

4.6.4 SMTP-Auth

Estensione di ESMTP definita in RFC-4954, comando AUTH + opzione di MAIL FROM, è necessaria l'autenticazione da parte dell'utente prima di poter inviare i messaggi di posta. Utile contro lo spamming: dopo il comando EHLO server invia meccanismi di autenticazione supportati, il client ne sceglie uno, viene eseguito il protocollo di autenticazione e nel caso in cui l'autenticazione fallisca il canale viene chiuso. Esistono differenti metodi di autenticazione: metodo login (vengono inserite prima la Username e poi la Password), metodo plain (Sintassi: AUTH PLAIN id-pwd[Base64], dove ip-pwd è definito come: [Authorize-id]\0 Authentication-id\0 pwd).

Esempio di autenticazione fallita perché il client propone un metodo non supportato dal server:

```
S: 220 example.polito.it - SMTP service ready
C:    EHLO mailer.x.com
S: 250-example.polito.it
S: 250 AUTH LOGIN CRAM-MD5 DIGEST-MD5
C:    AUTH PLAIN
S: 504 Unrecognized authentication type
```

AUTH metodo LOGIN:

```
1 S: 220.example.polito.it - SMTP service ready
2 C:    EHLO mailer.x.com
3 S: 250-example.polito.it
4 S: 250 AUTH LOGIN CRAM-MD5 DIGEST-MD5
5 C:    AUTH LOGIN
6 S  334 VXNlcm5hbWU6       Username:
7 C:    bGlveQ==               lioy
8 S: 334 UGFzc3dvcmQ6       Password:
9 C:    YW50b25pbw==           antonio
10 S: 235 authenticated
```

Per capire appieno l'esempio bisogna decodificare i messaggi scritti in base64 (righe 6-9), ad esempio tramite `openssl` (questa operazione è gia stata fatta nell'esempio ed il relativo risultato è mostrato in colore rosso):

```
VXNlcm5hbWU6
bGlveQ==
UGFzc3dvcmQ6
YW50b25pbw==
```
→ "openssl enc -d -a" →
```
Username:
lioy
Password:
antonio
```

Questo metodo di autenticazione è molto lungo perché richiede l'invio di sei messaggi. In questo tipo di autenticazione i dati sono trasmessi in codifica Base64 per evitare errori di trasmissione nel caso che vengano usati dati con MSB=1 (es. lettere accentate all'interno di username o password).

AUTH metodo PLAIN:

```
1 S: 220 example.polito.it - SMTP service ready
2 C:    EHLO mailer.x.com
3 S: 250-example.polito.it
4 S: 250 AUTH LOGIN PLAIN
```

```
5 C:    AUTH PLAIN bGlveQBsaW95AGFudG9uaW8=      lioy\0lioy\0antonio
6 S: 235 authenticated
```

Anche in questo caso bisogna decodificare la risposta del client per capire bene lo scambio (il risultato è già stato inserito in rosso nell'esempio):

`bGlveQBsaW95AGFudG9uaW8=` → `"openssl enc -d -a"` → `lioy\0lioy\0antonio`

Il metodo Plain risulta veloce del metodo Login poiché l'autenticazione richiede solamente due messaggi, ma è ugualmente insicuro perché chiunque possa leggere i dati in transito potrà effettuarne la decodifica e quindi venire a conoscenza di username e password dell'utente.

4.7 Il protocollo POP

POP (Post Office Protocol) è un protocollo di livello applicativo che permette di accedere ad una casella di posta elettronica presente su un server (POP server) e di scaricare sul computer le e-mail del proprio account. Dal punto di vista dell'architettura MHS realizza dunque il collegamento tra MUA e MS.

Il protocollo POP ha intrinsecamente bisogno dell'autenticazione ed è talvolta usato, in modo improprio, in assenza di SMTP-auth, per autenticare l'utente prima di spedire posta.

Di questo protocollo esistono due principali versioni, POP2 e POP3, che funzionano attraverso comandi radicalmente differenti e a cui sono attribuite porte diverse; delle due versioni l'unica effettivamente utilizzata è POP3.

POP ha come principale concorrente il protocollo IMAP, più evoluto ma ancora poco diffuso.

4.7.1 Formato comandi e risposte

I comandi POP sono stringhe di caratteri ASCII (byte) terminate dalla sequenza di chiusura "CR LF". Ogni stringa di comando contiene una parola chiave, in genere formata da tre o quattro caratteri, più una serie di parametri opzionali. Come separatore viene usato un singolo carattere di spazio e i comandi sono case-insensitive.

Le risposte POP sono anch'esse stringhe di caratteri ASCII separate da un carattere di spazio e terminate dalla solita sequenza "CR LF". A differenza dei comandi esse sono precedute da un *indicatore di stato* ed inoltre sono possibili risposte su più righe, nel qual caso l'ultima riga conterrà solo la sequenza "CR LF". Attualmente esistono solo due possibili indicatori di stato, `+OK` e `-ERR`, ed entrambi devono essere rigorosamente scritti in maiuscolo. Tra i due indicatori vale la pena approfondire la gestione degli errori: il server risponde con `-ERR` quando il client invia una richiesta errata, non supportata o corretta ma spedita nello stato sbagliato (es. `RETR` prima di `USER`).

4.7.2 Comandi POP obbligatori

QUIT : chiede l'uscita del server POP3;

STAT : richiesta di informazioni sui messaggi della casella di posta contenuti sul server, è seguita da una risposta del server nella sintassi

`+OK` nmsg totalmsgsize

ove "nmsg" indica il numero di messaggi e la "totalmsgsize" la dimensione totale degli stessi;

LIST [***msgid***] : richiesta di visualizzare informazioni circa il messaggio identificato da "msgid" o, nel caso in cui il campo "msgid" sia vuoto, informazioni su tutti i messaggi nella casella di posta elettronica. Tale richiesta è seguita da una risposta del server nella sintassi

`+OK` *msgid msgsize*

ove *msgid* indica l'identificativo e *msgsize* la dimensione del messaggio;

RETR msgid : richiesta di recuperare il messaggio "msgid" lasciandone una copia sul server;

DELE msgid : richiesta di cancellare il messaggio "msgid" definitivamente;

NOOP : richiesta di emettere una risposta affermativa (`+OK`) con lo scopo di mantenere la connessione attiva;

RSET : richiesta di tornare allo stato iniziale, ovvero di eliminare tutti i marchi di cancellazione assegnati ai messaggi dal comando `DELE`, facendo sì che non vengano cancellati nella fase di aggiornamento.

4.7.3 Comandi POP opzionali

USER *mailbox* : dichiara lo username associato alla casella di posta (mailbox) a cui l'utente desidera accedere;

PASS : indica la password associata alla casella di posta dichiarata nel comando USER;

APOP *mailbox digest* : fornisce un metodo di autenticazione alternativo, che non richiede l'invio della password in rete, sostituendola con la stringa "digest", elimina il rischio che venga intercettata, ;

AUTH *mechanism* : fornisce un metodo di identificazione alternativo a username e password;

TOP *msgid nline* : fa sì che il server spedisca le intestazioni del messaggio, una linea vuota che separa le intestazioni dal corpo del messaggio "msgid", ed infine le prime "nline" linee del corpo del messaggio;

UIDL *msgid* : fornisce una stringa di identificazione del messaggio "msgid" univoca per tutte le sessioni.

4.7.4 Conversazione POP3

Quando un client vuole scaricare la posta, deve stabilire una connessione TCP sulla porta 110/tcp del server POP3. Stabilita la connessione, il server si presenta e i due cominciano a scambiarsi comandi e risposte finché la posta non è stata scaricata, a questo punto, se richiesto, la posta scaricata viene cancellata dalla casella postale e infine la connessione viene chiusa.

La conversazione tra client e server POP3 può essere suddivisa in tre fasi sequenziali.

La prima fase è quella di autorizzazione in cui il client deve autenticarsi ed il server verificare che le credenziali fornite permettano l'accesso alla casella postale. Se la verifica ha esito positivo, il server accede alla casella postale dell'utente e ne impedisce l'uso da parte di altri client (lock).

Si entra quindi nella fase di transazione, in cui il client esamina la posta, eventualmente cancellando quella che non desidera conservare sul server.

Infine si entra nelo stato di aggiornamento (tramite il comando `QUIT`) in cui il server elimina dalla casella postale tutti i messaggi per i quali era stata richiesta la cancellazione nella fase precedente, chiude la connessione e sblocca la casella postale (unlock) permettendone quindi l'eventuale accesso da parte di altri client.

```
S: +OK POP3 server ready <7831.84549@pop.polito.it>
C:    USER lioy
S: +OK password required for lioy
C:    PASS antonio
S: +OK lioy mailbox locked and ready
C:    STAT
S: +OK 2 320
C:    LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C:    RETR 1
S: +OK 120 octets
S: ... messaggio n. 1 ...
S: .
C:    DELE 1
S: +OK message 1 deleted
C:    RETR 2
S: +OK 200 octets
S: ... messaggio n. 2 ...
S: .
C:    QUIT
S: +OK POP3 server signing off
```

4.8 Il formato MIME

4.8.1 Introduzione

MIME (Multipurpose Internet Mail Extensions) è uno standard per la rappresentazione di informazioni trasmissibili in Internet, nato per superare i limiti dei messaggi di posta elettronica in formato RFC-822:

- canale a 7 bit (impossibile trasmettere dati binari, quali audio o immagini, a meno di trasformarli in dati alfabetici[1]);
- caratteri US-ASCII a 7-bit (impossibile trasmettere caratteri di altri alfabeti, che tipicamente richiedono 8 o 16 bit);
- righe inferiori a 1000 caratteri (ossia al massimo dopo 1000 caratteri deve essere presente la coppia di caratteri CR-LF).

Oggi il formato MIME viene però usato anche in molti altri contesti, come ad esempio per trasmettere dati nel web o identificare dati in memoria.

MIME è descritto da cinque documenti IETF:

- RFC-2045 specifica gli header SMTP per messaggi MIME;
- RFC-2046 definisce la gestione dei diversi tipi di contenuto;
- RFC-2047 definisce estensioni per gli header RFC-822 che fanno uso di caratteri non US-ASCII;
- RFC-2048 definisce le procedure per la registrazione, all'interno di MIME, di nuove caratteristiche (ad esempio nuovi formati dati) nel rispetto dei limiti dell'estensibilità di MIME;
- RFC-2049 definisce i livelli di conformità e fornisce esempi di uso di formati MIME.

4.8.2 Header MIME

In aggiunta ai normali header RFC-822, MIME definisce i seguenti header addizionali:

- `Mime-Version`
- `Content-Type`
- `Content-Transfer-Encoding`
- `Content-Id`
- `Content-Description`
- `Content-Disposition`.

[1] Ad esempio tramite il programma `uuencode` o `binhex`

Mime-Version dichiara l'utilizzo delle estensioni di MIME, secondo la versione indicata, anticipando così la presenza di altri campi specifici. Può assumere il valore 1.0 (unica versione di MIME esistente) o rimanere vuoto se il messaggio è scritto secondo il vecchio standard RFC-822, con il quale è conservata dunque la compatibilità.

Content-Type indica la natura del dato tramite la specificazione di tipo, sottotipo e ulteriori parametri utili, permettendo al ricevente di scegliere il meccanismo più adatto per presentare i dati; ha la seguente sintassi:

$$\textit{tipo}\ \texttt{/}\ \textit{sottotipo}\ [\ \texttt{;}\ \textit{nomeParametro}\ \texttt{=}\ \textit{valoreParametro}\ \texttt{,}\ \ldots\ ]$$

L'uso dei parametri è opzionale e, nel caso ce ne sia più di uno, sono separati da una virgola. I seguenti sono alcuni esempi di tipo e sottotipo:

- `text/plain` – normale testo ASCII (7 bit, alfabeto US-ASCII), parametro opzionale `charset`=... richiede tipicamente anche `encoding`=...;
- `text/html`, `text/xml`, `text/css`, `text/csv` – testo in uno specifico linguaggio o formato indicato dal sottotipo;
- `video/mpeg`, `video/quicktime` – video in formato MPEG o Quicktime;
- `audio/basic` – audio monofonico, 8 bit, codifica mu-law ISDN, campionato a 8 kHz;
- `image/gif`, `image/jpeg` – immagine in formato GIF o JPEG con codifica JFIF;
- `multipart/mixed` – il body contiene parti diverse, da visualizzare in sequenza;
- `multipart/parallel` – il body contiene parti diverse, da visualizzare senza un ordine specifico;
- `multipart/alternative` – contiene gli stessi dati in formato diverso, in ordine crescente di fedeltà, non fornisce garanzie di coerenza tra i diversi formati;
- `multipart/digest` – messaggio contenente più messaggi di posta (parti tutte message/rfc822), utile per ricevere un solo messaggio giornaliero da una mailing list;
- `message/rfc822` – contiene un messaggio di posta elettronica mail (ad esempio per farne il forward);
- `application/postscript`, `application/pdf` – documento Postscript, PDF;
- `application/vnd.ms-excel`, `application/vnd.ms-powerpoint` – documento MS Excel, MS Powerpoint (in generale i sottotipi che iniziano con `vnd` sono definiti e gestiti da uno specifico *vendor*);
- `application/octet-stream` – formato applicativo generico (flusso di byte).

Un elenco completo delle definizioni esistenti (gestito dalla IANA) è reperibile al seguente indirizzo:

http://www.iana.org/assignments/media-types/

In generale il formato MIME è espandibile: le sue definizioni includono metodi per aggiungere nuovi valori ad alcuni attributi MIME, quali `Content-type`, `Content-transfer-encoding` e `Content-disposition`. Fino a quando un'estensione non è documentata in un RFC è da considerarsi sperimentale e deve iniziare con `x-` o `X-` (dall'inglese *eXperimental*).

L'header *Content-Transfer-Encoding* indica la codifica dei dati usata. Tale codifica è volta alla trasmissione sul canale di dati non naturalmente rientranti nei limiti imposti dal meccanismo di trasporto (es. SMTP) implementato nel sistema di posta. Questo campo può assumere valori in due categorie: non codifiche e codifiche MIME, entrambe discusse più avanti.

L'header *Content-Id* è un identificativo univoco del messaggio creato dal mittente, opzionale e poco usato.

L'header *Content-Description* è una descrizione testuale dei dati generata dal mittente, opzionale e poco usato.

L'header *Content-Disposition* è un campo che serve al ricevente per decidere come trattare il contenuto ed ha la seguente sintassi:

disposizione [`;` *nomeParametro* `=` *valoreParametro* `,` ...]

ove *disposizione* può assumere i valori `inline` (parte integrante del corpo del messaggio) o `attachment` (allegato autonomo, da aprire esplicitamente). Tra i parametri più spesso usati si citano `filename`, `creation-date`, `modification-date`, `read-date` e `size`, tutti con un nome auto-esplicativo. Ecco un esempio d'uso di questo header:

```
Content-Disposition: inline;
   filename=fish.gif, creation-date="Mon, 1 Apr 2013 08:37:46 +0100"
```

Non codifiche MIME

Con *non codifiche MIME* si indicano tre dei possibili valori assumibili dall'header Content-Transfer-Encoding, ovvero `7bit`, `8bit` e `binary`. Questi valori indicano che nessuna operazione di codifica è stata effettuata sul contenuto del messaggio ma, allo stesso tempo, forniscono informazioni sul tipo di dati contenuti nel messaggio stesso (indicando così implicitamente il tipo di canale di trasmissione adatto per trasportare il messaggio):

- `7bit` – il mittente garantisce che il messaggio abbia righe non superiori a 1000 caratteri e che i caratteri siano tutti ASCII 7-bit (ovvero byte con MSB=0), rendendo così il messaggio adatto alla trasmissione su qualunque canale;

- `8bit` – il mittente garantisce che il messaggio abbia righe non superiori a 1000 caratteri ma possono essere presenti caratteri non appartenenti al set ASCII 7-bit (ovvero byte con MSB=1) che potrebbero essere alterati nella trasmissione su canali non *8-bit clean*;

- `binary` – indica che il contenuto del messaggio è in formato binario (immagine, audio, ...), il mittente non fornisce garanzie né sulla lunghezza delle righe né sull'appartenenza dei caratteri al set ASCII 7-bit, rendendo necessaria la trasmissione solo su canali *8-bit clean*.

bin	*b64*	*bin*	*b64*	*bin*	*b64*	*bin*	*b64*
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Figura 4.5: tabella di conversione Base64.

Codifiche MIME

Corrispondono a due dei possibili valori, `base64` e `quoted-printable`, assunti dall'header Content-Transfer-Encoding quando un'operazione di codifica è stata applicata ai dati.

La codifica *base64* è preferita per trasmettere dati binari. Essa considera il contenuto originale come un flusso di bit e li raggruppa in gruppi di 6 bit, invece che di 8 bit come siamo soliti fare. Se i dati originali non sono un multiplo di 6 bit, vengono aggiunti bit con valore zero. Quindi ad ogni gruppo di 6 bit viene fatto corrispondere uno specifico carattere ASCII, secondo una specifica tabella di conversione (fig. 4.5) che contiene in tutto 64 caratteri (perché $2^6 = 64$). La sequenza risultante viene divisa in righe di 76 caratteri, tranne l'ultima riga che può anche avere una dimensione inferiore.

Si noti che ad ogni 6 bit del dato viene fatto corrispondere un carattere ASCII, ossia 8 bit. Si avrà dunque un aumento della dimensione del messaggio pari a:

$$\frac{8-6}{6} = \frac{2}{6} = +33\%$$

La figura 4.6 mostra un esempio di conversione in base64: preso un testo con caratteri non US-ASCII lo si trasforma in binario (scrivendo i codici ASCII di ciascun carattere), quindi si spezza la sequenza di bit in gruppi da 6 bit, si rappresenta in decimale il valore di ciascun gruppo e lo si usa per trovare il caratter della codifica base64 corrispondente (tramite la tabella in Fig. 4.5). Si ha quindi la corrispondenza:

`C'è` (codifica ASCII su 8 bit) diventa `Qyfo` (codifica base64)

con un aumento del 33% della dimensione del testo (che passa da 3 a 4 byte).

La codifica *quoted-printable* è preferita per la trasmissione di testi che contengono grosse quantità di caratteri nel set US-ASCII perché essa codifica solo quei pochi byte non conformi, ovvero i caratteri con codice ASCII > 127 (ovvero con MSB=1). La codifica viene fatta per ogni singolo carattere non conforme, sostituendo il carattere col simbolo di uguale seguito

1. testo originale:	C	’	è	
2. codice ASCII binario:	01000011	00100111	11101000	
3. gruppi da 6 bit:	010000	110010	011111	101000
4. valori decimali:	16	50	31	40
5. codifica Base64:	Q	y	f	o

Figura 4.6: esempio di conversione da testo ASCII a Base64.

dalla rappresentazione esadecimale su due cifre del codice ASCII del carattere. Ad esempio, sapendo che il carattere “è” nell’alfabeto ISO-8859-1 ha codice ASCII 232 (ossia 11101000 in binario e quindi E8 in esadecimale), si ha la seguente codifica:

`C’è` (codifica ASCII su 8 bit) diventa `C’=E8` (codifica quoted-printable)

La codifica dipende dall’alfabeto e righe più lunghe di 76 caratteri vengono interrotte con un “=” come ultimo carattere della linea. Viene garantita la trasparenza del contenuto della riga al delimitatore di linea, permettendo di inserire anche il carattere “=” nel testo attraverso la sintassi “=3D”. Si ha un aumento del messaggio variabile e dipendente dal numero di caratteri codificati, che da 1 byte diventano 3 byte.

4.8.3 Uso di MIME negli header RFC-822

Poiché anche in alcuni campi degli header RFC-822 possono servire caratteri non US-ASCII (ad esempio per inserire il nome del mittente o l’argomento del messaggio), si può utilizzare MIME per codificare il valore assegnato all’header, tutto o in parte. Il valore sarà codificato con la seguente sintassi:

`=?` *charset* `?` *encoding* `?` *testo_codificato* `?=`

ove *charset* corrisponde all’alfabeto usato nella codifica (es. US-ASCII, ISO-8859-1) ed *encoding* può assumere i valori `B` o `Q`, ad indicare rispettivamente la codifica base64 o quoted-printable. Ad esempio, volendo inserire il nome `Günter` in base64 nel campo mittente si può scrivere:

```
From: =?ISO-8859-1?Q?G=FCnter?=
```

4.8.4 Alfabeti MIME

Specificati nel parametro `charset`, i più comuni sono:

- US-ASCII;
- ISO-8859-1 (Latin-1);
- ISO-8859-2 (Latin-2);
- ISO-8859-3 (Latin-3);
- ISO-8859-4 (Latin-4);

- ISO-8859-5 (Latin/Cyrillic);
- ISO-8859-6 (Latin/Arabic);
- ISO-8859-7 (Latin/Greek);
- ISO-8859-8 (Latin/Hebrew);
- ISO-8859-9 (Latin-15);
- ISO-8859-15 (Latin-9);

Si noti che ISO-8859-15 aggiunge a ISO-8859-1 il simbolo dell'Euro, caratteri per la traslitterazione di parole russe ed alcune legature francesi, a costo di perdere il simbolo di pipe ed alcuni simboli diacritici isolati (umluat, cedilla, ...).

4.8.5 Un esempio MIME

La figura 4.7 contiene un esempio di messaggio di posta elettronica in formato MIME.

Sono evidenziati in blu gli header MIME che sono presenti sia a livello di header RFC-822 (per dichiarare l'uso di MIME ed indicare che il messaggio è composto da varie parti) sia a livello delle singole parti (per dichiarare tipo e codifica del contenuto di una specifica parte).

Trattandosi di un messaggio multipart, assume particolare importanza le righe (evidenziate in rosso) che delimitano le varie parti, composte dalla stringa dichiarata con l'attributo `boundary` preceduta da due caratteri meno ed seguita anche dagli stessi due caratteri per indicare la fine del messaggio MIME.

Si noti che all'inizio del body è presente una frase (evidenziata in verde) che non verrà visualizzata da un interprete MIME (perché non inserita in alcuna parte MIME) ma sarà invece visualizzata come prima frase del testo da parte di un interprete non-MIME. E' quindi d'uso inserire una frase di avviso che si tratta di un messaggio MIME e non di un normale messaggio RFC-822.

```
From: Antonio Lioy <lioy@polito.it>
To: Antonio Lioy <lioy@polito.it>
MIME-Version: 1.0
Subject: test di MIME
Content-Type: multipart/mixed; boundary="0107040803040507"

This is a multi-part message in MIME format.

--0107040803040507
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: quoted-printable

Questa =E8 la parte di testo.
--0107040803040507
Content-Type: application/x-zip-compressed; name="tesi.zip"
Content-Transfer-Encoding: base64
Content-Disposition: inline; filename="tesi.zip"

dwIAAEQFAAAIAAAAdGVzaS50eHRtVMuO1DAQvEfK/as453ggfff
. . .
kudHhOUEsFBgAAAAABAAEANgAAAJOCAAAAAA==
--0107040803040507
Content-Type: application/vnd.ms-excel; name="voti.xls"
Content-Transfer-Encoding: base64
Content-Disposition: inline; filename="voti.xls"

OM8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/CFaggTTTTT
. . .
OTA1OTJjPlONPj4Nc3RhcnR4cmVmDTE3Mw0lJUVPRg0=
--0107040803040507--
```

Figura 4.7: esempio di messaggio di posta elettronica in formato MIME.

Capitolo 5

Architetture di sistemi distribuiti

5.1 Le applicazioni informatiche

In generale un'applicazione informatica è sempre costituita da tre parti fondamentali (Fig. 5.1):

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.

5.1.1 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 5.2).

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

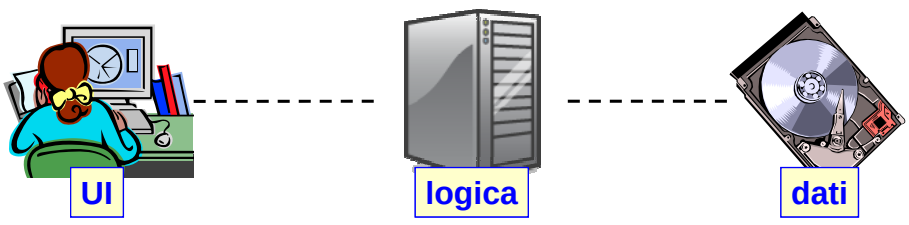

Figura 5.1: Modello di un'applicazione informatica.

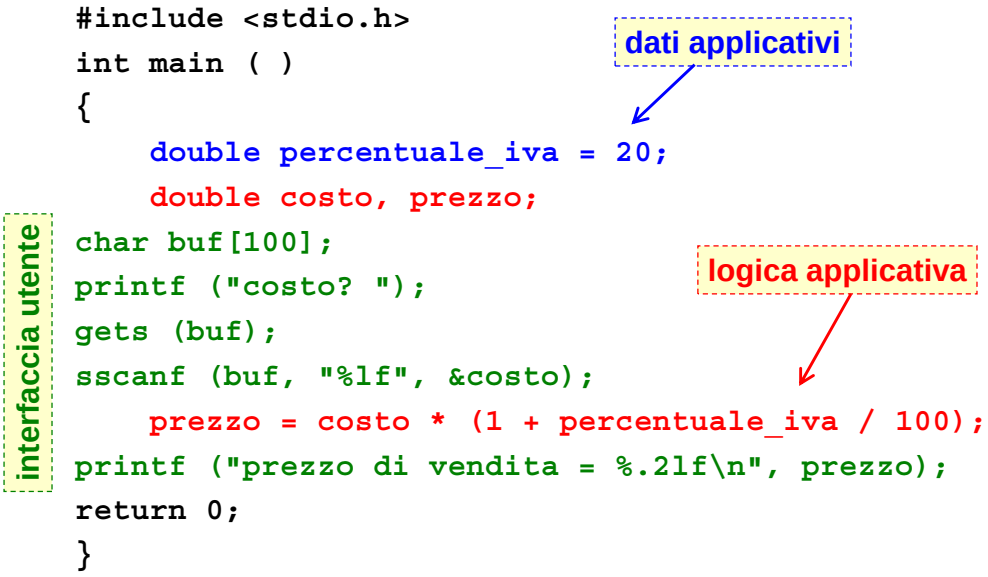

Figura 5.2: Esempio di un'applicazione classica

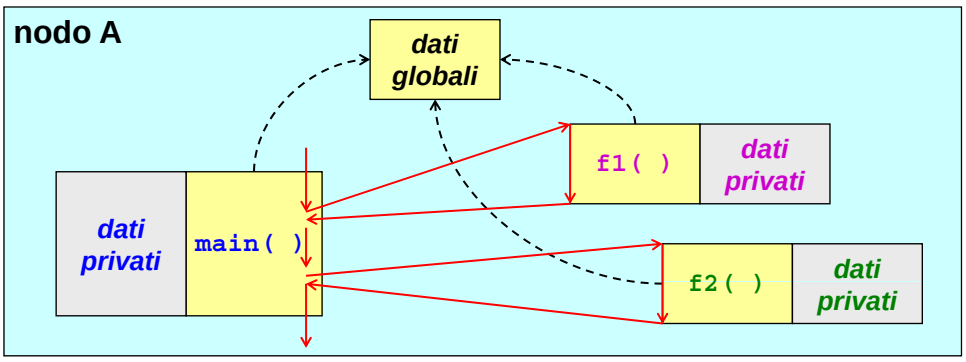

Figura 5.3: Flusso di elaborazione e localizzazione dei dati nel caso classico.

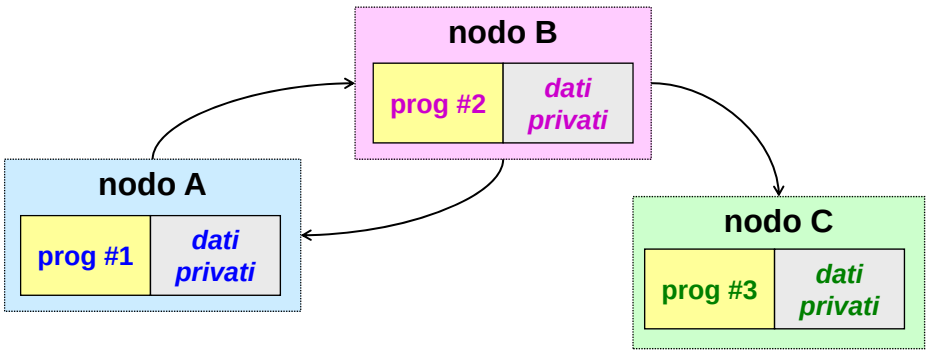

Figura 5.4: Flusso di elaborazione e localizzazione dei dati nel caso distribuito.

Come rappresentato in figura 5.3, 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.

5.1.2 Elaborazione distribuita

A differenza dell'elaborazione classica, in quella distribuita i dati sono solo privati e gli spazi di indirizzamento sono molteplici (figura 5.4). 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 5.5 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

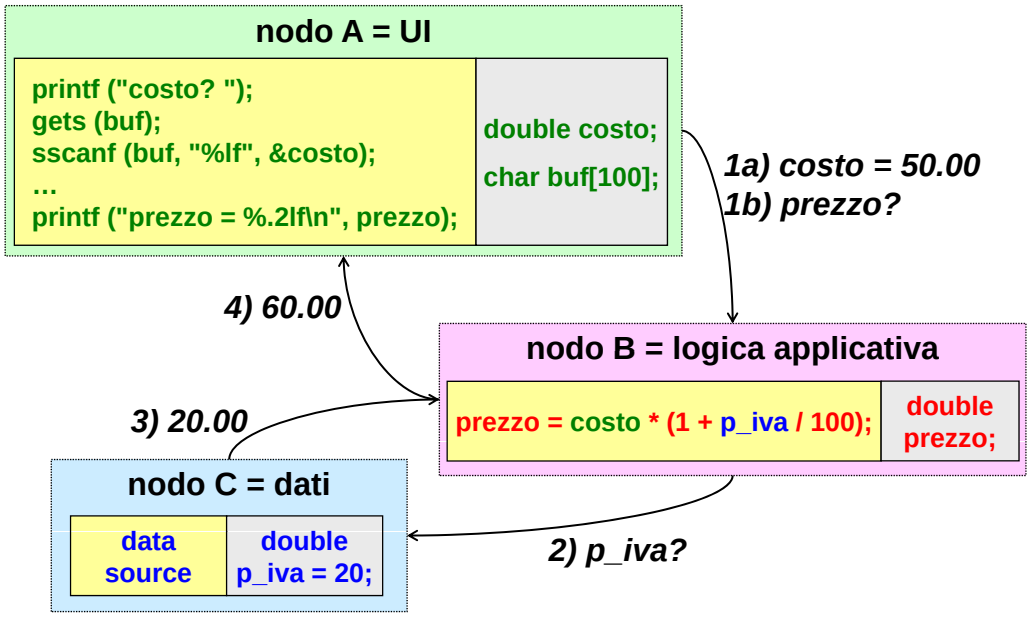

Figura 5.5: Esempio di un'applicazione distribuita

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).

5.2 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-

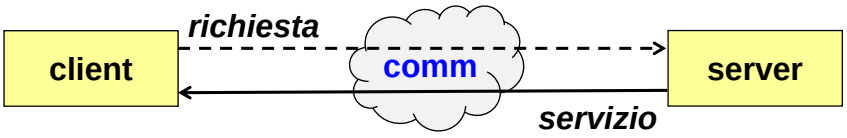

Figura 5.6: Architettura client-server.

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.

5.2.1 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.

5.2.2 Architettura client-server

In questo tipo di architettura, i client richiedono i servizi offerti dai server (fig. 5.6). 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 o livello e questa architettura è appunto a due livelli.

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

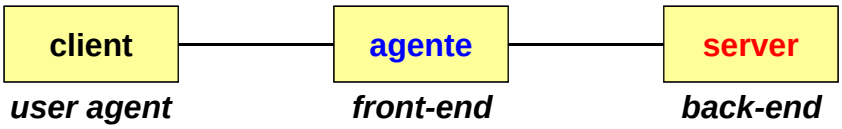

Figura 5.7: Architettura C/S 3-tier.

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.

5.2.3 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 5.7). 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.

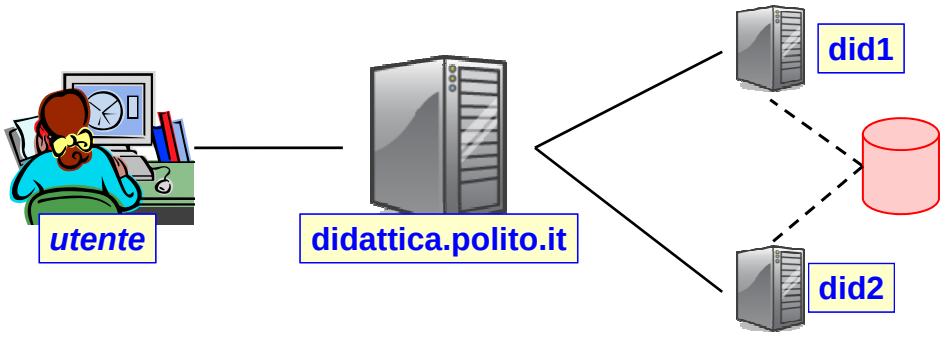

Figura 5.8: Esempio di architettura C/S 3-tier con bilanciatore di carico.

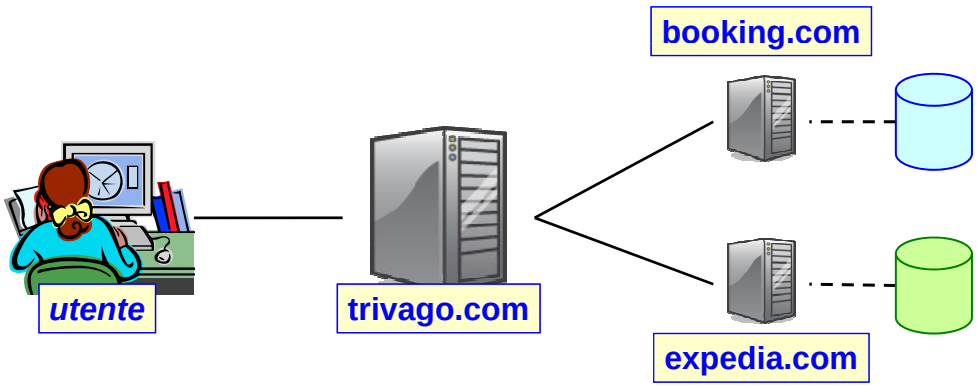

Figura 5.9: Esempio di architettura C/S 3-tier con broker.

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 5.8). 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 questo modo risulta improbabile che si verifichi simultaneamente lo stesso problema su entrambi i server, sia dal punto di vista hardware 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 5.9) 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 si agevola l'utente che non deve effettuare da solo 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`

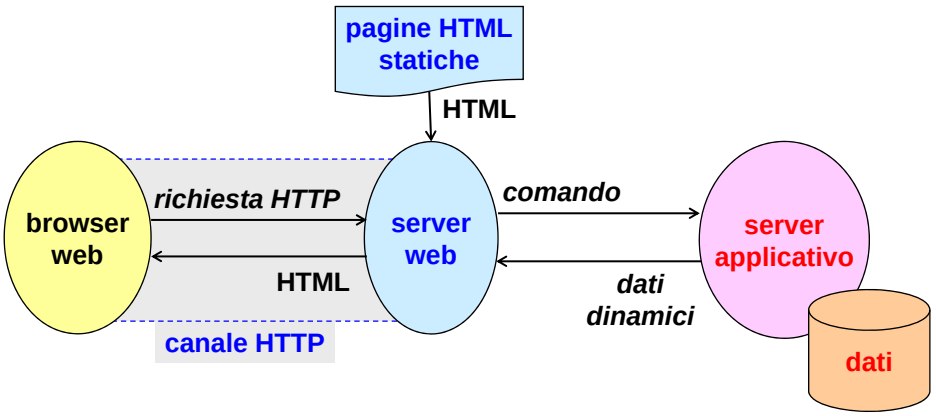

Figura 5.10: C/S 3-tier web-based – front-end leggero.

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 system), 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.

5.2.4 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 5.10.

Un secondo modello, rappresentato in figura 5.11, 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.

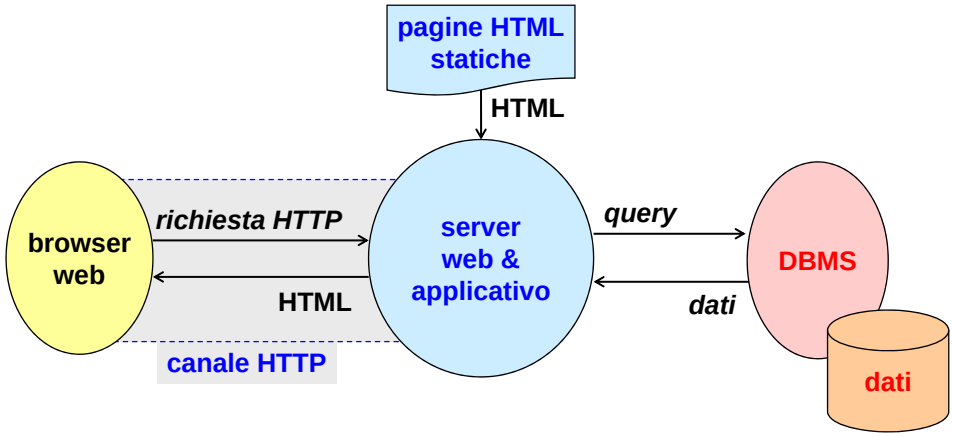

Figura 5.11: C/S 3-tier web-based – front-end pesante.

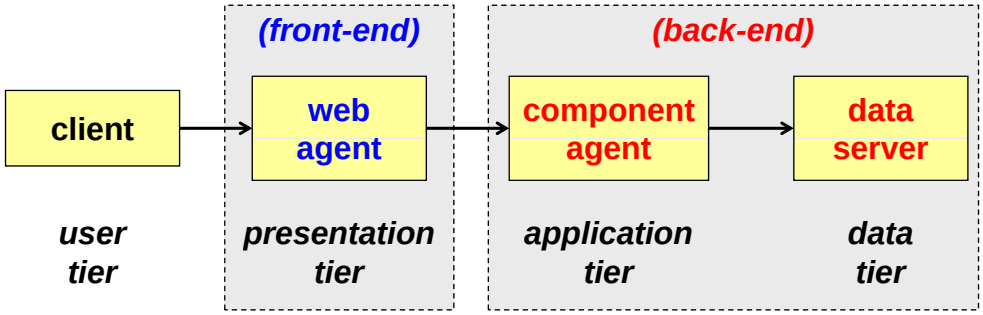

Figura 5.12: C/S 4-tier web-based.

5.3 Architettura C/S 4-tier

La soluzione migliore è quella con quattro livelli. Come si può vedere dalla figura 5.12, 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 Vodafone sono svantaggiati in quanto il traffico che arriva a Telecom da utenti di altri operatori è limitato da un filtro. Colui che fornisce il servizio di Internet si chiama *ISP (Internet Service Provider)* e tra ISP diversi si stabiliscono *accordi di peering* per specificare direzione e quantità di traffico che due ISP accettano di scambiarsi.

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 i 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.

5.4 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 (scritti 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.

5.5 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 edge 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 edge 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).

5.6 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.

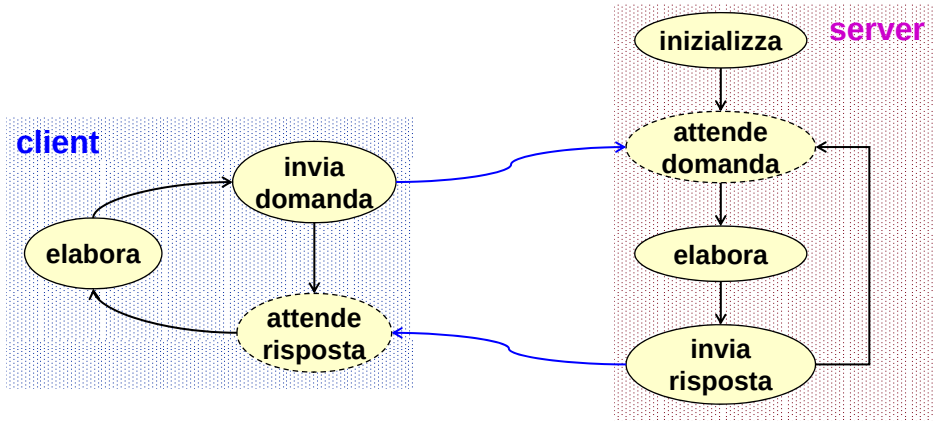

Figura 5.13: Server iterativo.

5.6.1 Server iterativo

Nel server iterativo (figura 5.13) 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 UTC (Universal Time Coordinated).

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 sistema 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 ricontattare il server più tardi) perché un server iterativo normalmente non gestisce una coda di richieste pendenti.

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 client che fanno richiesta simultaneamente. Il tempo di latenza medio sarà uguale al tempo di elaborazione moltiplicato per il valor medio di W:

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

5.6.2 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 hanno informazioni riguardo la frammentazione ed il throughput). Una lettura dal disco richiede almeno un accesso, ossia 0.01 s, e poiché si devono fare 10 letture 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 richiesta è trascurabile. Il tempo della rete per la risposta, indicato con T_r, è pari alla dimensione della risposta, 80 Mbit, diviso per 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:

$$T_e = T_c + T_d + T_r = 8.100\,05\,\text{s/servizio}$$

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 installare 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 serve quando il carico non è uniforme perché assorbe le differenze di carico quando ci sono dei picchi.

5.6.3 Server concorrente

Per gestire diversi clienti contemporaneamente si usa un server di tipo concorrente (figura 5.14). 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.

Un server concorrente è adatto per gestire servizi dei quali non è nota a priori la durata del lavoro di elaborazione. Esempi di server concorrente sono la maggior parte dei 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);

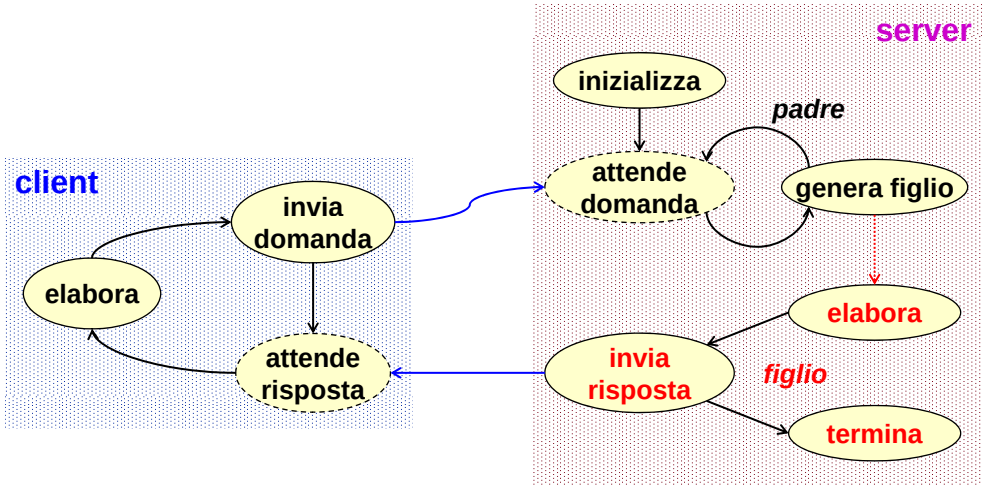

Figura 5.14: Server concorrente.

- *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 quando la durata di un servizio non è prevedibile a priori.

Il principale vantaggio di questo modello di server consiste nel carico teoricamente illimitato (si può aumentare il numero di figli in base alle richieste e non esiste un limite massimo) ma nei casi reali è bene tener conto che ogni figlio richiede una parte di risorse e che queste non sono infinite.

Questo vantaggio è da confrontarsi coi seguenti svantaggi:

- complessità di programmazione (scrivere programmi di questo tipo è complicato perché bisogna tenere conto della 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 date da:

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

Anche in questo caso possiamo stimare la latenza del servizio ed essa 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 (workload) è il numero di client simultanei e C il numero di CPU.

5.6.4 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 istruzioni/s), 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 si moltiplica il numero delle letture per il tempo necessario ad eseguirne una lettura:

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

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 cache/-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 5.15 Quando si ricevono due richieste da due client diversi, le due richieste

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

Figura 5.15: Esempio 4 Settembre 2009.

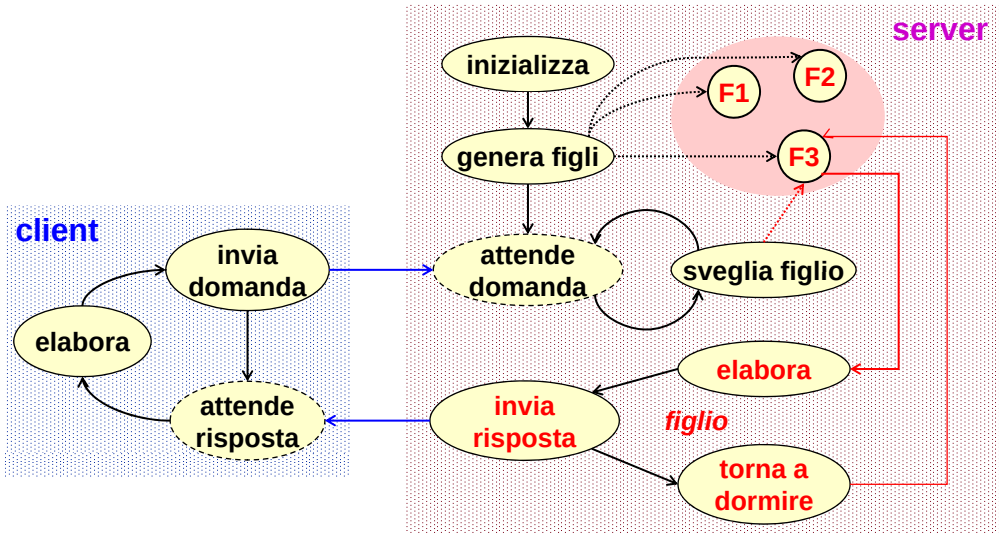

Figura 5.16: Server a crew.

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.

5.6.5 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 5.16) 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 alte prestazioni (server sottoposti ad alto carico oppure server in cui il ritardo di risposta non è accettabile e si vuole ridurre la latenza);
- esempi: servizi Web per e-commerce, server di database o comunque casi in cui la velocità di 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 semplicemente 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:

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

5.7 Programmazione concorrente

Nell'implementazione di server concorrenti o a crew possiamo avere due tipi di modelli di programmazione: a processi o thread (figura 5.17). 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. Ad esempio se si hanno due processi avrò due PC, due codici e due zone di memorie dedicate a ciascun 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 multi-core ogni thread viene eseguito su un 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 5.17: 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 debugger, perché il debug altera 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 (ad esempio in un videogioco) mentre preferiamo un modello a processi quando è importante la solidità del sistema (ad esempio, un sistema di controllo critico come quello del traffico ferroviario).

Capitolo 6

Programmazione in ambiente web

6.1 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 6.1, 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.

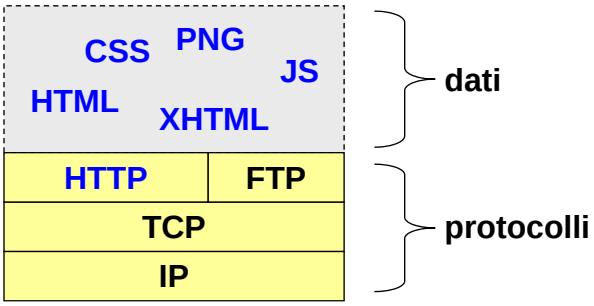

Figura 6.1: Struttura del web.

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.

6.2 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.

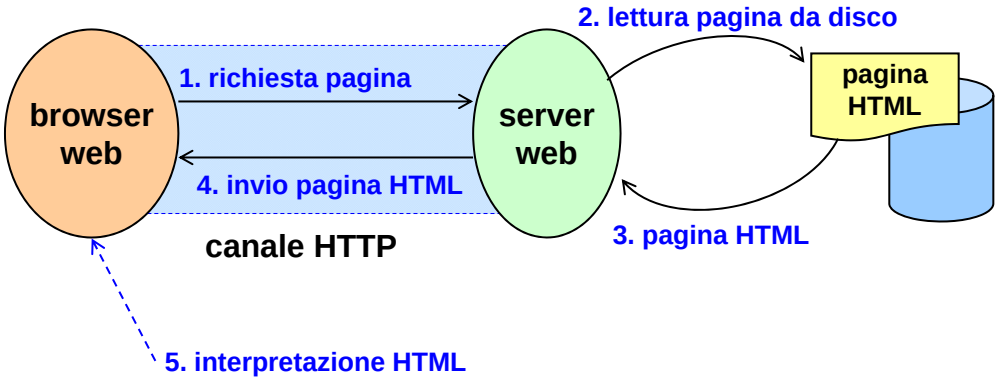

Figura 6.2: Modello di architettura web statica.

La procedura di richiesta e ottenimento della risorsa è quella di cui abbiamo un esempio nella figura 6.2. 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.

6.2.1 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.

6.2.2 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/01nbe/
```

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

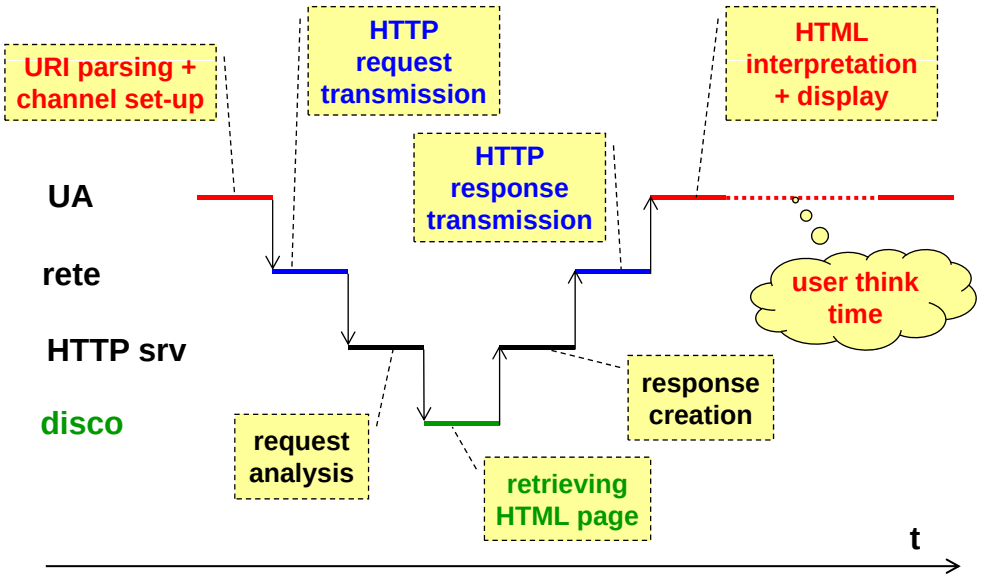

Figura 6.3: diagramma temporale di una transazione nel web statico.

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/01nbe/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].

6.2.3 Modello delle prestazioni nel web statico

Nella figura 6.3 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 6.3, 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.

6.3 User agent, origin server, proxy e gateway

Come già discusso, lo *User Agent (UA)* è frequentamente il browser dell'utente, ma questa funzione può anche essere svolta da altri programmi che si comportano da client, quali *spider* e *robot* (entrambi programmi che visitano automaticamente le pagine web per raccogliere informazioni, spesso al fine di creare indici dei contenuti).

L'*Origin Server (OS)*[3] 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.

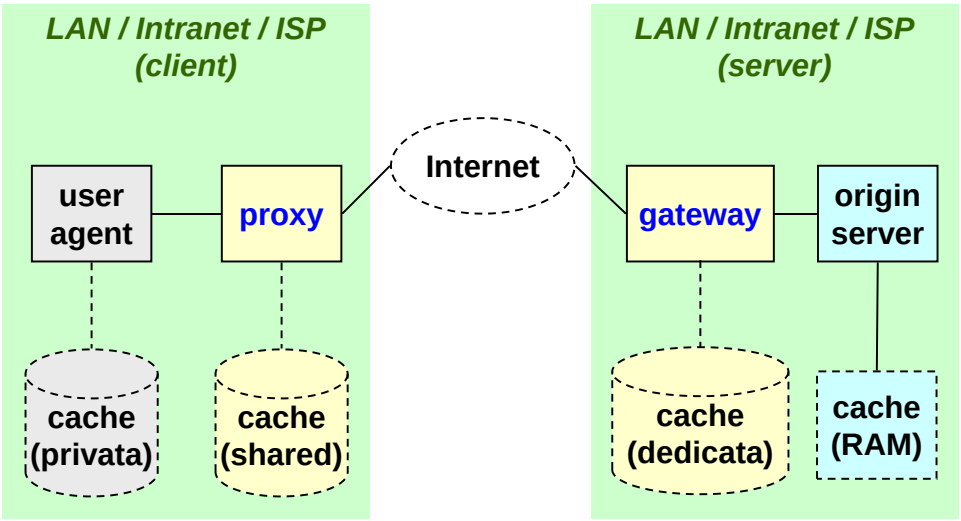

Figura 6.4: schema di un canale HTTP con proxy e gateway.

Come illustrato nella figura 6.4, tra UA ed OS possono esistere altri elementi che spezzano la comunicazione HTTP.

Il *gateway* (posto lato server) funziona come un'interfaccia pubblica per il server e viene spesso usato per funzioni sicurezza (il cosiddetto "reverse proxy") o per fare load balancing (nel caso che alle sua spalle siano presenti vari server equivalenti).

Il *proxy* (posto lato client) 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. Viene spesso usato anche con funzione di autenticazione e autorizzazione, per controllare quali utenti hanno diritto ad accedere alla rete esterna e quali tipi di server e/o contenuti possano accedere.

Gateway e proxy sono elementi bifronti, ossia da un lato si comportano come un client e dall'altro come un server secondo la necessità. 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. 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 noto 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. La cache del gateway è marcata come "dedicata" perché memorizza solo i contenuti del server a cui fa da interfaccia

[3]Non si confonda questa abbreviazione con quella normalmente usata in inglese per indicare il sistema operativo (OS = Operating System).

6.3.1 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, ad esempio la rete del Politecnico di Torino, quella italiana e quella europea.

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.

6.3.2 Configurazione del proxy sugli user agent

PARTE NON TRASCRITTA

6.4 Web statico con pagine dinamiche

Il termine web statico con pagine dinamiche indica 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. Lo schema di questa architettura è mostrato in figura 6.5.

6.4.1 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);

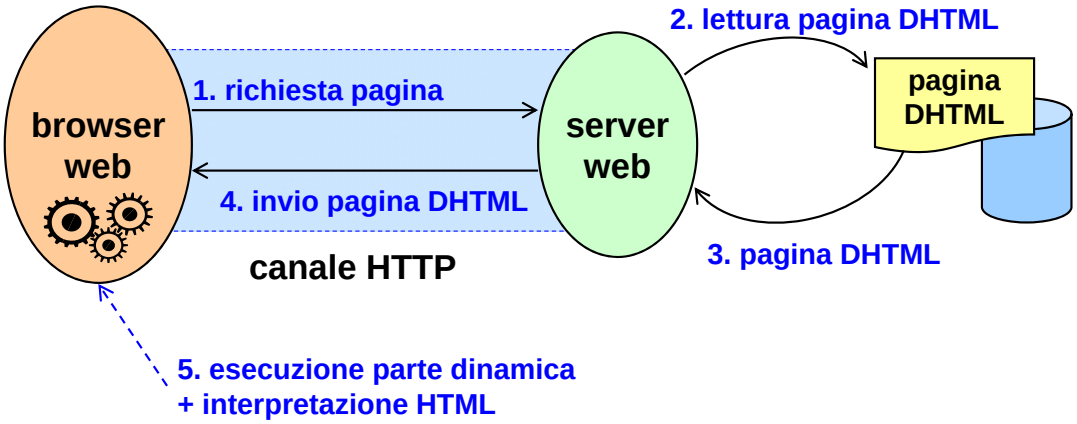

Figura 6.5: Modello di architettura web statica con pagine dinamiche.

- 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. Tutte queste considerazioni devo essere tenute presenti nella realizzazione di questo tipo di pagine.

6.4.2 Metodi d'implementazione

Un modo per rendere possibile la realizzazione di pagine dinamiche è l'utilizzo di applet Java o componenti ActiveX. Entrambi sono programmi che possono essere eseguiti dai web browser; i primi richiedono una Java Virtual Machine (JVM) mentre i secondi un sistema "Wintel", ossia uno UA eseguito su un nodo con sistema operativo Windows ed 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.

6.4.3 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 web 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.

6.4.4 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 6.6 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.

```
<!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 6.6: Un semplice esempio di script lato client.

La figura 6.7 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 10 e **??**.

6.5 DOM event handler

Il linguaggio JavaScript è stato sviluppato per essere inserito in altri sistemi, a differenza del linguaggio C e altri che permettono di sviluppare un'applicazione vera e propria ovvero

```
<!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 6.7: Script lato client usato per generare la tavola dei quadrati.

un programma a se stante. In particolare JavaScript è stato pensato per essere inserito all'interno del browser potendo così interagire con vari aspetti del funzionamento dello stesso. Questa interazione avviene tramite un modello chiamato DOM (Document Object Model). In questa lista di oggetti sono disponibili anche degli eventi che "capitano" sugli oggetti contenuti nella pagina. E' possibile associare comandi JavaScript ad eventi, tramite un"gestore di eventi" ("event handler"). Utilizzando questa sintassi:

<*tag . . . eventHandler* =" *codice_javascript* ">

è possibile aggiungere su un tag come ulteriore attributo il nome di un gestore di eventi e inserire come valore tra i doppi apici:

- direttamente il codice JavaScript, se la gestione dell'evento richiede solo poche istruzioni;
- richiamare una funzione JavaScript, se la gestione dell'evento viene demandata ad una specifica funzione.

6.5.1 Alcune tipologie di eventi

- `onclick` indica il click sul testo o immagine associata al tag (quando si fa click con il mouse su di un determinato elemento viene eseguito quel pezzo di JS);
- `ondblclick` identico al precedente ma con un doppio click;
- `onfocus` campo attivo per l'input (quando ci sono dei campi in cui è possibile inserire del testo posizionandoci sopra il mouse);
- `onblur` campo non più attivo per l'input (inverso di `onfocus`);
- `onsubmit` invio dei dati di un form tramite pressione del tasto Submit;

```
...
<head>
<title>Esempio JS associato ad onclick</title>
  <script type="text/javascript">
  function makeRed(x)
  {
    obj = document.getElementById(x);
    obj.style.color = "red";
  }
 </script>
</head>
<body>
  <p id="id1" onclick="makeRed('id1')">
  Click on this text to make it red!
  </p>
</body>
```

Figura 6.8: Esempio di JS associato all'evento `onclick`.

- `onreset` si attiva quando vengono modificati i dati di un form tramite pressione del tasto Reset;
- `onchange` indica campo modificato e non più attivo per l'input, ossia `onblur` più cambiamento del contenuto;
- `onload` corrisponde al caricamento della pagina (ad esempio, alcuni siti web quando vengono visitati presentano degli avvisi tramite questo evento);
- `onunload` corrisponde a lasciare la pagina (per vederne un'altra o chiudere il browser).

Questi sono solo alcuni degli eventi, ovvero quelli più frequenti. Per una lista completa di tutti gli eventi previsti da DOM si rimanda alla sezione 18.2.3 di HTML 4.01.

6.5.2 Esempi DOM con JavaScript

Un primo esempio di JS associato alla gestione di un evento è riportato in figura 6.8. All'interno dell'intestazione HTML è stato creato uno script contenente una funzione, quindi non del codice che deve essere eseguito immediatamente ma una funzione chiamata `makeRed` che rende rosso l'elemento indicato come parametro. All'interno delle parentesi graffe ci sono le istruzioni (ecco l'aggancio con il DOM):

`obj=document.getElementById(x)` e `obj.style.color="red"`

ove `document` è l'oggetto DOM che rappresenta tutta la pagina mentre `getElementById(x)` è un metodo che ricerca all'interno del documento quello che ha un certo identificativo (ricordiamo che `id` è un attributo che può essere messo su qualunque tag ed è un identificatore univoco). Il metodo restituisce l'unico elemento che ha quell'identificativo (indicato in parentesi, nell'esempio è `x`). Può anche non restituire nulla nel caso non ci sia quell'identificativo, in tal caso restituisce `null`. Con la notazione `obj.style.color="red"` si assegna il colore rosso allo stile dell'oggetto `obj`.

file `js3.html`

```
...
<head>
  <script src="js3.js"
   type="text/javascript">
  </script>
</head>
<body>
  <p id="id1"
   onclick="makeRed('id1')">
  Click on this text to make it
   red!
  </p>
</body>
```

file `js3.js`

```
function makeRed(x) {
  obj =
   document.getElementById(x);
  obj.style.color="red";
}
```

Figura 6.9: Esempio di codice JS definito in un file esterno.

Una volta creato lo script, essendo questa una funzione, non viene eseguita ma viene definita. Pertanto in JavaScript quando si vogliono creare delle funzioni lo si fa tipicamente nella parte di "head" di modo che le funzioni siano sempre disponibili anche ad altri. A questo punto nel body aggiungiamo un paragrafo e, essendo questo un tag, gli viene assegnato un id e poi aggiungendo l'evento "onclick" facciamo sì che quando un utente clicca sul paragrafo lo renda di colore rosso. Notare bene che dovendoci essere dopo `onclick` i doppi apici per definire l'istruzione JS e dovendo ancora definire il parametro che è una stringa, non si possono riutilizzare i doppi apici pertanto in JS è possibile definire le stringhe con entrambi i tipi di apici.

Nel caso in cui una funzione ci serva in più pagine conviene definirla in un file di testo separato e poi richiamarlo (esempio in figura 6.9). Per convenzione vengono salvate con estensione `.js`, vanno create con un normale file di testo (es. notepad) e devono avere un nome con lunghezza massima otto caratteri per questione di portabilità. Trattandosi solo di codice JavaScript non deve contenere il tag `<script>` (che serve solo in HTML). Occorre creare un file JavaScript che contiene esclusivamente la definizione della funzione (la stessa che nell'esempio precedente compare nel tag). Per poterla usare in un altro file nella parte di intestazione occorre scrivere:

`<script src="` *nome_file_JS* `" type="text/javascript">`

L'attributo `src` indica il nome (o meglio l'URI) della risorsa contenente il codice JS che si vuole richiamare e può essere direttamente un nome di file (se è contenuto nella stessa cartella) oppure può essere un pathname (relativo alla cartella corrente o assoluto se riferito alla radice del server web) o una risorsa che si trova su un altro server. Ovviamente bisogna sempre definire con l'attributo `type` il tipo di linguaggio di scripting usato nel file indicato dall'attributo `src`.

6.6 Web dinamico

PARTE NON TRASCRITTA.

Capitolo 7

Il linguaggio HTML

7.1 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.

7.2 Caratteristiche di un documento HTML

Un documento di tipo HTML si può realizzare con un qualsiasi editor in grado di creare un 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 è un formato binario. 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.

7.2.1 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 esesre 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.

7.2.2 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.

7.2.3 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 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] In inglese "line break", da cui il nome

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.

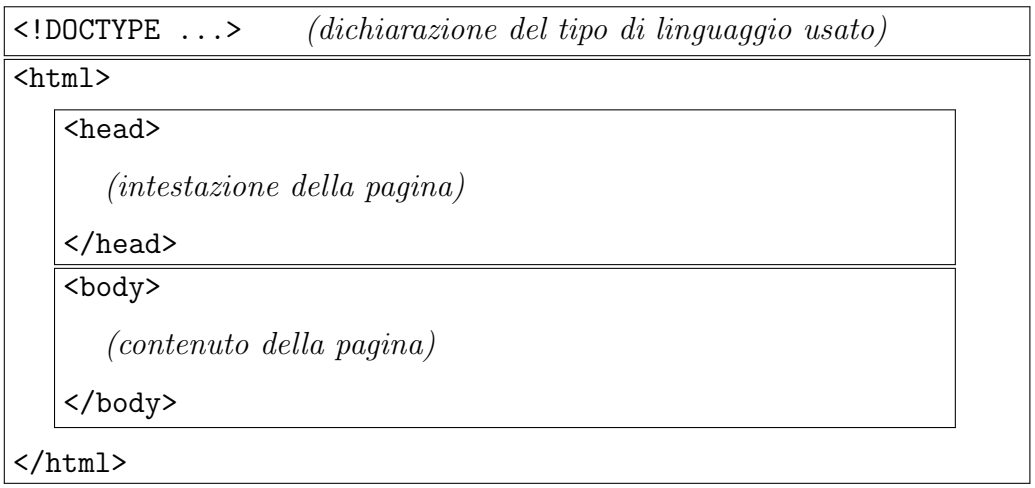

Figura 7.1: Struttura di una pagina HTML.

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.

7.3 Struttura generale di un documento HTML

Come illustrato nella figura 7.1 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 7.2.

```
<!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 7.2: Un semplice esempio HTML.

7.3.1 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 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

7.3.2 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.

7.3.3 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 usarein base alle informazioni fornite da 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
  ```

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

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

- l'attributo charset di un tag 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;

```
<!-- 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 7.3: esempio di uso del tag `<link>`.

  - `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 7.3 contiene un esempio di uso di vari tag link.

7.3.4 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:

```
<!-- 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

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

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.

Sezioni e titoli

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 dal 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.

7.3.5 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;
- `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 ...`).

codice HTML

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

risultato (nel browser)

Per superare l'esame 01NBE:

I. frequentare le lezioni

II. svolgere le esercitazioni di laboratorio

Figura 7.4: Esempio di lista ordinata con numeri romani.

La figura 7.4 presenta un esempio di lista ordinata.

Una lista di definizioni 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.

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).

7.4 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

7.5 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.

7.5.1 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.

7.5.2 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>`

7.5.3 Altri stili logici

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

Questi due stili logici fanno rispettivamente aumentare o di diminuire di un gradino nella scala delle grandezze il testo racchiuso e si possono usare ripetutamente in modo annidato per ottenere un effetto maggiore:

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

7.5.4 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)

7.6 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 €)

7.7 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>`.

7.7.1 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>`

```
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).

7.8 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

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

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

Figura 7.5: esempi di link relativi.

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 7.5 mostra vari esempi di link relativi.

7.9 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/01nbe/#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.

7.10 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. Nel seguente esempio viene inserita un'immagine contenuta nel file `polito.gif`:

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

Per le persone non vedenti le immagini sono inutili mentre per chi si collega tramite una linea lenta caricare immagine grossa può essere fastidioso. Entrambe queste categorie di utenti potrebbero decidere di non voler caricare le immagini (impostazione possibile su tutti i browser) È quindi importante, ed in certi casi obbligatorio (es. pubblica amministrazione), che il sito sia fruibile anche in assenza di immagini. Per ottenere questo risultato si può inserire l'attributo `alt` per fornire un testo alternativo alla visualizzazione dell'immagine, come nel seguente esempio:

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

Così, se il visitatore ha disabilitato il caricamento delle immagini, al posto di questa immagine verrà inserito il testo "Foto del Politecnico".

Si presti attenzione alla differenza tra inserire nella pagina direttamente l'immagine o un 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.

7.10.1 Posizionamento reciproco di 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 . . . >
```

7.10.2 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 (height), le quali devono essere specificate in pixel.

Questo permette di ridimensionare (resize) l'immagine, ossia scalarla; però se si sbaglia, si rischia di deformare l'immagine. Per esempio, se avessimo un'immagine rettagolare e facessimo width=10 e height=10 diventerebbe quadrata. Per non sbagliare è sempre meglio specificare solo una delle due dimensioni e l'altra verrà scalata in proporzione automaticamente.

```
<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.

7.11 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:

- `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

7.11.1 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>
```

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 7.6: colori standard HTML.

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 7.6 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.

7.12 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:

- `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.

[3] In inglese "table row".

- `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%">
```

7.12.1 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*

7.12.2 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.

7.12.3 Table: attributi di riga,header e dati

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

7.12.4 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.

7.13 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.

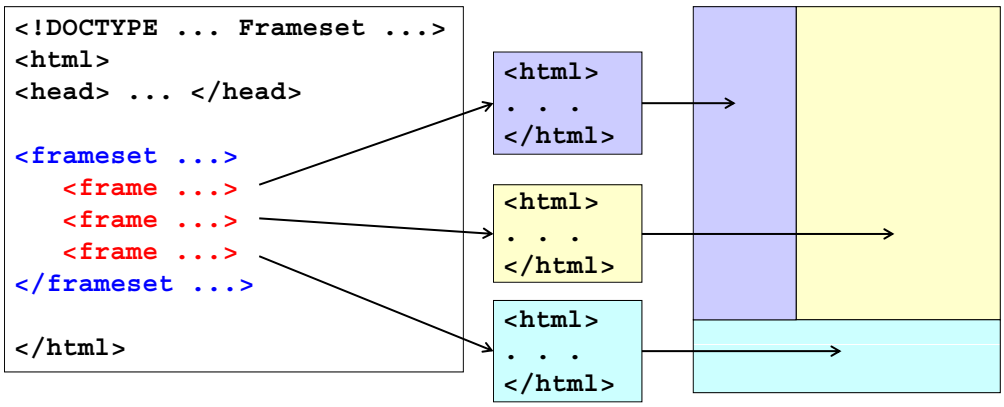

Figura 7.7: Esempio di pagina strutturata in frame.

7.13.1 Frameset e frame

La struttura della pagina organizzata in frame, come illustrato nella figura 7.7, 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.

7.13.2 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.

7.13.3 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.

7.13.4 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 7.7.

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`".

7.14 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.

7.15 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.

7.16 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`, ...

7.17 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 8

Il linguaggio CSS

NOTA: capitolo largamente incompleto.

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 introdotti i tag dedicati alla formattazione, come il tag `<font>`. ma il risultato fu negativo: un sito web è sviluppato e aggiornato da più persone, ognuna delle quali segue le proprie preferenze di stile generando così una disuniformità dell'aspetto grafico del sito. Per far fronte a questa problematica si è tornati alle origini, eliminando i tag di formattazione e delegando tale compito ad una specifica separata: i fogli di stile o CSS (Cascading Style Sheet). In HTML-4 è ancora possibile usare sia il CSS, sia i tag per la formattazione visuale, tuttavia quest'ultimi sono deprecati e non esistono più in HTML-5 e XHTML.

8.2 Introduzione ed evoluzione del CSS

La prima versione di CSS è del 1996. Nel giugno 2011 ne è stata rilasciata la versione 2.1, che aggiunge oltre 70 funzionalità alle 50 iniziali. Attualmente è in corso di sviluppo la versione 3, che non viene trattata in questo testo perché è ancora largamente incompleta. Si noti che tra le varie versioni di CSS esiste la compatibilità all'indietro: salvo rare eccezioni, una specifuca in formato CSS-1 è valida anche in formato CSS-2.

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.

8.3 Formato ed integrazione con HTML

Il codice che serve a definire il formato e il layout della pagina, deve essere scritto come puro testo ASCII e solitamente viene salvato in un file esterno alla pagina, con estensione `.css` (non obbligatoria ma caldamente consigliata).

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

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

Usando il parametro `media` si può richiedere che certe regole vengano applicate solo se il file è visualizzato da uno specifico tipo di UA, come nel seguente esempio che carica delle regole valide per qualunque UA (`polito_base.css`) ed opzionalmente altre regole aggiuntive nel caso che pagina HTML venga stampata (`polito_stampa_A4.css`):

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

8.4 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.

8.4.1 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;
```

8.4.2 Commenti

In CSS i commenti seguono la sintassi del linguaggio C, ossia iniziano con `/*` e terminano con `*/`, possono occupare più righe ed essere inseriti in qualunque punto di una specifica.

8.4.3 Selettori

HTML organizza il documento gerarchicamente: gli elementi contenuti nel `body` sono detti figli (e tra loro fratelli) (fig. 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 la proprietà `background` (colore dello sfondo) che non viene ereditata ma, poiché di default ha il valore `transparent`, il colore di sfondo impostato sul genitore risulta visibile anche su tutti i discendenti che non impostano esplicitamente un colore diverso.

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

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

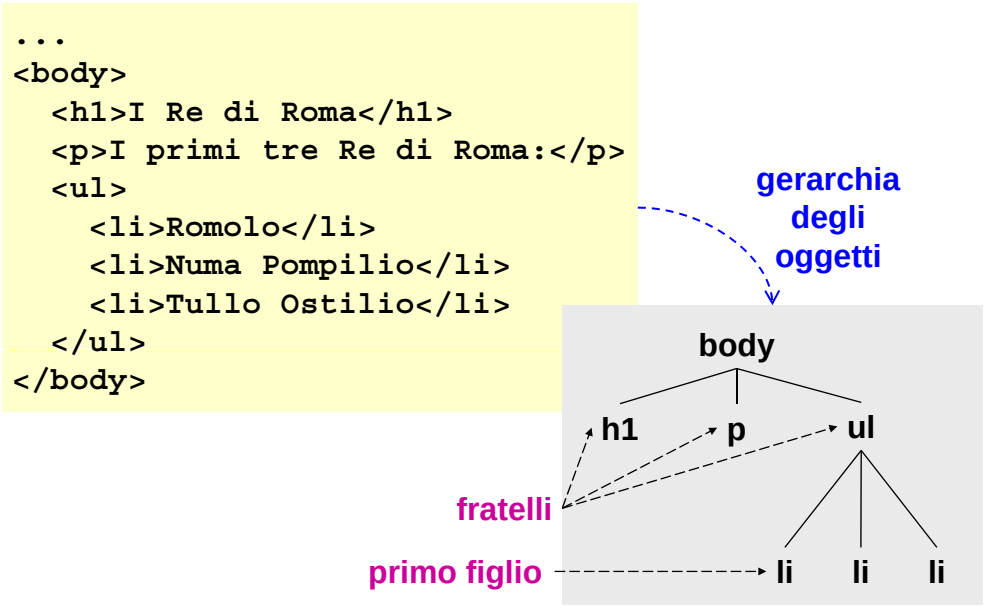

Figura 8.1: Esempio di gerarchia degli oggetti CSS.

selettore	*esempio*	*descrizione dell'esempio*
`:link`	`a:link`	i link mai visitati
`:visited`	`a:visited`	i link già visitati
`:active`	`a:active`	il link attivo
`:hover`	`a:hover`	il link su cui vi è posto il cursore
`:focus`	`input:focus`	il controllo di input (di un form) che ha il focus
`:first-letter`	`p:first-letter`	il primo carattere di ogni paragrafo
`:first-line`	`p:first-line`	la prima riga di ogni paragrafo
`:first-child`	`p:first-child`	ogni paragrafo che è il primo figlio del suo genitore
`:lang(...)`	`p:lang(it)`	ogni paragrafo con attributo `lang` uguale a `it`

Figura 8.2: pseudo-classi CSS e relativi esempi.

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

```
.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 `aqua`:

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

La tabella in figura 8.2 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 ...

8.4.4 Dimensioni

Nel CSS per indicare dimensioni di oggetti si possono usare unità relative o assolute (fig. 8.3): le prime sono consigliate per uno schermo (essendo ignota la sua dimensione fisica reale), mentre le seconde si preferiscono per la stampa (ipotizzando nota la dimensione del foglio su cui si desidera stampare).

Unità relative:		
`em`	=	altezza della lettera "M" nel font corrente
`ex`	=	altezza della lettera "x" nel font corrente
`px`	=	dimensione di un pixel
Unità assolute:		
`in`	=	inch (25.4 mm)
`cm`		centimetri
`mm`		millimetri
`pt`	=	punto tipografico (1/72 di inch)
`pc`	=	pica (12 pt)

Figura 8.3: unità di misura CSS.

8.4.5 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.

8.4.6 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/

8.4.7 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
```

8.4.8 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.

8.4.9 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 usare 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 sono `Times`, `Times New Roman`, `Palatino`, `Georgia` (progettato per il web) e `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` e `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
```

8.4.10 Contenitori

Ogni blocco viene visto come un contenitore secondo il modello illustrato in figura 8.4

8.4.11 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.

- `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`;

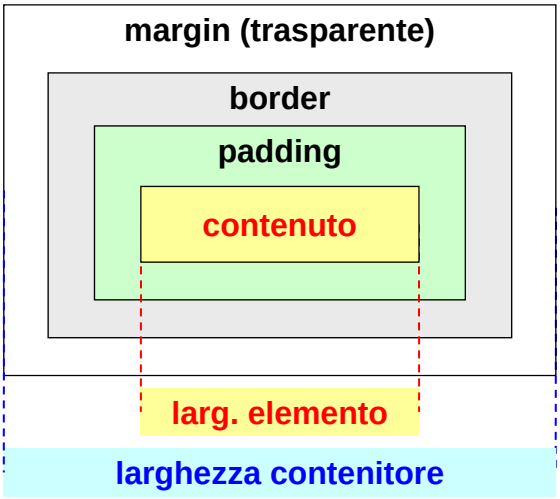

Figura 8.4: Modello dei contenitori in CSS.

- `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.

8.4.12 Bordi

Esistono quattro proprietà per specificare le caratteristiche del bordo in alto, a destra, in basso e a sinistra:

```
border-top-style border-right-style border-bottom-style border-left-style
```

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.

8.4.13 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`.

un solo parametro equivale a	...	`top_bottom_left_right`
due parametri equivalgono a	...	`top_bottom left_right`
tre parametri equivalgono a	...	`top right_left bottom`
quattro parametri equivalgono a	...	`top right bottom left`

Ecco alcuni esempi:

- `border-style: dashed`
  realizza un bordo tratteggiato su tutti e quattro i lati;
- `border-width: thin thick`
  realizza un bordo sottile sopra e sotto, spesso a destra e a sinistra;
- `border-color: red black blue transparent transparent`
  realizza un bordo rosso sopra e blu sotto, nero a destra e trasparente a sinistra.

Esiste un ulteriore metodo per specificare le caratteristiche dei bordi in maniera sintetica, indicando il bordo (uno solo o tutti insieme) e le sue proprietà di spessore, stile e colore:

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

Ad esempio possiamo richiedere un bordo destro rosso, sottile e punteggiato scrivendo:

```
border-right: thin dotted red
```

oppure chiedere che tutti i bordi siano spessi, neri e tratteggaiti (si noti come non importi l'ordine dei parametri):

```
border: black thick dashed
```

8.4.14 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;

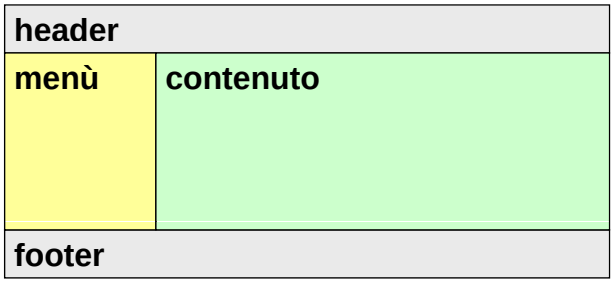

Figura 8.5: Esempio di layout grafico.

- `: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}
```

8.4.15 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 8.5 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 dal 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, sottolineato...).

E' possibile suddividere i tipi di carattere in due categorie principali: i font *sans-serif* (privi di grazia) ed i font *serif* (con grazia); i caratteri graziati hanno alle estremità degli allungamenti ortogonali, detti per l'appunto grazie.

Per quanto riguarda la scelta della famiglia da utilizzare, dobbiamo tenere in considerazione che, in generale, i font 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) per testo da visualizzare nel browser. Alcuni esempi di font sans-serif sono: Arial, Geneva, Helvetica, Lucida Sans Trebuchet e Verdana (font progettato appositamente per il web).

I font con grazia (serif), invece, non vengono resi bene dagli schermi ma sono perfetti se la pagina deve essere stampata: risultano più rilassanti per gli occhi e sono resi bene, grazie alla migliore qualità di stampa, anche da stampanti di bassa categoria (che solitamente hanno almeno una densità di 300 DPI). Alcuni esempi di font serif sono: Garamond, Georgia (progettato per il web), New York, Palatino, Times, Times New Roman.

Inoltre esistono font minori come quelli della famiglia *monospace* che risultano molto utili per scrivere codice o input da tastiera (Courier, Monaco), quelli della famiglia *fantasy* che si prestano bene per accentuare l'enfasi in un testo (Desdemona, Impact) e quelli *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 i caratteri accentati.

Infine, indifferentemente dalla scelta della famiglia, si sconsiglia di richiedere un font "raro" perché aumenta la possibilità che esso non sia disponibile per l'UA che dovrà visualizzare la pagina e diminuisce la portabilità del testo.

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 1920×1200 px avrà una densità di soli 147 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 DPI.

9.3 Leggibilità del testo

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

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 trovando una giusta combinazione tra sfondo e testo (evitare, ad esempio, un testo rosso o giallo su sfondo arancione, in quanto risulterebbe poco leggibile).

Dimensioni. Si misurano in pt (punti in verticale, ossia 1/72 di inch). Bisogna considerare che, a parità di dimensione in pt, font diversi utilizzano spazi verticali diversi (ad esempio, Verdana occupa il 58%, Times New Roman il 46%, mentre Flemish Script solo il 28%). Font che occupano maggiormente lo spazio verticale sono più facilmente leggibili. Per ovviare a questo problema esiste la proprietà `font-size-adjust` del CSS. Per la stampa si sconsiglia di scendere sotto i 10pt di dimensione del carattere (11-12pt è un'ottima scelta). Per la visualizzazione a video è consigliato non impostare una dimensione statica del font (in pt) ma, piuttosto, una dimensione percentuale (es. 80%, 100% o 110%) che si adatti alle preferenze scelte dall'utente nelle impostazioni del proprio browser.

Spaziatura tra righe dello stesso paragrafo. Andrebbe lasciato da un minimo del 10% ad un massimo del 30% per ottenere una migliore leggibilità. Il valore tipico è 20%.

Spaziatura tra le lettere. Una buona spaziatura (o l'utilizzo della famiglia monospace) aumentano la leggibilità del testo.

Font e stile. E' sconsigliato l'utilizzo di font condensed, che hanno le lettere vicine per risparmiare spazio orizzontale, 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'uso di carta lucida in quanto il riflesso disturba le 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 termine *web design* è usato 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 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 poiché, 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 eliminato.

9.4.1 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 deve 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 perché molti utenti utilizzano stampanti in bianco e nero.

- Modificare il font.

  La maggior parte delle pagine Web sono scritte con font sans-serif, perché 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.

9.4.2 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`

```
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.

```
<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 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 Gestione dei colori

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

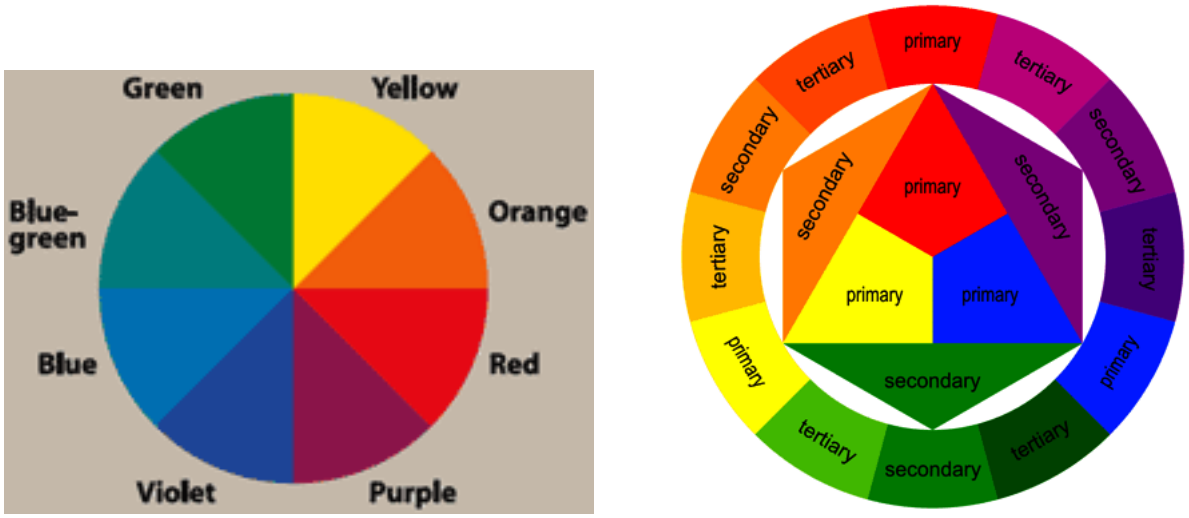

Figura 9.2: Il cerchio dei colori in base al parametro Hue.

9.5.1 Coordinate di colore

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 ed 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 identificati i colori primari, secondari (combinazione di due primari) e terziari (combinazione di un primario ed un secondario).

Il parametro *Saturation*, come mostrato in figura 9.3, parte dall'asse centrale ed aumenta andando in fuori. Più c'è saturazione e più il colore risulta intenso.

Infine il parametro *Lightness*, che varia lungo l'asse verticale, definisce la luminosità del colore, ovvero la percentuale di luce. Tutti i colori tendono ad assomigliare al nero quando la luminosità è bassa (0%), al contrario, quando è altissima (100%), tutti i colori tendono ad assomigliare al bianco. Il colore puro si ottiene a metà della luminosità (50%).

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;
- usare, per quanto possibile, colori standard (per nome o valore RGB) in modo che siano disponibili nell'UA; un testo rosso su uno sfondo blu, per esempio, può essere ben leggibile da uno UA e molto meno da un altro, a seconda della resa dei colori dello UA.

Talvolta al posto delle coordinate HSL sono usate le coordinate *HSV* dove V (value) è anche detto Brightness (luminosità) che varia dallo 0% (nero) al 100% (colore puro). Il colore puro corrisponde al bianco se la saturazione è 0, ossia non è presente alcun colore.

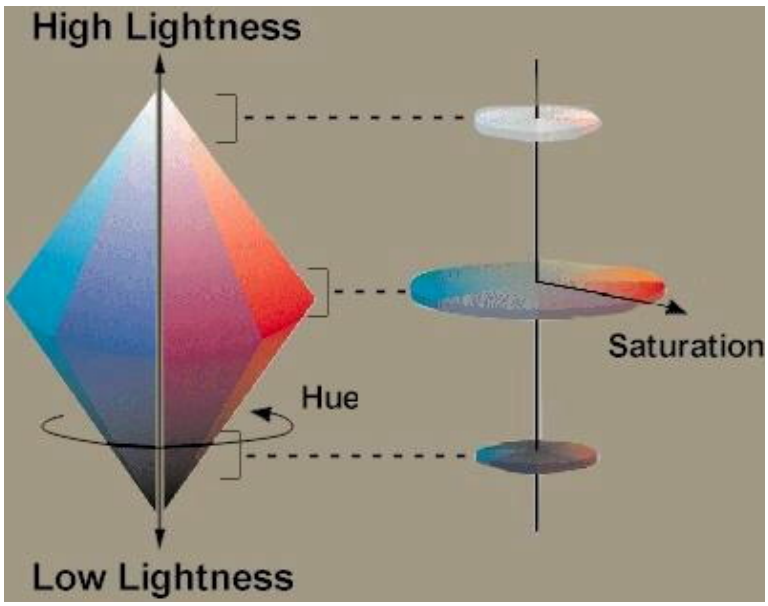

Figura 9.3: Le coordinate HSL dei colori.

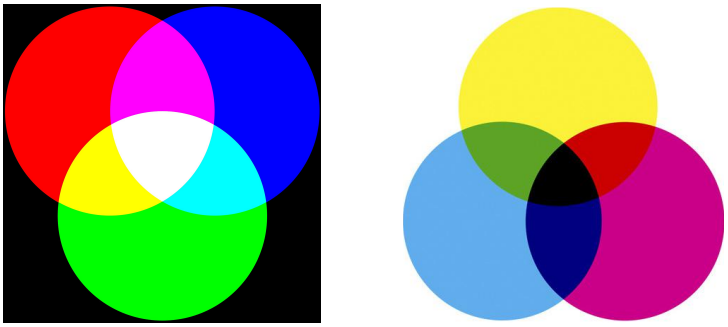

Figura 9.4: Il modello dei colori additivo (a sinistra) e quello sottrattivo (a destra).

9.5.2 Modelli dei colori

Un colore può essere ottenuto secondo due modelli opposti chiamati modello additivo e modello sottrattivo.

Il modello additivo è quello usato nei video che, normalmente, se spenti sono neri (non c'è colore). Per vedere delle immagini si deve accendere della luce. In questo modello i tre colori primari, che sono il rosso, il verde ed il blu (da cui il nome *RGB*, per Red-Green-Blue), si combinano come in figura 9.4 ed il bianco è la combinazione di tutti i colori.

Il modello sottrattivo è quello usato per la stampa su carta bianca (che è bianca perché riflette la luce ed ha tutti i colori che arrivano dai raggi solari). Si parte dal bianco e si sottrae luce (figura 9.4). Per questo modello i colori primari sono il ciano, il magenta ed il giallo (da cui il nome *CMY* per Cyan-Magenta-Yellow) ed essi sono colori sottrattivi. Anche se il nero è ottenibile come somma di sottrazioni dei tre colori primari, per maggior efficienza, le cartucce delle stampanti a colori includono un inchiostro nero (*CMYK* = CMY + blacK).

Quindi lo stesso colore è rappresentabile negli schemi RGB o CMY a seconda di dove deve essere visualizzato, ovvero video rispetto a carta stampata.

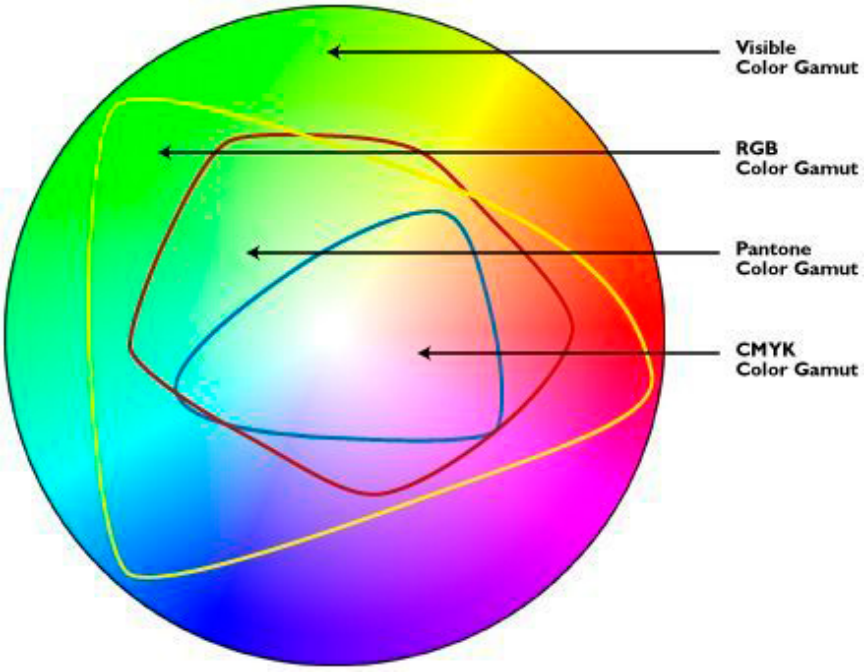

Figura 9.5: Il Gamut dei colori.

9.5.3 Colori utilizzabili

Il *Gamut* è l'intervallo di colori che è visibile o rappresentabile da un certo dispositivo. Esso è diverso per ogni dispositivo video o di stampa. Come mostrato in figura 9.5, esistono diverse aree ed in ordine decrescente di grandezza sono:

- Il cerchio esterno, che rappresenta tutti i colori che l'occhio umano può vedere, cioè il cosiddetto *Visible Color Gamut*.
- L'area racchiusa dalla linea gialla, che contiene i colori rappresentabili come RGB.
- L'area racchiusa dalla linea rossa, che contiene i colori *Pantone* (colori più usati per dipingere pareti).
- L'area racchiusa dalla linea blu, che contiene i colori CMYK. Questo significa che quando si stampa su carta non si riuscirà mai ad avere tutti i colori RGB. Dunque, un'immagine stampata risulterà sempre diversa rispetto alla stessa immagine mostrata a video, a meno che si abbia avuto l'accortezza di usare a video solo la gamma di colori del Gamut CMY. Questa accortezza è tipica del bravo progettista web.

Le linee rappresentate nella figura sono quelle che racchiudono tutto il corrispondente Gamut teorico ma, nella realtà, esse dipenderanno dalla qualità degli schermi (Gamut RGB) e dalla qualità delle stampanti (Gamut CMYK).

Per ovviare a questo problema sono stati definiti i colori web-safe e web-smart.

I colori *web-safe* sono quelli che hanno i byte del codice RGB scelti tra 00, 33, 66, 99, CC, FF (ad esempio 66FF99) ed in totale sono 6^3 combinazioni, quindi 216 colori. Questo è l'insieme originale disponibile per monitor a 8 bit e tali colori garantiscono il massimo di retro-compatibilità.

Con l'evoluzione tecnologica dei monitor le limitazioni dei colori Web-safe sono risultate evidenti. Così, sono stati definiti i colori *web-smart* che hanno i byte del codice RGB composti

da coppie identiche di cifre esadecimali da 0 a F (ad esempio 5522EE), per un totale di 16^3=4096 combinazioni. Questo è l'insieme esteso per i monitor a 16 bit.

Nella progettazione di una pagina web si deve avere lo scopo di mostrare i colori in modo consistente su qualunque monitor (non solo su PC ma anche su smart-phone e televisori). I colori web-safe e web-smart sono molto utili anche se oggi è frequente avere monitor a 24 bit, per i quali non è stata definita una scala di colori sicuri.

Si ricorda che è consigliato evitare, per quanto possibile, i gradienti di colore poiché la resa può essere molto differente a seconda dello UA.

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/
- 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 linguaggio JavaScript

Originariamente il linguaggio *JavaScript* (spesso abbreviato come JS) era stato pensato per il web, ma ora viene usato anche per altri ambienti. Ad esempio quando si carica un documento PDF bisogna prestare attenzione al fatto che questo non conterrà solamente le immagini ed il testo del documento originale, ma potrebbe contenere al suo interno del codice JS che serve, per esempio, nel caso in cui un utente dovesse cliccare su di una parola o svolgere delle azioni sul testo. Il linguaggio JS può essere usato lato client o lato server. Un caso particolare è rappresentato da alcuni server web che sono loro stessi scritti in linguaggio JS, twitter ne è un ottimo esempio: si tratta di un server scritto interamente in JS perché tutta l'interazione tra gli utenti è basata su eventi dovendo essere molto rapido e non essendo necessario caricare una grande quantità di dati. Come già visto in precedenza il linguaggio JS:

- può essere inserito in pagine HTML preceduto dal tag `<script>` e se viene inserito nell'`head` viene eseguito al caricamento della pagina, mentre se viene inserito nel `body` verrà eseguito quando si incontra il suo tag;
- può essere usato per istruzioni specifiche relative ad un determinato evento (tramite un eventHandler).

Quando si esegue il codice JS la normale funzione di interpretazione HTML viene interrotta. Infatti un browser in grado di capire il codice JS è dotato di due interpreti: uno per l'HTML e l'altro per JS; nel momento in cui si presenta del codice JS l'interprete HTML lascia spazio all'interprete JS.

Il linguaggio JS (figura 10.1) è composto da una parte core che costituisce il linguaggio di programmazione e poi da alcune funzioni che servono:

- solo quando viene eseguito lato client (es. l'interazione con la history o l'interazione con i click che hanno un senso solo se JS è eseguito su uno UA);
- solo quando viene eseguito lato server, dove le estensioni servono solo in questo ambiente.

Bisogna prestare attenzione al fatto che la parte core è una parte standard, così come lo è abbastanza anche la parte client (perché dettata dal DOM e quindi dall'interazione con il browser). L'unica che non è standard è la parte lato server, perché non esiste un unico tipo di server e quindi ogni sviluppatore ha creato un proprio modo per estendere il linguaggio JS.

estensioni client-side (es. window, history)	*Core JavaScript* (variabili, funzioni, controlli di flusso, ...)	*estensioni server-side* (es. database, gestione server)

Figura 10.1: JS core ed estensioni.

```
 <script type="text/javascript">
<!--
        document.writeln("<p>Ciao!></p>")
//-->
</script>
```

Figura 10.2: commenti HTML e JavaScript.

10.1 Sintassi: istruzioni e commenti

Il linguaggio JS è case-sensitive, ossia c'è differenza nello scrivere in maiuscolo o minuscolo. Ricordiamo che JS si comporta come il web, quindi quando incontra un errore non segnala nulla ma procede, non compare nessuna segnalazione ma il sistema potrebbe non funzionare e risulta difficile capire il problema. Ci sono due possibilità nella scrittura delle istruzioni:

- si può scrivere un'istruzione su di una riga ed andare a capo;

  ```
  a = 3 + b
  c = 2 * a
  ```

- si possono mettere più istruzioni su una stessa riga, ma si devono separare con un punto e virgola.

  ```
  a = 3 + b ; c = 2 * a;
  ```

Un'istruzione JS deve concludersi tutta quanta su di una riga, perché altrimenti se una parte venisse per esempio scritta nella riga sottostante non verrebbe considerata:

```
a = 3
  + b;
```

Si noti che non verrebbe neanche segnalato un errore ma semplicemente il risultato non sarà quello atteso (in questo specifico esempio sarà `a=3`).

E' buona norma di programmazione inserire durante il codice dei commenti esplicativi. In JS ciò può essere fatto in due modi diversi:

- inserendo il simbolo `//` il commento inizia da questo punto e finisce al finire della riga;
- inserendo il commento tra `/*` e `*/`.

Bisogna prestare particolare attenzione all'interazione tra i commenti HTML e JS. JS riconosce il simbolo di commento HTML `<!--` come simbolo di un commento lungo una ed una sola riga (finisce quando si passa alla riga successiva), ma non riconosce il simbolo di chiusura di commento HTML `-->`. Quindi per utilizzare il commento HTML è bene premettere il

simbolo di commento JS // (vedi figura 10.2). Questi simboli erano usati in passato per nascondere il codice JS a tutti quei browser che non erano in grado di interpretarlo. Se il browser non conosceva questo linguaggio tutta la parte relativa ad esso veniva commentata. Al giorno d'oggi è veramente raro incontrare dei browser che non conoscano JS. Un'ottima soluzione comunque sarebbe quella di caricare il linguaggio JS su di un file esterno in modo tale che un browser che non lo conosca non ne carichi direttamente il relativo file.

10.2 Tipi dati, variabili e costanti

Sappiamo che in ogni linguaggio di programmazione ci sono dei tipi di dati che corrispondono a precise codifiche. In JS è possibile utilizzare:

- i numeri decimali (es. 14, -7, 3.14, 10.7e-4), i numeri scritti direttamente in codice ottale che devono sempre essere preceduti da uno 0 (es. 016 non è il numero 16, ma è 1 e 6 in ottale, quindi 8+6=14) ed i numeri esadecimali che devono essere sempre preceduti da 0x (es. 0xE);
- i valori Booleani (scrivendo `true` e `false`);
- le stringhe di caratteri che possono essere racchiuse sia tra i doppi apici che tra i singoli (es."ciao mamma", 'ciao babbo');
- gli oggetti;
- il valore speciale `null` che indica tipicamente una variabile alla quale non è ancora stato assegnato un valore, non ancora inizializzata;
- il valore speciale `undefined` che indica che la cella di memoria a cui un utente sta facendo riferimento ha un nome che non è mai stato definito;
- il valore speciale `NaN` (Not a Number), quando si vuole utilizzare come numero qualcosa che non è un numero.

Le celle di memoria vengono identificate tramite il loro nome:

- il nome deve iniziare con un carattere alfabetico, `$` o `_`;
- possono contenere caratteri alfanumerici, `$` o `_`;

Esempi di nomi di variabili validi sono:

```
costo    $1    studente_12345    _2009q1
```

Importante ricordare che, mentre nel linguaggio C è necessario indicare il tipo di una variabile (es. int, float, char, ...), in JS non è necessario. Il tipo di dati contenuti in una variabile può cambiare continuamente grazie al fatto che il linguaggio JS è interpretato. Le variabili assumono un tipo al momento della loro inizializzazione e possono poi cambiarlo automaticamente, per adattarsi al contesto in cui vengono utilizzate.

Per creare una variabile si può usare l'istruzione esplicita `var`:

- `var totale;` contiene il valore speciale `null` perché non è ancora stata inizializzata;

- `var totale = 0;` è una variabile inizializzata ed è anche di tipo intero;
- `var saluto = "ciao";` è una variabile stringa inizializzata.

Non è necessario utilizzare `var` se decidiamo ad un certo punto del codice di usare una variabile dandole un valore.

```
totale = 0;
```

Questa viene automaticamente creata dall'interprete se in precedenza non esisteva.

E' sempre comunque consigliabile definire tutte le variabili all'inizio del codice. Se decidiamo di usare una variabile come ad esempio `totale+5`, senza averla prima inizializzata:

- questa potrà essere `undefined`/`NaN` se era stata dichiarata tramite l'istruzione `var`;
- si genererà un runtime error, ovvero la parte di JS si blocca se la variabile non era stata dichiarata. Per errore si intende che tutto il codice precedente l'errore verrà eseguito, mentre tutto quello successivo all'errore non viene eseguito ma l'interpretazione non si ferma e procede grazie al doppio sistema d'interpretazione, poiché gli errori che si verificano in questa parte di codice non bloccano la pagina.

Se si desidera creare una cella di memoria contenente un valore fisso (ossia una costante) si usa l'istruzione `const`:

```
const iva = 0.19;
const autore = "A. Lioy";
```

All'interno di una stringa è possibile usare direttamente la codifica ISO o Unicode, quindi si possono utilizzare direttamente le lettere accentate, ma bisogna ricordare che questo è possibile solo fino a quando si è all'interno del codice JS. Si possono inoltre utilizzare .le classiche sequenze di escape del linguaggio C (es. `\r \n \t \\`) ed inoltre, visto che gli apici semplici e doppi hanno un significato particolare, per essere usati semplicemente come simboli devono essere preceduti da \, ossia `\'` e `\"`.

10.3 Input e output

Non esiste in JS uno standard di istruzioni input ed output, ma dipende dall'ambiente di esecuzione. Per l'input lato client si possono usare:

- i dati provenienti da una finestra di pop-up (alcuni siti possiedono questa tipologia di finestre mentre altri contengono al loro interno delle zone in cui è possibile inserire dell'input);
- i dati provenienti da un form HTML (tramite DOM);

Il primo è direttamente JS, mentre il secondo è un evento generato che farà poi parte di JS. Il pop-up è la forma più diretta mentre il form è più indiretto, ma allo stesso tempo più generale perché alcuni browser bloccano i pop-up. Per l'output lato client si possono usare:

- una finestra di pop-up come per l'input;
- scrivere il risultato tramite DOM nella pagina HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Pop-up di I/O: esempio 1</title>
<script type="text/javascript">
var n = window.prompt("Nome?", "nessuno")
window.alert(n);
</script>
</head>
<body>
<p>Fine dell'esempio.</p>
</body>
</html>
```

Figura 10.3: Primo esempio di sintassi associato a pop-up di I/O.

Pop-up di I/O

Per generare un pop-up di output si può usare un oggetto `window` che è uno degli oggetti DOM resi disponibili dal linguaggio JS:

- `window.alert (messaggio)` associato al metodo `alert` rappresenta la finestra attuale del browser creando un pop-up contenente il testo che viene indicato tra parentesi. Se si vuole scrivere direttamente il testo lo si mette tra " ", se invece il testo è contenuto in una variabile è sufficiente richiamare direttamente la variabile. Si apre un pop-up bloccante che deve per forza essere letto dall'utente e contiene un solo pulsante `ok` che deve essere premuto una volta letto il testo della finestra per poter proseguire;
- `window.prompt(prompt_msg[, valore_iniziale]` è invece un pop-up di input, si tratta di una finestra in cui viene mostrato come minimo un messaggio e richiesto un valore (opzionalmente è possibile inserire un valore iniziale). Il pop-up avrà due pulsanti: `ok` che viene cliccato dall'utente una volta fornita la risposta o `Cancel` che viene premuto altrimenti. Questo oggetto dunque restituirà: il valore inserito se l'utente decide di rispondere, oppure `null` nel caso l'utente abbia premuto il tasto Cancel o la X che chiude direttamente la finestra.

Questi non sono oggetti JS ma implementazioni dell'oggetto `window` che fa parte del `DOM livello 0` e relativi metodi.

Le istruzioni vengono scritte all'interno di uno script (figura 10.3), che verrà eseguito al caricamento della pagina (perché si trova nell'head). Lo script comprende una variabile `n` che contiene come valore quello restituito da una finestra di pop-up. Il `prompt` richiede all'utente di inserire il suo nome e nel campo ci sarà già preimpostato il valore "nessuno", tramite `window.alert(n)` dice qual è il nome introdotto. Ricordiamo che `window.prompt` è un pop-up bloccante quindi la pagina HTML non verrà visualizzata fino a quando non verrà fornita una risposta alla domanda.

E' anche possibile (figura 10.4) inserire lo script non nell'head ma nel body. In questo caso prima di visualizzare il pop-up verrà visualizzata la frase "`Inizio dell'esempio`" ed alla fine di questo verrà visualizzato "`Fine dell'esempio`". Nell'HTML della pagina non compare traccia del codice JS.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Pop-up di I/O: esempio 2</title>
</head>
<body> <body>
<p>Inizio dell'esempio.</p>
<script type="text/javascript">
var n = window.prompt("Nome?", "nessuno")
window.alert("Ciao "+n);
</script>
<p>Fine dell'esempio.</p>
</body>
</html>
```

Figura 10.4: Secondo esempio di sintassi associato a pop-up di I/O.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Output tramite HTML: esempio</title>
</head> </head>
<body>
<script type="text/javascript">
var n = window.prompt("Nome?", "nessuno");
document.writeln("<p>Ciao "+n+"</p>");
</script>
</body>
</html>
```

Figura 10.5: Esempio di output tramite HTML.

Output tramite HTML

Per inserire output generato da JS in HTML è possibile usare uno dei seguenti metodi di scrittura associato all'oggetto DOM `document`:

- `document.write(text)` inserisce il testo indicato tra le parentesi nel punto esatto in cui siamo arrivati;
- `document.writeln(text)` inserisce il testo e va anche a capo, ovvero inserisce una coppia CR LF.

In figura 10.5 si può notare che l'input viene sempre fatto con l'oggetto `window.prompt` ma invece che usare un pop-up viene utilizzato un `document.writeln` e quindi il testo inserito dall'utente viene inserito nella pagina concatenando a "Ciao" il nome dato.

10.4 Operatori

10.4.1 Operatori relazionali e logici

Gli operatori relazionali e logici sono utili per il confronto di variabili e dati. Nella tabella sottostante è possibile vedere quali siano gli operatori di questo tipo disponibili in JS.

descrizione	*simbolo*
uguaglianza (valore)	`==`
identità (valore e tipo)	`===`
disuguaglianza (valore)	`!=`
non identità (valore e tipo)	`!==`
maggiore di / maggiore o uguale a	`> >=`
minore di / minore o uguale a	`< <=`
appartenenza	`in`
AND logico	`a`
NOT logico	`!`
OR logico	——

Si tratta degli stessi operatori presenti nel linguaggio C fatta eccezione per:

- `===` che rappresenta l'uguaglianza di due elementi considerando sia il valore che il rispettivo tipo (es. se si uguagliassero due zeri ma uno è floating point e l'altro è intero sarebbero uguali per valore ma non per tipo);
- `!==` opposto a quello sopra riportato;
- `in` indica se il valore a cui si riferisce appartiene ad un determinato insieme(come vedremo più avanti gli insiemi vengono definiti con degli array);

In precedenza si è vista l'esistenza dell'operatore `false` che equivale a Falso, ma è bene sapere che esistono altri valori equivalenti ad esso:

- `0`
- `NaN`
- la stringa vuota " "
- `null`
- `undefined`

Qualsiasi altro valore equivale a Vero. Attenzione ai confronti:

- `27==true` fornisce valore Vero perché in effetti l'operatore utilizzato confronta le due variabili in base al valore e 27 non corrisponde a nessuno dei valori sopra indicati pertanto non può essere Falso;
- `27===true` confronta invece i due dati in base sia al valore che al tipo e fornirà valore Falso perché `true` è una variabile Booleana mentre `27` è un intero.

10.4.2 Operatori aritmetici

Gli operatori aritmetici sono gli stessi del linguaggio C ma in JS valgono sia per numeri interi che per numeri floating-point.

descrizione	*simbolo*
addizione	`+`
incremento unitario	`++`
sottrazione	`-`
decremento unitario	`--`
moltiplicazione	`*`
divisione (floating-point)	`/`
modulo (resto della divisione intera)	`%`

10.4.3 Operatori di assegnazione

descrizione	*simbolo*	*esempio*	*equivalenza*
assegnazione	`=`	`a = 5`	
assegn. con somma	`+=`	`a += 5`	`a = a + 5`
assegn. con sottrazione	`-=`	`a -= 5`	`a = a - 5`
assegn. con prodotto	`*=`	`a *= 5`	`a = a * 5`
assegn. con divisione	`/=`	`a /= 5`	`a = a / 5`
assegn. con modulo	`%=`	`a % 5`	`a = a % 5`

10.5 Le stringhe di caratteri

Qualunque input inserito dall'utente tramite browser è una stringa.

- l'assegnazione può avvenire semplicemente con l'uso dell'operatore `=` ;
- a differenza del linguaggio `C` è possibile confrontare in ordine alfabetico le stringhe tramite gli operatori relazionali (`== != > >= < <=`);
- la concatenazione è possibile tramite gli operatori `+` e `+=`.

Qualsiasi stringa, numero, simbolo inseriti vengono trattati come stringhe quindi bisogna fare sempre uso delle parentesi. Vediamo alcuni esempi:

```
ris =  "N=" + 5 + 2; // ris = N=52
```

In questo primo caso la prima variabile è una stringa e l'operatore `+` significa "concatena" perciò si ha come risultato la concatenazione dei tre elementi;

```
ris =  "N="  + (5 + 2); // ris =  N=7
```

Inserendo i numeri tra due parentesi queste ultime avranno la priorità e l'operatore al loro interno è aritmetico quindi avremo come risultato la somma dei due numeri concatenata alla prima variabile;

```
ris = "N=" + 5 - 2; // ris = NaN
```

Concatenando invece il numero `5` ed il numero `-2` il simbolo `-` non ha nessun significato nell'ambito delle stringhe pertanto l'operazione non riuscirà perché si tratta di un calcolo tra un numero ed una variabile che non è un numero.

```
n = Number("2"); // n = 2
n = Number("2.3"); // n = 2.3
n = Number("2,3"); // n = NaN
n = Number("2mila"); // n = NaN
n = Number("2 mila"); // n = NaN
```

Figura 10.6: Esempio di conversioni stringhe - numeri.

```
n = parseInt("10",2); // n = 2
n = parseInt("2.3"); // n = 2
n = parseInt("2,3"); // n = 2
n = parseInt("2mila"); // n = 2
n = parseInt("duemila"); // n = NaN
```

Figura 10.7: Esempio di conversioni di stringhe e numeri con `parseInt` e `parseFloat`.

10.5.1 Conversioni di stringhe in numeri

Ogni qualvolta venga creato un pop-up di prompt in cui viene chiesto all'utente di inserire un valore, come già detto, qualunque valore egli inserisca viene letto come stringa. Nel caso in cui questo valore voglia essere usato come numero deve essere trasformato. Il metodo usato è `Number (oggetto)` che restituisce una rappresentazione numerica dell'oggetto, se la conversione è possibile, o il valore `NaN` in caso contrario. Viceversa se è necessario applicare una trasformazione da un numero ad una stringa si usa il metodo `String (oggetto)` che restituisce una rappresentazione a caratteri dell'oggetto, oppure i valori `null` o `undefined`.

In figura 10.6 è possibile osservare alcune conversioni. Nei primi due casi si tratta di due numeri uno intero ed uno decimale, il primo viene interpretato come un intero e viene restituito il suo valore ed il secondo che ha tre caratteri, viene anch'esso interpretato come numero ed il valore restituito è appunto un numero decimale. I restanti tre esempi rappresentano tre conversioni che restituiscono tutte il valore `NaN`. Nel primo caso perché la virgola non è un carattere appartenente ai numeri e quindi la stringa passata non è convertibile, così come nei due casi successivi la parola "mila" e lo spazio non sono convertibili in numero.

In questo modo ogni volta che viene chiesto all'utente di inserire un valore, con i metodi appena visti, è possibile controllare se il valore inserito sia un numero o meno.

Il linguaggio JS fornisce dei comandi utili per interpretare la variabile inserita dall'utente:

- `parseInt(stringa[, base ])` è una funzione che interpreta la variabile come se fosse un numero intero ed è anche possibile specificare in quale base si desidera l'interpretazione (se la base non è indicata si avrà come valore di default la base 10). Se la funzione riesce ad interpretare il numero restituirà in valore interpretato nella base richiesta, altrimenti restituirà il valore `NaN`.

- `parseFloat(stringa)` interpreta la variabile inserita come un numero floating point (decimale). Anch'essa in caso di interpretazione avvenuta con successo restituisce il risultato interpretato e in caso contrario il valore `NaN`.

Entrambe le funzioni di `parse` considerano solamente la parte iniziale e si fermano dunque al primo carattere non valido.

Osserviamo nella figura 10.7 alcuni esempi di interpretazioni valide e non. Nel primo caso si richiede di interpretare la stringa "10" in base 2, si tratta di una stringa binaria e si otterrà come risultato `n=2`. Se si richiede l'interpretazione `parseInt` di un numero decimale come `2.3` la funzione considererà solamente la parte intera e fornirà lo stesso risultato dell'esempio precedente. Inserendo i valori `2,3` e `2mila` la funzione non riconosce la virgola come un simbolo appartenente ai numeri, così come il carattere `m` della parola "mila", e si ferma al primo dato restituendo sempre `n=2`. L'unico caso in cui la conversione non è riuscita è l'ultimo in cui nessuno dei caratteri (essendo tutti alfabetici) viene riconosciuto e pertanto il risultato è `NaN`.

10.5.2 Proprietà e metodi dell'oggetto String

Nel linguaggio JS è possibile manipolare e controllare le stringhe attraverso una serie di proprietà e metodi della classe String. Questo perché, se creiamo una variabile con la normale sintassi

```
var variabile=‘‘testo’’;
```

Javascript effettua automaticamente la conversione da stringa ad oggetto String e viceversa.

Proprietà: Esiste una sola proprietà utile par la manipolazione delle stringhe, si tratta di `length` che calcola la lunghezza della stringa, nello specifico il numero di caratteri che la compone.

Metodi: tutti i metodi seguono la forma sintattica variabile.metodo(); . I metodi più utilizzati sono i seguenti:

- `charAt(pos)`: restituisce il carattere in posizione *pos* (ricordando che l'indice di conteggio dei caratteri di una stringa lunga N parte da 0 ed arriva a N-1);

  ```
  var testo = "Hello, world!";
  document.write(testo.charAt(0));  //restituisce "H"
  document.write(testo.charAt(testo.length - 1));  //restituisce "!"
  ```

- `charCode(pos)`: funziona come il metodo charAt(), ma restituisce il codice numerico Unicode del carattere in posizione *pos*;

- `indexOf(search[,start])`: questo metodo cerca, nella stringa su cui viene chiamato, la prima occorrenza della sotto-stringa *search* e ne restituisce la posizione (a partire da 0); se la stringa non viene trovata, restituisce -1. Il parametro opzionale *start* specifica la posizione dalla quale iniziare la ricerca (di default è 0);

  ```
  var stringa = "Hello, world!"
  var posiz = stringa.indexOf("w");  // valore 7
  var posiz2 = stringa.indexOf("k");  // valore -1
  ```

- `lastindexOf(search[,start])`: questo metodo funziona analogamente ad indexOf() ma inizia la ricerca dalla fine della stringa;

- `slice(begin[,end])`: permette di estrarre caratteri da una stringa iniziando dal carattere *begin* fino a fine stringa, o da begin fino al carattere *end* (escluso), se specificato;

```
var stringa="Hello happy world!";
document.write(stringa.slice(6,11));   //restituisce "happy"
document.write(stringa.slice(-6));   //restituisce "world!"
```

- `substring(begin[,end])`: metodo che permette di creare una reale sotto-stringa, estraendo dalla stringa su cui viene chiamato i caratteri compresi tra l'indice *begin* e fine stringa o tra begin e il carattere *end* (escluso), se specificato;

```
var stringa="Hello world!";
document.write(stringa.substring(3));   //restituisce "lo world!"
document.write(stringa.substring(3,7));   //restituisce "lo w"
```

- `substr(begin[,length])`: questo metodo permette di estrarre un numero di caratteri pari a *length*, partendo dalla posizione *begin* della stringa su cui viene chiamato. Se il numero di caratteri da estrarre non viene specificato, il metodo include tutti i caratteri a partire dalla posizione *begin* fino a fine stringa;

```
var stringa="Hello world!";
document.write(stringa.substr(3));  //restituisce "lo world!"
document.write(stringa.substr(3,4));  //restituisce "lo w"
```

- `toLowerCase()`: restituisce la stringa con i caratteri convertiti in minuscolo;

- `toUpperCase()`: restituisce la stringa con i caratteri convertiti in maiuscolo;

10.6 Test sui valori errati

Nel caso si voglia testare se una variabile abbia o meno valore numerico, ossia se presenti il valore Nan, non ci si può limitare a controllare se questa contenga i caratteri "Nan". Questo perché potremmo trovarci di fronte ad una stringa contenete al suo interno proprio questa sequenza di caratteri. In tal caso una semplice stringa verrebbe interpretata come un valore non numerico. Per testare, quindi, i casi strani non si può guardare direttamente il valore della variabile, ma è necessario utilizzare particolari accorgimenti. Ad esempio:

- `isFinite(number)`: metodo che restituisce valore vero se il numero (*number*) non è pari a +/- infinito o non è pari a "Nan";

- `isNan(number)`: è il modo corretto per testare se un numero non è un numero. Restituisce valore vero se il numero (*number*) ha valore" NaN";

- `typeof(x)`: poiché il linguaggio Javascript è non tipato, ossia utilizza variabili che possono cambiare tipo nel tempo, si ha la necessità di un metodo che esprima quale tipo di dato corrisponde alla variabile x nel momento in cui si esegue il test.
  Tale test può restituire come possibile risposta il valore boolean, function, number, object, string o undefined (quest'ultimo corrisponde al caso in cui alla variabile x non sia ancora stato assegnato alcun valore).

10.7 Controllo di flusso

Come tutti i linguaggi di programmazione, JS prevede strutture di controllo utili a verificare l'esecuzione di un programma in modo non sequenziale. Sotto molti aspetti la sintassi di tali controlli è simile a quella del linguaggio C.

Vi sono due tipi di controlli di flusso: quelli che consentono di prendere delle decisioni (`if`, `if-else` e `switch`) e quelli che consentono di eseguire cicli di istruzioni (`while`, `do-while`, `for` e `for-in`).

10.7.1 Controllo di flusso "if" / "if-else"

Questi due tipi di controlli prevedono l'esecuzione condizionale di un blocco di istruzioni in base al valore assunto da una condizione Booleana.

Nel caso IF semplice viene eseguito il blocco di istruzioni se si verifica la condizione data:

```
if (condizione)
{
   ... istruzioni ...
}
```

Nel caso IF-ELSE viene eseguito un blocco di codice se si verifica la condizione indicata, un diverso blocco di istruzioni in caso contrario:

```
if (condizione)
{
   ... blocco istruzioni 1 ...
}
else
{
   ... blocco istruzioni 2 ...
}
```

Ecco un esempio di costrutto "if-else":

```
<script type="text/javascript">
   var temp = window.prompt("Temperatura misurata?");
   if (temp <= 0)
      alert("l'acqua e' ghiacciata");
   else if (temp >= 100)
      alert("l'acqua e' vapore");
   else
      alert("l'acqua e' allo stato liquido");
</script>
```

Attraverso una finestra di pop-up questo programma richiede il valore di temperatura all'utente e lo memorizza in una variabile *temp*. Usando i costrutti "if" e "if-else" si testano i possibili valori assunti dalla variabile e, attraverso una seconda finestra di pop-up, viene restituito un messaggio all'utente.

10.7.2 Controllo di flusso "switch"

Il costrutto "switch" rappresenta una forma alternativa ad un "if-else" particolarmente ramificato, in cui viene presa in esame un'unica variabile che può assumere differenti valori.

Tra parentesi viene indicata l'espressione, seguita dai vari *case* con i differenti valori da testare. Viene inoltre utilizzata l'istruzione *break*, che serve ad interrompere lo switch per non continuare con il caso successivo, ogni volta che si rientra nella casistica che ci interessa. Nel caso la variabile da testare non ricada in nessuno dei casi previsti, viene eseguita l'istruzione di default.

```
switch (espressione)
{
case valore1: ... istruzioni;
break;
case valore2: ... istruzioni;
break;
...
default: ... istruzioni;
}
```

Viene di seguito presentato un esempio di costrutto "switch":

```
<script type="text/javascript">
   var frutto = window.prompt("Quale frutto vuole?");
   switch (frutto) {
   case "pera":
      alert ("pere a 2 Euro/kg"); break;
   case "mela":
      alert ("mele a 1.5 Euro/kg"); break;
   case "banana":
      alert ("banane a 1 Euro/kg"); break;
   default:
      alert ("spiacenti, non abbiamo "+frutto);
   }
</script>
```

Questo esempio mostra come il valore da testare venga acquisito attraverso una finestra di pop-up e memorizzato all'interno di una variabile. A seconda del valore assunto da tale variabile, verrà visualizzato all'utente un messaggio differente e, nel caso il valore non sia tra quelli previsti dai case, viene visualizzato il messaggio di default. Per migliorare questo programma, potremmo applicare il metodo toLowerCase alla variabile *frutto*, in quanto l'utente potrebbe inserire stringhe con caratteri maiuscoli e in tal caso il programma visualizzerebbe sempre il messaggio di default.

10.7.3 Controllo di flusso "while"

Questa struttura viene utilizzata per ripetere un blocco di istruzioni finché una certa condizione è e rimane valida. Le istruzioni del ciclo possono quindi essere eseguite più volte se la condizione rimane vera, o nessuna se la condizione è falsa già in partenza.

```
while (condizione)
{
```

```
    ...istruzioni...
}
```

Viene di seguito presentato un esempio di ciclo "while":

```
<script type="text/javascript">
   var x = 5;
   while (x >= 0)
   {
      alert(x);
      x--;
   }
</script>
```

Questo programma decrementa la variabile x ad ogni iterazione, finché questa raggiunge valore nullo e visualizzandone il relativo valore attraverso una finestra di pop-up. Nel momento in cui tale variabile assume valore -1, la condizione del while non risulta più verificata ed il ciclo si interrompe.

10.7.4 Controllo di flusso "do-while"

Questo struttura è simile al costrutto "while", con la sola differenza che, essendo il controllo svolto alla fine del ciclo, quest'ultimo viene sempre eseguito almeno una volta.
Il "do-while' implementa il costrutto della programmazione strutturale "repeat-until", ossia le istruzioni vengono prima eseguite e poi si testa una condizione per verificare se è necessario ripeterle. Il blocco di istruzioni viene ripetuto finché tale condizione risulta verificata.

```
do
{
    ...istruzioni...
} while ( condizione );
```

Esempio di ciclo "do-while":

```
<script type="text/javascript">
   var ris;
   do {
   ris = window.prompt(
      "Scrivi 'ciao' o resti bloccato qui");
   } while (ris != "ciao");
</script>
```

Questo esempio mostra come, finché l'utente non inserisca nella finestra di pop-up la parola "ciao", la condizione del ciclo "do-while" rimanga verificata e l'istruzione continuerà ad essere eseguita.

10.7.5 Controllo di flusso "for"

Questa struttura permette di eseguire un blocco di istruzioni finché una certa condizione rimane vera. All'interno di questo costrutto viene inizializzato un indice, viene valutata la condizione e viene incrementato o decrementato l'indice in modo da poter ripetere le istruzioni.

```
for ( inizializzazione ; condizione; azione_ripetitiva )
{
... istruzioni_da_ripetere
}
```

Esempio di ciclo "for" numerico:

```
<script type="text/javascript">
var totale = 0;
for (var i=1; i <= 10; i++)
{
totale = totale + i;
}
alert("Somma dei numeri [1...10] = "+totale);
</script>
```

In questo caso è stato utilizzato un ciclo for per il calcolo della somma dei primi 10 numeri naturali. Per poter eseguire il ciclo è stata creata appositamente la variabile i. Tale variabile verrà "distrutta", liberando memoria, non appena l'esecuzione del ciclo sarà terminata. Attraverso una finestra di pop-up viene restituito il risultato all'utente.

10.7.6 Istruzioni "break" e "continue"

L'istruzione `break` viene usata per interrompere l'esecuzione del ciclo in cui è contenuta, in modo da riprendere l'esecuzione del programma dalla prima istruzione successiva al ciclo.

Esempio:

```
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
  {
  if (i==3)
    {
    break;
    }
  document.write("Numero " + i);
  document.write("<br />");
  }
</script>
```

Utilizzando il "break", il ciclo si interrompe alla quarta iterazione, visualizzando:

```
Numero 0
Numero 1
Numero 2
```

L'istruzione `continue`, invece, blocca l'esecuzione dell'istruzione corrente, facendo proseguire il programma dalla prossima iterazione del ciclo in cui è contenuta.

Esempio:

```
<script type="text/javascript">
var i=0
```

```
for (i=0;i<=10;i++)
  {
  if (i==3)
    {
    continue;
    }
  document.write("Numero " + i);
  document.write("<br />");
  }
</script>
```

In questo caso l'istruzione "continue" permette di saltare la quarta iterazione, visualizzado i numeri da 0 a 2 e da 4 a 10.

10.8 Array

Gli array sono dei vettori, quindi nel linguaggio JS rappresentano degli oggetti sui quali è possibile chiamare dei metodi.
Gli array devono essere istanziati, dichiarando una dimensione iniziale. Nel caso, però, ci accorgessimo di aver bisogno di più celle di quelle istanziate, il linguaggio JS ci consente di espandere l'array. Per far ciò è necessario interrompere l'esecuzione del programma e copiare le celle in un'altra zona di memoria più grande. L'estensione dell'array avviene automaticamente, ma ha un costo per l'esecuzione del programma che rallenta.

Gli array possono avere indice numerico o composto da una stringa (in questo caso si parla di **array associativi**).
Poichè le variabili utilizzate nel linguaggio JavaScript non sono tipate, ogni singolo elemento del vettore può contenere un tipo diverso di dato. Inoltre gli array possiedono proprietà e metodi per inserire, cancellare e reperire gli elementi.

Esempio:

```
var Vettore = new Array(10);
for (var i=0; i<10; i++) {
Vettore[i] = "Test " + i;
}
```

Questo esempio mostra l'utilizzo di un vettore costituito da 10 elementi. Ad ogni iterazione del ciclo for viene salvata, in una cella dell'array, la stringa contenete la parola "Test" seguita dal numero dell'indice del vettore.

10.8.1 Array con indice numerico

Esempio:

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

var it2eu = new Array(32) // 0 ... 30 30L
for (var i=0; i<18; i++) it2eu[i] = "D"
for (var i=18; i<24; i++) it2eu[i] = "C"
for (var i=24; i<29; i++) it2eu[i] = "B"
```

```
for (var i=29; i<=31; i++) it2eu[i] = "A"
var voto = prompt("Voto italiano?")
alert("Voto europeo = " + it2eu[voto])

</script>
```

Questo programma permette di convertire i voti italiani in voti europei. L'array di conversione è costituito da 32 elementi e presenta un indice numerico. Per ottenere il corrispondente voto europeo, è sufficiente estrarre dal vettore l'elemento in posizione *voto*.

10.8.2 Array con indice non numerico

Primo esempio:

```
<script type="text/javascript">
   var vocab = new Array()
   vocab["giallo"] = "yellow"
   vocab["rosso"] = "red"
   vocab["verde"] = "green"

   var colore = prompt("Colore?")
   alert( colore + " = " + vocab[colore] )
</script>
```

Questo esempio rappresenta un traduttore di parole da italiano a inglese.
In questo caso non è stata determinata a priori la dimensione dell'array, per cui il numero di celle viene espanso man mano che l'utente inserisce i dati. Attraverso un prompt viene chiesto in input un colore e, indicando come indice del vettore *vocab* la stringa che rappresenta tale colore, ne viene restituita in output la traduzione. Se, però, in input si indica un colore non presente nell'array, il prompt di output restituirà il valore "undefined".

Secondo esempio:

```
<script type="text/javascript">
   var vocab = new Array()
   vocab["giallo"] = "yellow"
   vocab["rosso"] = "red"
  vocab["verde"] = "green"

   var colore = prompt("Colore?")
   if (typeof(vocab[colore]) != "undefined")
      alert( colore + " = " + vocab[colore] )
   else
      alert("Spiacente, traduzione non disponibile")
</script>
```

Questo secondo esempio presenta lo stesso traduttore dell'esempio precedente, al quale però è stato aggiunto un test per gestire il caso in cui la parola da tradurre sia sconosciuta al vocabolario. Per far ciò si testa il tipo della cella avente come indice il colore specificato in input. Se il tipo della cella risulterà "undefined", significa che tale cella non è stata ancora creata e all'utente verrà comunicato che la traduzione non è disponibile; in caso contrario verrà mostrata in output la traduzione della parola.

10.8.3 Controllo di flusso "for-in"

Questa struttura permette di verificare l'appartenenza di una variabile ad un determinato insieme e può essere applicata sia a vettori che ad oggetti.

"For-in" con vettori

Per quanto riguarda i vettori, permette di scandire tutti gli elementi dell'array senza conoscerne la relativa dimensione e l'indice che lo scorre assume tutti i valori numerici compresi tra 0 e lenght-1 (ultimo elemento).

Esempio:

```
var vettore = new Array(10);
for (var i in vettore)
document.writeln(vettore[i]+"<br>");
```

Questo esempio mostra come l'utilizzo di un ciclo "for-in" non necessiti dell'inizializzazione di una variabile per poter eseguire le iterazioni e neanche di una condizione di terminazione del ciclo, ma soltanto della dichiarazione di una variabile (in questo caso *i*), che andrà ad assumere il valore di tutti gli indici dell'array, scorrendolo dall'inizio alla fine.

"For-in" con oggetti

Per quanto riguarda gli oggetti, invece, il ciclo "for-in" ne scandisce le proprietà (ossia tutti i valori memorizzati all'interno dell'oggetto e che hanno un proprio nome) senza doverne conoscere il nome, in quanto l'indice assumerà tutti gli identificativi delle proprietà, nell'ordine in cui sono dichiarate.
Gli oggetti sono caratterizzati anche da metodi, ma il "for-in" ne scandisce solo le proprietà.

Esempio:

```
var myObject = new Object();
myObject.name = "Antonio";
myObject.age = 24;
myObject.phone = "123456";
...
for (var prop in myObject)
{
   document.writel("<p>myObject."+ prop+ " ="+ myObject[prop] + "</p>");
}
```

In questo esempio è stato creato un oggetto (*myObject*), costituito da 3 proprietà: *name, age* e *phone*. Per stamparle viene usato un ciclo "for-in", all'interno del quale è stata definita una variabile *prop* che assume il valore di tutte le possibili proprietà dell'oggetto.
Tale variabile non è una proprietà, ma una stringa; è un indice che rappresenta il nome della proprietà e va quindi indicata tra parentesi quadre dopo il nome dell'oggetto.

10.9 Funzioni

Le funzioni vengono chiamate quando si verifica un determinato evento.
Si definiscono con l'istruzione *function*, seguito dal nome della funzione e tra parentesi ne

vengono indicati i parametri, ossia le informazioni che si vuole passare alle funzioni.
Per terminare l'esecuzione della funzione si può usare l'istruzione *return*, o *return(valore)* se si vuole uscire dalla funzione avendo in output un valore.

```
function nome_funzione (par1, par2, ...)
{
... istruzioni ...
}
```

Vengono di seguito presentati due esempi di funzioni.

```
function somma (a, b) { return(a+b); }
document.write(somma(1,2));
```

Questa funzione può essere utilizzata sia con variabili numeriche che con stringhe.
Se come parametri si passano 2 numeri, la funzione ne restituisce la somma algebrica; se i parametri in input sono invece delle stringhe, la funzione somma ne fa la concatenazione.

```
function minoreDi (a, b) {
if (a < b) return (true) else return (false);
}
var a=5;
var b=2;
if (!minoreDi(a,b))
document.write(a + "non è minore di " + b);
```

Questa funzione non effettua operazioni sulle variabili, ma restituisce valore *true* se la condizione indicata nell'if risulta verificata, *false* in caso contrario.

10.9.1 Variabili locali e globali

Variabili locali

Una variabile locale viene dichiarata all'interno di una funzione ed è accessibile soltanto alle rispettive istruzioni; per questo, al termine della funzione, la variabile viene distrutta automaticamente.

Variabili globali

Una variabile globale viene dichiarata nel corpo dello script, all'esterno di qualunque funzione. Risulta accessibile a tutte le istruzioni, incluse quelle nelle funzioni richiamate dallo script e viene automaticamente distrutta al termine dello script.

Viene riportato un esempio di utilizzo di variabili locali e globali.

```
<script type="text/javascript">
        var i=2; // variabile globale
        function print_var()
        {
                var j=4; // variabile locale a print_var()
                alert("print_var(): i="+i);
                alert("print_var(): j="+j);
```

```
        }
        print_var();
        alert("i="+i);
        alert("j="+j );
        document.writeln("<p>Fine dello script.</p>")
</script>
<p>Fine della pagina.</p>
```

In questo script sono state usate 2 variabili: *i*, variabile globale definita a livello di script, e *j*, variabile locale alla funzione.

Quando viene chiamata tale funzione, si generano 2 pop-up: il primo restituisce il valore della variabile *i*, il secondo quello della variabile *j* e poi la funzione termina.
Successivamente sono stati inseriti altri due pop-up; il primo viene eseguito, restituendo il valore di *i*, il secondo pop-up, invece, non viene eseguito. Questo perché si è verificato un errore: si sta cercando di visualizzare il valore di *j* che, essendo una variabile locale alla funzione, è già stata distrutta al suo termine.
Questo errore interrompe l'interpretazione dello script e le istruzioni JavaScript successive al punto in cui si è verificato l'errore non vengono più eseguite; si passa infatti dall'interprete JS a quello HTML e viene visualizzato direttamente il messaggio "Fine della pagina".
Per poter eseguire tutto lo script senza errori è necessario dichiarare la variabile *j* al di fuori della funzione.

10.9.2 Funzioni e parametri

Una funzione può essere richiamata con meno parametri di quelli definiti. I parametri mancanti assumono valore *undefined* ma, esclusivamente in questo caso, il loro uso non genera errore perché è come se avessi dichiarato una variabile che non assume valore. Se, però, tali parametri vengono usati anche in altre istruzioni, il loro valore viene propagato e, nel caso di calcoli aritmetici, generano NaN.
Oppure una funzione può essere richiamata con più parametri di quelli definiti e tutti i parametri in eccesso sono ignorati.
Da ultimo è anche possibile definire una funzione senza parametri e poi accedere a tutti quelli che sono stati passati tramite il vettore arguments[ ], che prende in input i valori dalla linea di comando e contiene arguments.length valori distinti. Tramite questa proprietà, è quindi possibile scrivere funzioni generiche che prendano come parametri tutto ciò che si passa in input.

Esempio:

```
<script type="text/javascript">
function media()
// calcola la media aritmetica di tutti i numeri passati come parametri
{
var total = 0;
var n = arguments.length;
for (var i=0; i<n; i++)
total += arguments[i];
return (total / n);
}

// esempio di uso (media di tre numeri)
```

```
alert( media(11,12,16) );
</script>
```

La funzione media è definita senza parametri, per cui è possibile chiamarla con un numero a piacere di variabili in ingresso.

10.10 Oggetti predefiniti Javascript

10.10.1 L'oggetto Date

Quando si programmano dei siti web è molto importante tener presente la data e l'ora. Si pensi, ad esempio, a quei siti che si occupano di vendite on-line e che permettono di tenere in memoria gli eventuali acquisti per alcuni giorni; il sistema deve sapere quanto tempo è passato e deve poter memorizzare in qualche modo la data e l'ora della visita. Questo è possibile creando l'oggetto **Date**.

Scrivendo **new Date()** creo un oggetto che contiene la data e l'ora attuale, sul sistema che esegue lo script; se quindi lo script viene eseguito lato client, la data e l'ora saranno quelle del browser, se lo script viene eseguito lato server, la data e l'ora saranno quelle sul server. Se si utilizza il "new Date()" lato client, le informazioni memorizzate nell'oggetto sono quelle impostate sul pc, per cui se questo risulta mal configurato, data e ora risulteranno sbagliate. In alcuni casi risulta quindi più conveniente utilizzarlo lato server e poi trasmettere le informazioni al client.

Se invece specifico tra parentesi mese, giorno, anno ed opzionalmente ore, minuti, secondi (**new Date("Month day, year [HH:MM:SS)"**), creo un oggetto che contiene esattamente quella data e ora. I parametri devono essere scritti in lingua inglese (per esempio: *new Date("March 25, 2009 22:00:07")*).
Oppure è possibile indicare i parametri con valori numerici (**new Date(YYYY, MM, DD [, HH, MM, SS])**. Per esempio :*new Date(2009, 2, 25, 22, 00, 07)*, dove i mesi sono numerati da 0=Gennaio fino a 11=Dicembre. Se ore, minuti e secondi sono omessi, vengono considerati come zero.

La rappresentazione della data come stringa dipende dal Sistema Operativo su cui è eseguito lo script. Questo risulta essere rischioso, perché non sempre si è a conoscenza della versione (inglese, francese, ecc.) che l'utente sta utilizzando. Meglio, quindi, utilizzare metodi specifici per impostare individualmente i singoli elementi.

Metodi

Vengono presentati alcuni metodi utilizzabili sull'oggetto Date.

- **getDay( )** : restituisce il giorno della settimana (dove 0=Domenica,..., 6=Sabato);
- **setDay(giorno_ settimana)**: imposta il giorno della settimana;
- **getDate( )**: restituisce il giorno del mese, quindi da 1 a 31;
- **setDate(giorno_ mese)**: imposta il giorno del mese;
- **getMonth( )**: restituisce il numero del mese (dove 0=Gennaio,..., 11=Dicembre);

- **setMonth(mese_ num)**: imposta il mese;
- **getFullYear( ) / setFullYear (anno)**: per ottenere/ impostare le 4 cifre dell'anno;
- **getHours( ) / setHours (ora)**: per ottenere/ impostare l'ora (da 0 a 23);
- **getMinutes( ) / setMinutes(minuti)**: per ottenere / impostare i minuti (da 0 a 59);
- **getSeconds( ) / setSeconds(secondi)**: per ottenere/ impostare i secondi (da 0 a 59);

Tutti questi metodi sono disponibili anche nelle versione (**getUTC...( ) / setUTC...( )** riferita a UTC (Universal Time Ordinated), ovvero all'ora di Greenwich, non influenzata, quindi, dal fuso orario. Questa versione è utile nel caso si debbano confrontare due orari, ognuno con il proprio fuso orario.

L'oggetto Date presenta inoltre i metodi:

- **toString()** che converte la data in stringa secondo il formato nativo di JavaScript, ossia anglosassone. Per esempio se scriviamo:

  ```
  var d=new Date();
  var n=d.toString();
  ```

  il risultato di n sarà : *Thu Apr 19 2012 15:33:37 GMT+0200 (ora legale Europa occidentale)* ;

- **toLocaleString()** che converte la data in stringa secondo il formato locale del Paese in cui è stato impostato il Sistema Operativo che esegue lo script. Per esempio se scriviamo:

  ```
  var d=new Date();
  var n=d.toLocaleString();
  ```

  il risultato sarà: *giovedì 19 aprile 2012 15:33:37.*

10.10.2 L'oggetto Math

L'oggetto *Math* è un oggetto statico; non possiamo crearne uno nuovo e quindi esiste un unico oggetto *Math* in tutto JavaScript.

Proprietà

Questo oggetto ha una serie di proprietà **statiche** che possono essere richiamate per avere una serie di valori. Tali proprietà vengono chiamate sull'oggetto Math nella forma Math.proprietà e sono:

- `E`, costante di Eulero (vale circa 2.718);
- `LN2`, logaritmo naturale di 2 (circa 0.693);
- `LN10`, logaritmo naturale di 10 (circa 2.302);

- `LOG2E`, logaritmo in base 2 del numero di Eulero (circa 1.442);
- `LOG10E`, logaritmo in base 10 del numero di Eulero (circa 0.434);
- `PI`, pi greco (circa 3.14159);
- `SQRT1_ 2`, radice quadrata di 1/2 (circa 0.707);
- `SQRT2`, radice quadrata di 2 (circa 1.414).

Metodi

I metodi, che si invocano sull'oggetto Math nella forma Math.metodo(valore), sono anch'essi **statici**.

- `abs(x)`, restituisce il valore assoluto di x;
- `acos(x)`, restituisce l'arcocoseno di x, in radianti;
- `asin(x)`, restituisce l'arcoseno di x, in radianti;
- `atan(x)`, restituisce l'arcotangente di x come un valore numerico compreso tra-PI / 2 e PI / 2 radianti;
- `atan2(y,x)`, restituisce l'arcotangente del quoziente dei suoi argomenti;
- `ceil(x)`, restituisce x, arrotondato per eccesso al numero intero più vicino;
- `cos(x)`, restituisce il coseno di x (x è in radianti);
- `exp(x)`, restituisce il valore di eˆ x;
- `floor(x)`, restituisce x, arrotondato all'intero piu vicino più piccolo;
- `log(x)`, restituisce il logaritmo naturale di x;
- `max(x,y,z,...,n)`, restituisce il numero con più alto valore;
- `min(x,y,z,...,n)`, restituisce il numero con minor valore;
- `pow(x,y)`, restituisce il valore di x alla potenza y;
- `random(seed)`, restituisce un valore casuale compreso tra 0 e 1 escluso. Seed rappresenta il numero da cui partire per creare i numeri casuali;
- `round(x)`, arrotonda x all'intero più vicino;
- `sin(x)`, restituisce il seno di x (x è in radianti);
- `sqrt(x)`, restituisce la radice quadrata di x;
- `tan(x)`, restituisce la tangente dell'angolo x;

10.10.3 L'oggetto Number

L'oggetto Number rappresenta il wrapper per valori primitivi numerici.

Proprietà

Anche l'oggetto Number presenta una serie di proprietà statiche. Esse sono richiamate nella forma Number.proprietà:

- `MAX_VALUE`, restituisce il maggior numero possibile in JS;
- `MIN_VALUE`, restituisce il minor numero possibile in JS;
- `NEGATIVE_INFINITY`, rappresenta l'infinito negativo (restituito in caso di overflow);
- `POSITIVE_INFINITY`, rappresenta l'infinito positivo (restituito in caso di overflow);
- `NaN`, se il valore del parametro non può essere convertito in un numero.

Metodi

L'oggetto number viene utilizzato se si vuole avere una certa rappresentazione numerica, per questo sono disponibili alcuni metodi da utilizzare nella forma Number.metodo(valore):

- `tofixed(x)`, formatta un numero con un numero x di cifre dopo la virgola;
- `toexponential(x)`, converte il numero in formato esponenziale;
- `toprecision(num_cifre_significative)`, converte alla precisione indicata (eventualmente usando il formato esponenziale se necessario);

Tutti questi metodi arrotondano il risultato se il numero di partenza ha più cifre del necessario. Per esempio:

```
var num=5.126
alert(num.toprecision(3)) // visualizza 5.13
alert(num.toprecision(2)) // visualizza 5.1
```

Si noti che il numero originale non viene modificato, è soltanto la sua rappresentazione che viene cambiata.

Capitolo 11

Espressioni regolari in Javascript

11.1 Le espressioni regolari

Le espressioni regolari (spesso chimatae *RegExp*) sono uno strumento molto utile ed importante e definiscono un lessico (ovvero un"insieme" di parole lecite) specificando i caratteri e le sequenze ammissibili. I caratteri si distinguono in alfabetici (es. A, a), numerici (es. 1, 7) e segni di interpunzione (es. . : ;). Un esempio di sequenza ammissibile è "una sequenza di uno o più caratteri A" Le espressioni regolari utilizzano un formalismo "compatto" fanno cioè uso di metacaratteri. Per esempio se si scrive A^+ si intendono tutte le sequenze A, AA, AAA, AAAA, ldots. Questo linguaggio ha come fine quello di controllare l'input dell'utente ed è composto potenzialmente da un numero infinito di stringhe.

Il termine "espressioni regolari" è stato coniato dal matematico S. Kleene negli anni '50; questo formalismo viene creato all'interno della teoria dei linguaggi (che ha la stessa potenza espressiva di macchine a stati deterministiche). Le espressioni regolari non sono nate espressamente per il Web (che in quegli anni non esisteva ancora); queste ultime si rivelarono negli anni successivi molto comode per i linguaggi di programmazione. Le espressioni regolari sono molto usate in ambiente Unix. Gli utilizzi tipici di questo formalismo si riconoscono nella formulazione di potenti ricerche testuali (per esempio grep) e nell'automazione di editing testuale (per esempio sed).

11.1.1 Insiemi di caratteri

Per inidcare un insieme di caratteri si usa racchiuderli tra parentesi quadre ed è possibile specificare singoli caratteri o intervalli di caratteri adiacenti (indicando la prima e l'ultima lettera dell'iterallo separate dal segno meno). Per esempio l'intervallo `[A-Z]` indica tutte le lettere alfabetiche maiuscole mentre l'intervallo `[a-zABC]` riconosce una qualsiasi lettera alfabetica minuscola oppure A, B o C. L'intervallo `[a-c]` indicherà le lettere a, b e c; fa fede infatti l'ordine dei caratteri nel codice ASCII. Una variante dell'insieme precedente è data da `[^...]` che ha il significato di negazione: la sequenza viene utilizzata per escludere uno qualsiasi dei caratteri in parentesi. Per esempio `[^0-9]` indica qualsiasi carattere non numerico.

11.1.2 I metacaratteri

I metacaratteri sono caratteri speciali mediante i quali si associa un significato preciso ad un particolare carattere che permette di migliorare le tempistiche nella definzione dei "vincoli

all'utente" riguardanti i gruppi più comuni. Il simbolo chiamato escape cioè \ viene utilizzato per segnalare l'inizio di sequenze speciali ed evitare che questi ultimi siano interpretati come caratteri normali. Per esempio `\[` cerca il carattere `[`. Ci sono diversi simboli speciali che vengono usati per identificare un carattere. La sequenza `\d` indica un carattere numerico (equivalente a [0-9]); la sequenza `\D` indica, al contrario, un carattere non numerico (equivale a dire`[^0-9]`); i caratteri `\s` identificano lo "spazio" (blank, CR, LF, FF, HT, VT) e al contrario `\S` non e' equivalente a "spazio"; la sequenza `\w` indica tutti i caratteri alfanumerici o `_` (ossia equivale a scrivere [a-zA-Z0-9_]) e `\W` e' la sequenza che nega la precedente.

Esiste poi il carattere — cioe' l'or logico per esprimere un'alternativa tra due espressioni: la sintassi e' del tipo regexp1—regexp2. Per esempio A—B riconosce sia il carattere A che il carattere B. Esiste poi il metacarattere "." che indica un carattere qualsiasi: per esempio "R.E" riconosce una stringa di tre caratteri che inizi con R e finisca con E; saranno quindi accettate espressioni come "RaE", "RAE" o "RiE". Riassumendo le proprietà appena citate, se viene scritto "[Aa]b—[Bb]a" saranno accettate tutte le sequenze che cominciano con la A maiuscola o la a minuscola seguite dalla b minuscola e tutte le sequenze che cominciano con la B maiuscola o la b minuscola seguite dalla a minuscola.

11.1.3 Inizio e fine riga

La sintassi che viene utilizzata per identificare l'inizio della stringa è del tipo "^regexp". Per esempio "^buongiorno" riconoscerà l'espressione "buongiorno, eccomi qui!" e non accetterà "Dimmi almeno buongiorno". Un'applicazione di quest'espressione potrebbe essere il vincolo di mettere un numero all'inizio di una password. La sintassi che viene utilizzata per identificare la fine della stringa è invece del tipo "regexp$". Per esempio se viene scritto "200$" verrà accettata l'espressione "nell'anno 1200" ma non l'espressione "nell'anno 2000". Il seguente esempio mostra l'applicazione di entrambe le espressioni regolari per copiare un file eliminando le righe vuote:

```
grep -v "^$" file1  >  file2
```

Ricapitolando, se venisse scritto "^casa" si identificherebbero tutte le frasi che contengono la parola casa esattamente all'inizio della frase; si otterrebbe invece l'effetto contrario scrivendo "casa$". La frase "Casa è bella" è riconosciuta dalla prima espressione; "Bella casa" è riconosciuta dalla seconda; la frase "La mia casa è bella" invece non viene accettata.

11.1.4 I raggruppamenti

I raggruppamenti vengono utilizzati per aggregare espressioni e creare clausole complesse. La sintassi che viene utilizzata è del tipo:

`(` *regexp1 op regexp2* `)`

Un esempio potrebbe essere l'espressione "`pa(ss|zz)o`" che accetterà sia la parola `pazzo` che `passo`. Vengono poi usate le parentesi {} per specificare la numerosità: se esatta si inserirà solo {N}; se minima {Nmin}; se minima e massima {Nmin,Nmax}. La sintassi sarà regexp{ }. Per esempio l'espressione "`\d{2,4}`" riconosce numeri composti da almeno due cifre ed al massimo da quattro come "23", "655" o "4556". Non verranno accettate espressioni come "2", "23456" e "2+3".

11.1.5 Il numero delle occorrenze

Vengono utilizzati caratteri speciali per regolare il numero delle occorrenze. L'asterisco * indica che possono essere inserite zero o più occorrenze di un'espressione; per esempio (ab)* riconosce sequenze di "ab" di qualsiasi lunghezza come "ab", "abab", "ababab" ma anche "". Il simbolo + indica una o più occorrenze di un'espressione: per esempio (ab)+ riconosce sequenze di "ab" di qualsiasi lunghezza come "ab", "abab", "ababab" ma non accetta "". Il punto interrogativo ? indica zero o al piu' un'occorrenza: per esempio "Mar(i)?a" riconosce le sequenze "Mara", "Maria", "Mariangela" ma non "Mariiiiiiiiiiiia".

11.2 Le espressioni regolari in Javascript

Nel linguaggio Javascript le espressioni regolari possono essere create come sequenze letterali o come costruttore dell'oggetto `RegExp`. Nel primo caso l'espressione sarà valutata quando viene caricato lo script; questo metodo è più veloce se l'espressione rimane costante. Un esempio di sintassi nel caso di sequenze letterali è:

```
re=/ab+c/
```

Nel secondo caso invece viene utilizzata la forma:

```
re=new RegExp(ab+c'')
```

l'espressione in questo caso viene valutata a runtime. E' necessario porre attenzione al carattere backslash; sarà necessario scrivere:

```
re=/\d{2}/
```

oppure

```
re=new RegExp("\\d{2}").
```

11.2.1 Flag delle espressioni in JS

Nella creazione della regexp è possibile inserire dei flag. A seconda del caso si userà la sintassi re=/pattern/flags o re=new RegExp("pattern", "flags"). I flag possibili sono tre: il flag "i" permette di ignorare la differenza tra caratteri minuscoli e maiuscoli (rende cioè case-insensitive); il flag "g" non si ferma al primo match ma prosegue la ricerca su tutta la stringa (detto global); il flag "m" controlla se la stringa contiene terminatori di linea e se ciò avviene considera ogni riga una stringa diversa ai fini dei caratteri `$` e `^` (viene anche detto multiline). Per esempio:

```
re = /ciao/i
```

11.2.2 Metodi Javascript per espressioni regolari (base e avanzati)

Il metodo string.search(regexp) restituisce l'indice iniziale della (prima) stringa trovata oppure -1. Il metodo regexp.test(stringa) restituisce true se all'interno della stringa compare l'espressione, false altrimenti. Un esempio di codice è:

```
var s = "ciao, mamma"
var re = /\W/
n = s.search(re) //restituisce 4
x = re.test(s)   //restituisce true
```

Le espressioni regolari possono essere usate per identificare una o più sottostringhe (e cambiarle o salvarle in un array). Il metodo string.match(regexp) restituisce le stringhe trovate oppure null; il metodo string.replace(regexp, new) restituisce la stringa di partenza con new al posto delle parti selezionate dall'espressione regolare; il metodo string.split(regexp) restituisce le stringhe identificate in base alla regexp; il metodo regexp.exec(string), infine, restituisce le stringhe identificate in base alla regexp.

11.2.3 Validazione dei dati di un form

```
function verifica(d)
{
   var expr=/^\d{2} - \d{2} - \d{4}$/
   if (expr.test(d))
           window.alert(d+":formato corretto")
   else
           window.alert(d+":formato errato")
}

<form>
        data (gg-mm-aaaa) <input type="text" name="data">
        <input type="button" value="verifica"
   onclick="verifica(data.value)">
</form>
```

L'esempio riguarda il controllo del formato della data al momento dell'inserimento. Essa infatti dovrà avere due cifre intere per il giorno, due per il mese e quattro per l'anno.

Un altro esempio applicativo può riguardare l'inserimento della password: si desidera che quest'ultima sia alfanumerica di minimo 8 caratteri e di massimo 16; inoltre è buona norma che ci sia almeno una maiuscola, una minuscola e una cifra. Viene quindi creata una funzione che restituisca true se la password è corretta, false altrimenti.

```
function is_pwd_OK(p)
{
return
  (
  /^[a-zA-Z0-9]{8,16}$/.test(p) &&
  /[A-Z]/.test(p) &&
  /[a-z]/.test(p) &&
  /[0-9]/.test(p)
  )
}
```

Se invece si vuole verificare il formato di un numero naturale (ed eventualmente avere come risposta il numero senza gli eventuali punti (separatori delle migliaia)) si può utilizzare questa funzione:

```
function n_naturale(x)
{
  re=/^\d{1,3}(\.?\d{3})*$/
  if (re.test(x))
        return x.replace(/\./g,"")
  else
        return -1
}
```

Il test dei programmi è un arte ed è anche una scienza. Questo serve per testare sia i casi "positivi" che quelli "negativi".

E' opportuno che i controlli inseriti mediante le espressioni regolari vengano fatti sia lato client che lato server. La replicazione di questi ultimi viene fatta per prevenire attacchi di eventuali hacker.

Capitolo 12

I form HTML ed il loro uso nel web dinamico

12.1 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 client; 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 ciascun 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).

12.2 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`.

12.2.1 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.

12.2.2 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 12.1: esempio di un form.

12.2.3 Esempio di form

La figura 12.1 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.

12.2.4 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).

12.2.5 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

```
<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 12.2: 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 12.3: esempio di utilizzo di `radio`.

anche nessuna). La figura 12.2 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 12.3 contiene un semplice esempio di creazione di un controllo a scelta multipla con `radio`.

12.2.6 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 suo valore 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.

12.3 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: indicando la gerarchia completa dei nomi secondo il modello DOM o estraendo direttamente l'elemento tramite l'ID univoco. Esempi:

```
alert (document.f1.frt.value)  <!-- accesso tramite gerarchia DOM --->
alert (document.getElementById("frt").value)  <!-- accesso tramite ID -->
```

12.4 Validazione client-side dei valori 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)

```
<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 12.4: Esempio di codice per la validazione di un form.

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 12.4 riporta un esempio di codice per la validazione di un form.

12.4.1 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.

12.5 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.

12.5.1 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 URI, 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 12.5 mostra i caratteri US-ASCII stampabili ed i rispettivi codici esadecimali (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>
```

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, verrebbe creata la seguente stringa:

```
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 PHP ed usiamo le variabili `$_GET`, `$_POST` o `$_REQUEST`, i dati che leggeremo saranno già trasformati lato server e ritroveremo quindi i caratteri originali.

codice	*carattere*	*codice*	*carattere*	*codice*	*carattere*
20	*spazio*	40	@	60	‘
21	!	41	A	61	a
22	"	42	B	62	b
23	#	43	C	63	c
24	$	44	D	64	d
25	%	45	E	65	e
26	&	46	F	66	f
27	’	47	G	67	g
28	(	48	H	68	h
29	)	49	I	69	i
2A	*	4A	J	6A	j
2B	+	4B	K	6B	k
2C	,	4C	L	6C	l
2D	-	4D	M	6D	m
2E	.	4E	N	6E	n
2F	/	4F	O	6F	o
30	0	50	P	70	p
31	1	51	Q	71	q
32	2	52	R	72	r
33	3	53	S	73	s
34	4	54	T	74	t
35	5	55	U	75	u
36	6	56	V	76	v
37	7	57	W	77	w
38	8	58	X	78	x
39	9	59	Y	79	y
3A	:	5A	Z	7A	z
3B	;	5B	[	7B	{
3C	<	5C	\	7C	|
3D	=	5D	]	7D	}
3E	>	5E	^	7E	~
3F	?	5F	_	7F	DEL

Figura 12.5: Caratteri stampabili US-ASCII.

```
<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 12.6: Form per prenotazione aula.

12.5.2 Esempio trasmissione di un form con GET

Consideriamo come esempio un form di prenotazione (Figura 12.6) 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).

12.5.3 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.

12.5.4 Esempio trasmissione di un form con POST

Consideriamo lo stesso form già visto in precedenza (Figura 12.6), 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.

12.5.5 Esempio trasmissione di un form con POST e multipart

Consideriamo lo stesso form già visto in precedenza (Figura 12.6), 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).

12.6 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)
```

12.7 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.

12.8 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 13

Il protocollo HTTP

HTTP (HyperText Transfer Protocol) è un protocollo client-server ideato per la richiesta e la trasmissione di pagine HTML, oggi è utilizzato per lo scambio di risorse web in generale. HTTP/0.9 è la prima versione ed ha avuto un uso esclusivamente sperimentale ed estremamente limitato. HTTP/1.0 è stata la prima ad avere larga diffusione e ad oggi è ancora molto usata. Tutti i sistemi al mondo oggi sono in grado di gestire questo protocollo. In realtà tale versione sarà superata a breve da HTTP/1.1, che ha l'obiettivo di migliorare l'efficienza del web.

Cosa significa migliorare l'efficienza del web? Il protocollo HTTP/1.0 era stato pensato nell'ottica di non sovraccaricare troppi i server; oggi invece, la rete è diventata il vero collo di bottiglia, infatti HTTP 1.1 è stato sviluppato con l'intento di ridurre il carico sulla rete, ad esempio attraverso una gestione più oculata della cache.

13.1 Il protocollo HTTP/1.0

Documentato nell'RFC-1945, è un servizio offerto di default tramite il protocollo TCP sulla porta 80, ed è un protocollo stateless ovvero ne il client ne il server tengono memoria di come sta andando la connessione. In particolare HTTP/1.0 è stato sviluppato avendo ben chiaro che i server sono pochi e i client sono tanti, quindi tipicamente il server non può ricordarsi di tutti i client, perciò si è pensato di limitare al massimo le informazioni memorizzate. Inoltre è possibile che il server non possa rispondere in tempo alle richieste del client e come conseguenza di ciò si accetta che il client possa chiudere la connessione prima di aver ricevuto la risposta o parte di essa. Escluso questo caso particolare in cui è il client a prendere la decisione, generalmente è il server a chiudere il canale: il client fa la domanda, il server manda la risposta e poi è il server a chiudere il canale.

In HTTP/1.0 i dati vengono trasmessi interamente a 8 bit, infatti è un protocollo 8-bit clean, ovvero qualsiasi file viene fatto passare senza alcuna manipolazione, neppure nei passaggi intermedi, al contrario di protocolli come SMTP. Inoltre è lecito trasmettere anche le lettere accentate, ma visto che la parte alta della codifica ASCII è diversa da lingua a lingua, è anche necessario definire quale linguaggio si sta utilizzando: di default si ha la versione europea, la ISO-8859-1, il cosiddetto Latin-1,questo perché il protocollo è stato inventato in Europa.

13.1.1 Connessione stateless

Quando il browser decide di voler comunicare con il server, il primo passo consiste nel creare una connessione, ovvero viene aperto un canale TCP verso la porta 80 del server facendo il cosiddetto processo di three-way-handshake per aprire la connessione TCP. All'interno di questa connessione viene mandata quella che si chiama request HTTP. A fronte di questa richiesta il server invia una risposta, tipicamente sotto la forma di codice HTML, ma può mandare anche codice javascript, che dovrà essere interpretato dal client. Quando il server ha terminato di inviare la risposta chiude la connessione, tramite il four-way-hand-shake, anche se il client avrebbe bisogno di altri file. Questo permette di liberare la risorsa e renderla disponibile per altri client.

Visto che il canale viene chiuso non appena è stata fornita una singola risposta, il collegamento tra una risorsa e quella successiva non avviene automaticamente, quindi dobbiamo capire come tenere traccia del fatto che un browser abbia già fatto certe domande o visitato certe pagine. Il motivo per cui HTTP/1.0 è stato sviluppato in questo modo è quello diminuire al massimo il carico sul server e consentire una gestione più equa delle risorse. Se il client, dopo aver ricevuto una risposta, necessita di altro, deve nuovamente fare un richiesta. Se non ciò avvenisse il client che riceve una risposta dal server e si dimentica di chiedere immediatamente quanto gli serve, tiene occupato inutilmente il server che può tenere aperti simultaneamente solo un numero finito di canali di rete (che dipende dalla quantità di RAM presente nel server). Invece quello che accade è che, una volta ricevuta la risposta, se il client necessita di altro deve rimettersi in competizione con tutti gli altri browser, permettendo una fruizione più equa del servizio (un esempio di servizio non fornito equamente è quello dell'ufficio postale in cui ho solo una persona davanti ma ha tantissimi bollettini da pagare, essa monopolizza il servizio).

13.1.2 URL

Per identificare una risorsa vengono utilizzate le cosiddette URL (Uniform Resource Locator). Essa permette di identificare in modo univoco una qualunque risorsa presente in rete.

schema `://` *user* `:` *password* `@` *host* `:` *porta* `/` *path* `#` *àncora*

Lo schema indica il metodo con il quale possiamo accedere a tale risorsa. Molto spesso coincide con un protocollo (ad esempio `http:` o `ftp:`) oppure con `file:` che sta a indicare il fatto che per accedere a questa risorsa è necessario andare sul file system.

Dopo `://` è possibile specificare username e password nel caso in cui la risorsa sia protetta da username e password non inseribile all'interno di un form ma direttamente a livello http. In realtà oggi la maggior parte delle risorse non sono protette in questo modo, quindi questo metodo è poco usato.

Dopo la `@` si deve indicare l'host ovvero il nome del fruitore della risorsa e dopo : la porta (di default 80). Uno `/` separa la parte di rete da quella locale, ovvero l'indicazione della serie di cartelle logiche da attraversare per giungere alla risorsa che interessa.

E' infine possibile specificare dopo `#` il punto in cui si vuole aprire la pagina, se essa non è indicata la risorsa è aperta dall'inizio, se indicata è invece possibile accedere ad essa in un punto specificato.

Esiste anche la possibilità di usare schemi irregolari che non seguono questa sintassi ad esempio news, ormai poco utilizzato, oppure per la spedizione di messaggi di posta elettronica

direttamente dalla pagina web `mailto` seguito da `:` e dall'indirizzo a cui inviarlo. Affinchè questo funzioni, però, il computer deve sapere come inviare la posta, perciò è necessario che abbia un MUA.

Questa è la definizione base presente nell'RFC-1738, ma le URL continuano ad evolversi per altri protocolli, ad esempio LDAP definito nell'RFC-1959 e RFC-2255 è un protocollo per trattare i cosiddetti directory service (ovvero dei server) ed è una sorta di pagine gialle che data una chiave forniscono tutte le informazioni relative a quella chiave. IMAP, definito nell'RFC-2192, è usato invece per inviare posta elettronica e NFS per trattare dischi in rete (definito nell'RFC-2224) .

13.1.3 URN e URI

Le URL puntano a tutti gli effetti a degli elementi fisici, invece sarebbe molto meglio poter dare l'indirizzo logico di una risorsa, ad esempio l'homepage, così se un giorno un utente volesse cambiare la cartella del file non dovrebbe cambiare URL e informare tutti gli interessati. Per evitare ciò è stato definito il concetto di URN (Uniform Resource Name), l'obiettivo è quello di usare dei nomi logici per identificare le risorse (ad esempio "il server del professor Lioy"). Il problema che non è ancora stato risolto è creare una corrispondenza tra nome logico e risorsa corrispondente.

Le URN sono molto utilizzate per definire elementi nel linguaggio XML, l'uso nel web è ancora molto scarso.

Inoltre è sato definito un nuovo identificatore, la URI (Uniform Resource Identifier), una generalizzazione che autorizza ad usare sia una URL che una URN, ed è definita nell'RFC-1959. Ad oggi il termine URI coincide praticamente con URL per quanto riguarda i browser.

13.1.4 Comandi

Una volta scelto il canale, su di esso vengono inviati dei comandi con caratteri ASCII, ogni comando è una riga, per distinguere una riga dalla successiva si usa la combinazione di `CR+LF`. I dati che vengono trasmessi come risposta ad un comando possono essere anche binari, ad esempio si può inviare un'immagine o un eseguibile. Il messaggio che ci si scambia tra client e server si compone di header e body. L'header è costituita di 8 righe, ogni riga inizia con una determinata `keyword:` più il valore ad essa assegnato. La distinzione tra header e body è ottenuta attraverso una riga vuota che contiene solo `CR+LF`. La prima riga dell'header non contiene una keyword, ma deve contenere obbligatoriamente un metodo tra GET, HEAD E POST.

Richiesta

Con `GET /` *URI* `/` *versione http* chiedo la risorsa associata a una URI. Con versione http si intende quella che il richiedente è in grado di supportare. La risposta del server sarà il contenuto di quella pagina.

Con `HEAD /` *URI* `/` *versione http*, richiedo la risorsa, ma nella risposta non voglio tutta la pagina, bensì soltanto l'header. Questo metodo è utile perché potrei non aver bisogno di quel file essendo già esso presente nella mia cache e voglio perciò valutare soltanto se la copia del file richiesto sia sempre la stessa o se magari il file sia stato aggiornato e solo al quel punto richiederne la versione completa. Il rischio è quello di farsi mandare l'header per

poi rendersi conto che era necessario ottenere anche il body, cosa che comporta una perdita di tempo, perchè dopo la richiesta di HEAD il canale viene chiuso e si dovrà riaprire per fare la richiesta GET.

Con `POST /` *URI* `/` *versione http*, posso inviare al server delle informazioni che dovranno essere elaborate dalla URI indicata. Perciò tale URI dovrebbe essere un programma attivo, perchè dovrà fare dei calcoli e restituirmi una risposta. Viene usato quando sul browser abbiamo un form e dopo averlo compilato devo inviare i dati inseriti al server. Le keyword presenti sono:

- `Content-type`: indica come sono scritti i dati che sto inviando.
- `Content-length`: indica la lunghezza di tali dati.

Risposta

La risposta contiene nella prima riga la versione HTTP scelta dal server, ovvero la più alta in comune tra client e server. Viene poi inserito un codice di stato, ovvero un numero che indica brevemente come è andata a finire la richiesta. In più opzionalmente è possibile che ci sia anche un commento testuale che può aiutare il lettore a capire il significato del codice. La URI è il path della risorsa richiesta, non bisogna riscrivere tutto ma solo la parte che specifica il percorso all'interno del server (GET è già un pezzo del protocollo http quindi non può essere seguito da `http://`). Invece è necessaria l'intera URI se esiste un proxy ovvero un elemento intermedio nella comunicazione tra client e server.

Vediamo nella prima immagine un esempio pratico. Il client fa una richiesta di GET e chiede, tramite l'URI indicata, che gli venga inviato il file di default che si trova in tale server. Sta usando il protocollo HTTP/1.0. Il server risponde che l'operazione ha avuto successo, ma prima della risposta vera e propria fornirà una serie di campi:

- `Date:` indica data e ora di generazione del messaggio di risposta.
- `Server:` indica il software utilizzato dal server.
- `Content-Type:` indica se il body è scritto in formato testo ma html. Ecco che si ritrova la sintassi del MIME.
- `Content-length:` indica il numero di caratteri di cui è formato il body.
- `Last-modified:` indica l'ultima data di modifica del contenuto.

Nella seconda immagine è presente un esempio di richiesta HEAD. L'header è uguale, ma non è presente il body. Il client andrà a verificare soltanto il campo `Last-modified:` e se corrisponde al campo della versione che ha in cache andrà a prendersi quella evitando di fare la richiesta di GET, necessaria solo se la data di modifica è più recente.

La terza immagine corrisponde alla richiesta POST. La URL è diversa dalle due precedenti, perchè tipicamente con un POST non si richiede una pagina HTML statica, bensì si richiede una pagina dinamica, interrogando ad esempio una programma CGI o PHP, che, dati certi input, restituisce una risposta. Nella richiesta POST ci sono degli header non visti nei metodi precedenti, che indicano il formato del testo che si sta per mandare e la lunghezza. Dopo il solito carattere di terminazione dell'header ci sarà il BODY (unico caso in cui è presente anche nella richiesta) che conterrà i dati che si vogliono mandare. La risposta sarà generata come prima , dandomi però conferma che i dati sono stati inviati; manca inoltre `content-length` perché il server non può prevedere la lunghezza della risposta dato che questa sarà fornita da un programma attivato dopo l'header.

13.1.5 Test manuali

Per capire bene il protocollo è importante fare dei test manuali tramite un programma che serve ad emulare un protocollo manualmente, TELNET, presente nelle linee di comando dei sistemi operativi.

In Windows i comandi necessari saranno:

- `telnet` per far comparire il prompt Microsoft telnet.
- `set localecho` significa che ciò che viene trasmesso è anche visibile anche sullo schermo, perché normalmente il programma telnet non mostra ciò che si sta scrivendo.
- `set crlf` indica che il tasto return deve trasmettere la coppia di caratteri `CR+LF`.
- `set logging` significa che si vuole tenere traccia di quello che capita tra client e server, per vedere come evovono le cose.
- `set logfile mylog.txt` mette tale traccia nel file `mylog.txt`.
- `open www.polito.it 80` apre il collegamento al server indicato (possiamo mettere il nome o l'indirizzo IP) verso la sua porta 80.
- `GET / HTTP/1.0<enter>` il canale è aperto, si fa una richiesta HTTP, e si batte `<enter>` per indicare che è finito l'header. Se lo si batte per una seconda volta la richiesta viene inviata .

13.1.6 Codici di stato

Tutte le risposte contengono un codice di stato di 3 cifre, di cui la prima fornisce il major status dell'azione richiesta mentre la seconda e terza affinano tale stato. Nell'immagine sono presenti tutti i codici di stato standard.

1. Informational, informativo.
2. Success, operazione eseguita con successo.
   - `200 OK`
   - `201 Created` se il client ha chiesto di creare qualcosa sul server
   - `202 Accepted`
   - `203 No Content` indica che la risposta esiste ma non contiene niente
3. Redirection, il server mi indica un altro indirizzo a cui chiedere il file, perché lui non lo ha, ma sa dove si trova e devia quindi la richiesta verso un altro server.
   - `301 Moved permanently` il file chiesto si troverà sempre lì dove è stato indicato.
   - `302 Moved temporarily` la pagina chiesta non è qui solo temporaneamente.
   - `304 Not modified` il server non ha più la pagina ma mi dice che non è cambiata, così se la ho in cache la prendo da li.
4. Client error, è il client che ha commesso l'errore.

- `400 Bad request` la sintassi è sbagliata.
- `401 Unauthorized` il client non ha il permesso per accedere a quella risorsa.
- `402 Forbidden` il client sta cercando di eseguire una azione vietata ad esempio sta provando a fare POST su una pagina HTML
- `403 Not found` la pagina richiesta non è stata trovata.

5. Server error, il server ha problemi.
   - `500 Internal server error`
   - `501 Not implemented` il metodo che ho richiesto non è implementato dal server, ad esempio alcuni permettono solo GET e non POST.
   - `502 Bad gateway` quando si cerca di utilizzare un certo gateway per arrivare al server, ma il server non gli riconosce i permessi.
   - `504 Service unavailable` il servizio non è al momento disponibile.

Nell'immagine sono presenti tutti i codici di stato standard.

13.1.7 Redirect

Se richiediamo una risorsa con una certa URI ed essa non è presente a tale indirizzo, il server può fornire l'indicazione della nuova URI tramite una risposta con codice 3, che nell'header di risposta avrà un campo `location:` *nuova URI*. Leggendo l'header location il browser può connettersi automaticamente alla nuova URI se il metodo con sui stava cercando di accedere era GET o HEAD, ovvero un metodo con cui stava cercando di leggere delle informazioni. Gli è vietato farlo se il metodo è POST, perché significa il client stava inviando dei dati al server, che potrebbero essere segreti (ad esempio il codice della carta di credito) e se il server gli dice di inviare tali dati ad un nuova URI, non è ritenuto essere sicuro perché non si è a conoscenza di informazioni che ne garantiscano la sicurezza. L'utente dovrà quindi decidere se inviare o no i dati alla nuova URI. Per supportare i browser più vecchi che non conoscono redirect si consiglia di non inviare soltanto una location ma anche un body nella risposta con una pagina HTML che spieghi la situazione.

13.1.8 Header

Header generali utilizzabili sia nelle richieste che nelle risposte:

- `Date:` *http-date* contiene la data di invio della richiesta o risposta.
- `Pragma:` *no-cache* fornisce un'indicazione pragmatica, l'unica indicazione che può essere fornita in http 1.0 è no-cache. Se lo dice il server al client significa: non salvare il file nella cache; può essere utile nel caso in cui i dati che mi ha inviato il server esauriscano la loro utilità entro breve (indicazioni dell'ora per esempio) oppure nel caso in cui la risposta contenga dati riservati (ad esempio un estratto conto) che è meglio siano solo visualizzati senza essere salvati. Se lo chiede il client nella richiesta, si vuole il file originale contenuto nel server, non una copia di esso presente nella cache di un proxy o di un gateway sulla strada.

Header di richiesta:

- `Authorization:` *credentials* è la richiesta delle credenziali di autenticazione.
- `From:` *user-agent-mailbox* serve per indicare chi è l'utente che sta inviando dati a quel server, oggi è poco usato per problemi di spam. Utile al server che in caso di problemi poteva contattare l'utente.
- `If-Modified-Since:` *http-date* permette di evitare il metodo HEAD se abbiamo già il file in cache. Si richiede il file con un GET solo se esso è diverso dalla versione che ho in cache, ovvero a condizione che sia stato modificato dalla data nella quale ho ricevuto la mia copia. Se non è cambiata il messaggio di risposta avrà solo un header e il codice di stato 304.
- `Referer:` *URI* viene utilizzato quando c'è un redirect, indica chi è che ha reindirizzato l'utente alla nuova URI, è utile per esempio nell'ambito pubblicitario, in modo da tenere traccia di chi è stato a pubblicizzare il mio sito.
- `User-Agent:` *product* indica il tipo di browser che sto utilizzando per fare la richiesta, utile per fare statistiche.

Header di risposta:

- `Location:` *absolute-URI* serve in unione con un codice 3 per fare un redirect.
- `Server:` *product* indica il software utilizzato dal server.
- `WWW-Authenticate:` *challenge* serve per questioni di sicurezza, è una sorta di indovinello a cui il browser deve rispondere per poter accedere al server.

Header riguardanti il body:

- `Allow:` *method* indica quali metodi possono essere utilizzati.
- `Content-Encoding: x-gzip | x-compress` è possibile trasmettere il body compresso e si può indicare anche il tipo di compressione.
- `Content-Length:` *length* indica la lunghezza in byte del body inclusi i caratteri `CR+LF`.
- `Content-Type:` *MIME-media-type* indica il tipo di contenuto seguendo la sintassi MIME.
- `Expires:` *http-date* indica la data oltre la quale la risposta che sto inviando non ha più valore. Ciò è utile in particolare per le pagine dinamiche.
- `Last-Modified:` *http-date* indica la data di ultima modifica del body.

13.1.9 Indicazioni di tempo in HTTP

Ci sono tre possibili formati:

1. RFC-822, ad esempio `Sun, 06 Nov 1994 08:49:37 GMT`;
2. RFC-850, ad esempio `Sunday, 06-Nov-94 08:49:37 GMT`;
3. asctime, ad esempio `Sun Nov 6 08:49:37 1994`.

In generale tutti i componenti di un sistema HTTP devono eesere in grado di accettare tutti questi formati ma si consiglia di generare sempre e solo il formato RFC-822 perché è l'unico a lunghezza fissa ed è quindi più facile da trattare.

Si noti oltre che l'ora è sempre indicata includendo il fuso orario. Nonostante sia possibile indicare qualunque fuso, si consiglia di indicare sempre l'ora con riferimento al meridiano fondamentale di Greenwich, ossia esprimere sempre l'ora in formato GMT (Greenwich Mean Time) lasciando ai vari componenti la traduzione nel fuso orario locale (es. del browser o del server).

13.2 Il protocollo HTTP 1.1

Di solito è consigliabile aggiornare la versione di un protocollo con molta cautela perché si rischia di introdurre problemi di compatibilità, dunque andrebbe fatto solo in casi di effettivo bisogno. La versione 1.1 di HTTP è stata creata per risolvere una serie di problemi presenti nella versione 1.0 e per migliorare l'efficienza del web.

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 e non essendo possibile inserire l'header `Content-Length:`, c'è il rischio di un troncamento dei dati nel caso si verifichino errori o problemi di rete perché questo non è distinguibile dalla chiusura del canale da parte del server; inoltre in caso di errore nella trasmissione, dobbiamo riscaricare il file dall'inizio mentre, per ottimizzare il traffico di rete, bisognerebbe poter riprendere il file dal punto in cui si è interrotto.

HTTP/1.0 era stato pensato per il download di pagine singole e non è efficiente nella trasmissione di più file, perché essendo un protocollo stateless comporta il bisogno di aprire una nuova connessione TCP per ogni oggetto che dobbiamo scaricare. L'apertura di una connessione TCP introduce un tempo di TCP setup (three-way-handshake) e una perdita di tempo dovuta all'algoritmo di slow start (partenza lenta quindi velocità limitata), inoltre la chiusura del canale comporta la perdita di tempo dovuta al four-way-handshake e alla perdita delle informazioni relative alla finestra TCP. Alla prossima apertura si dovrà ricominciare con lo slow start da capo. Inoltre la gestione della cache era elementare (solo on/off), potevo infatti solamente decidere se attivarla o no (`pragma:` *no cache*).

13.2.1 Migliorie di HTTP/1.1

Oggi il focus di chi si occupa di migliorare il protocollo HTTP è la rete, non più il sovraccarico del server, dunque le soluzioni sviluppate devono andare in questa direzione. Se con fatica si è riusciti ad aprire una connessione TCP e a portare la finestra ad una dimensione decente, non si vuole ricominciare con lo slow start appena si inizia a scaricare un altro file. Anche i programmatori web devo andare in questa direzione, ad esempio ricordandosi di far validare un form a livello client, per evitare di dover mandare al server troppe richieste.

La descrizione di HTTP/1.1 è presente nell'RFC-2616. Nell'HTTP 1.1 le connessioni non vengono chiuse ogni volta ma al contrario, può essere aperta 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. Se una pagina è presente in tante lingue bisogna essere in grado di adattarsi in diretta al client. 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 resa più raffinata, per cercare di ridurre il traffico sulla rete. Si possono scegliere varie modalità e sono previste gerarchie di proxy fra il client e l'origin server. Questo ultimo punto però, pur essendo possibile nel protocollo, non è stato implementato dagli operatori di telecomunicazione, perchè essi hanno interesse che gli utenti facciano più traffico e siano portati a richiedere upgrade di banda e quindi a pagare di più per una connessione più veloce. Neppure le aziende si impegnano a implemementare proxy, spesso per ignoranza o per evitare di avere costi aggiuntivi. HTTP/1.1 offre tutti gli strumenti per mettere in piedi proxy che permettano di scaricare la rete, purtroppo però, fino ad ora non sono stati sfruttati abbastanza. Inoltre, anche in questo caso, i programmatori devono porre maggiore attenzione al sovraccarico della rete, evitando di scrivere tutto con pagine dinamiche: se non è necessaria dinamicità, é molto meglio creare pagine statiche, che possono essere salvate nella cache e sono migliori dal punto di vista della sicurezza.

E' stato introdotto il supporto di server virtuali, cioè la possibilità di associare ad un unico indirizzo IP diversi server logici. Questo ha reso possibile il *web hosting*, cioè la tendenza ad affidare la gestione del proprio sito a grosse società come Amazon, che tramite un unico grosso server sono in grado di ospitare tanti siti web, tutti con un proprio indirizzo logico diverso.

Sono stati aggiunti i metodi PUT, DELETE, TRACE, OPTIONS e CONNECT ed è stato introdotto un nuovo metodo di autenticazione, basato su digest, direttamente a livello di trasporto dei dati, che permette di far sapere al server che si conosce la password senza dovergliela inviare.

13.2.2 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 web logici. Questo è dovuto al fatto che i server sono ottenibili tramite alias nel DNS (Cname o nome canonico) e per distinguere i diversi server il client deve indicare il virtual host desiderato tramite il suo FQDN (Fully Qualified Domain Name) nella richiesta HTTP.

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

Per comprendere meglio lo scopo ed il funzionamento dei virtual host 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.

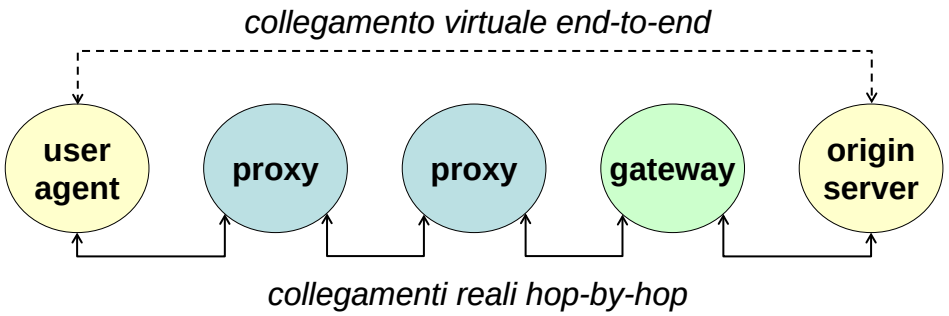

Figura 13.1: Collegamenti fra User Agent ed Origin Server.

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.

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`, perchè se non lo specificassi non saprei alla radice di quale server virtuale io mi stia riferendo.

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. La richiesta non è più ambigua.

13.2.3 Connessioni persistenti

Le connessioni persistenti permettono la trasmissione di richiesta e risposta su un unico canale TCP, eliminando i problemi relativi al setup del canale TCP e dello slow start. Il canale verrà chiuso solamente quando l'utente o il server inseriranno l'header `Connection: close` che in pratica serve a tornare ad una connessione di tipo HTTP/1.0.

Questo header non è più 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. Dunque l'istruzione `Connection: close` non è relativa al singolo percorso, ma ad unico collegamento hop-by-hop. 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 16.13 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 conviene perché il gateway è l'interfaccia pubblica, ma l'origin server parla sempre con lo stesso gateway, invece un canale non persistente fra gateway e proxy potrebbe servire a evitare un sovraccarico del gateway in quanto i canali TCP non utilizzati verrebbero chiusi. Anche tenere aperto il canale tra browser e il proxy aiuta a risparmiare tempo.

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 HTTP 1.1 come protocollo e `www.polito.it` come host. Il server risponde inviando la pagina richiesta ma, al contrario di quanto capiterebbe con HTTP 1.0, non chiude il canale. A questo punto, dopo aver ricevuto la risposta, il client 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). Si possono vedere tutti questi passaggi nell'immagine.

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

Le connessioni persistenti servono quando sto prendendo pagine una dopo l'altra e non quando passa tanto tempo tra una richiesta e l'altra. Infatti generalmente i server hanno un timeout applicativo; un canale che rimane inattivo può essere chiuso dal server anche senza la ricezione dell'header connection close, se si rende conto che è aperto e inutilizzato da troppo tempo. Il timeout è utilizzato per non tenere impegnata una risorsa di rete senza comunicazioni per troppo tempo.

Vantaggi e svantaggi delle connessioni persistenti

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

- l'overhead di apertura e di chiusura del canale è minore perché se ne effettua un numero minore; da notare che l'operazione 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 ed è dunque molto peggio in termini di tempo. 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 pipelining delle richieste, cioè inviare le richieste in modo sequenziale, senza avere ancora ricevuto le risposte alle domande precedenti; 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'UA all'OS lavorino di meno. Questo comporta un risparmio di 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 a HTTP 1.0.

Le connessioni persistenti però portano anche alcuni svantaggi, tra cui:

- maggior sovraccarico del server, perché l'idea base delle connessioni persistenti è opposta a quella che era il punto di forza di HTTP 1.0, dove il server chiude sempre il canale per permettere a tutti i client di contattarlo ed evitare che alcuni lo monopolizzassero; siccome, come è stato detto, il problema maggiore oggi è nella rete, questo svantaggio è stato accettato.

- tenendo aperto un canale inutilizzato si rischia una possibile saturazione delle risorse (le connessioni persistenti potrebbero essere sfruttare per quello che in sicurezza è chiamato un attacco denial-of-service[1]).

13.2.4 Come funziona il Pipeline

Il pipeline è la possibilità di inviare più richieste senza attendere le risposte; questo ottimizza TCP ed è molto utile quando si richiedono in un colpo solo più elementi (di una stessa risorsa). Siccome dopo poco la mia finestra TCP ha la dimensione di qualche KB, se mando soltanto una richiesta GET alla volta sto sprecando lo spazio a mia disposizione e sto mandando un file praticamente vuoto. Fare pipeline vuol dire mettere in un unico segmento TCP tutte le richieste che devo fare. 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.

13.2.5 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 e 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 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

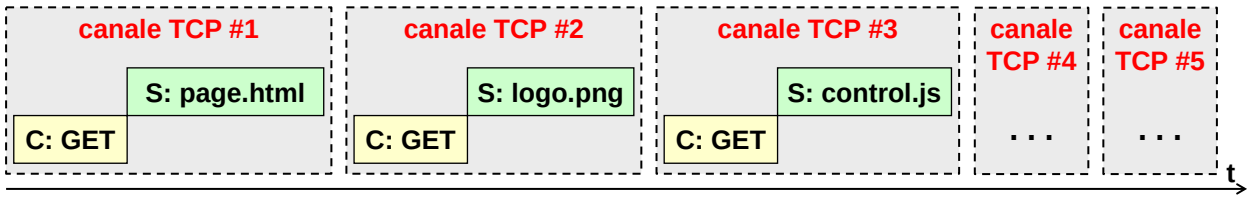

Figura 13.2: Diagramma temporale del trasferimento con HTTP/1.0.

La figura 13.2 illustra il caso in cui sia 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).

Nella figura 13.3 è stato usato HTTP/1.1, ma senza il pipeline: 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.

Nella figura 13.4 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

[1] (negazione-del-servizio) E' un attacco che non mira a rubare dati ma a rendere il servizio inutilizzabile.
[2] Per esigenze di impaginazione nella figura sono mostrati solo i primi tre canali TCP.

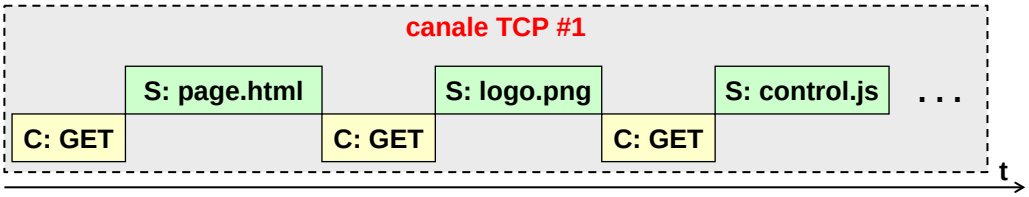

Figura 13.3: Diagramma temporale del trasferimento con HTTP/1.1 base.

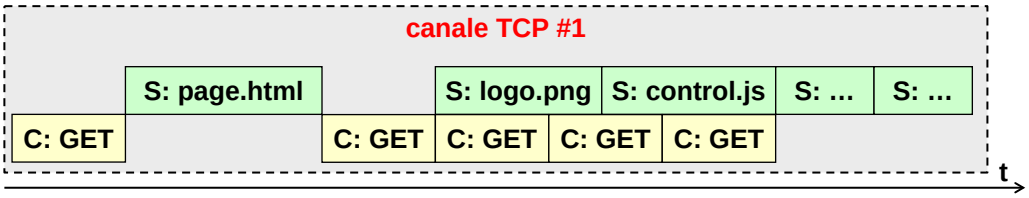

Figura 13.4: Diagramma temporale del trasferimento con HTTP/1.1 e pipeline.

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.

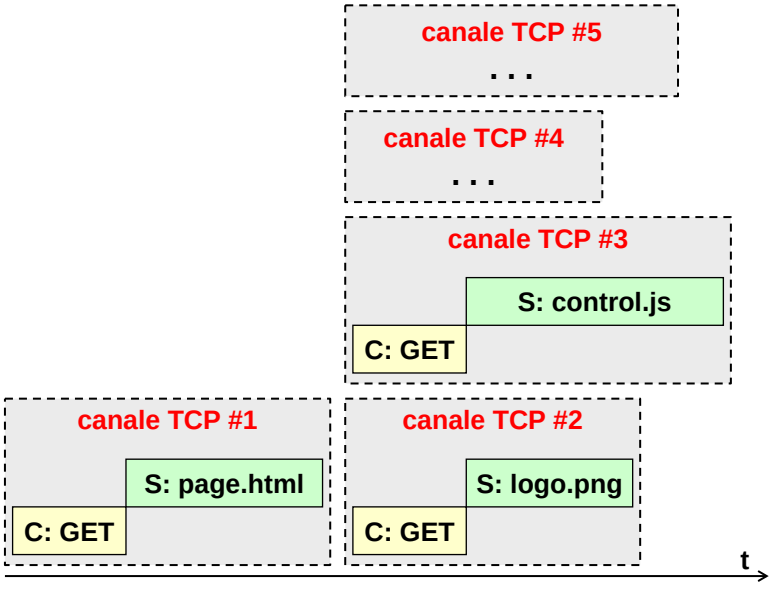

Figura 13.5: Diagramma temporale del trasferimento con HTTP/1.0 e canali paralleli.

Nella figura 13.5 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.

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.

13.2.6 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 anche un risparmio di tempo (come percepito dall'utente finale).

Per comprimere i dati il server sarà costretto ad utilizzare maggiormente 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 è sostanzialmente la banda: comprimere i pacchetti e inviare meno byte è 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. La cosa migliore sarebbe capire se il client è un utente desktop o mobile, e in questo ultimo caso magari evitare la compressione, per non costringerlo a consumare troppa batteria.

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). Sono attualmente i via di sviluppo nuove versioni di software pei dispositivi mobili che spegono il GPS ogni volta che non serve.

13.2.7 Codifiche dati e codifiche di trasmissione

Il protocollo HTTP/1.1 permette di specificare, tramite diversi header, due tipi di codifica diversa: la codifica dati, se non il server non sta mandando più html, bensì direttamente un file zip e la codifica di trasmissione, in cui ciò che viene mandato è ancora html, ma viene fatta una compressione al volo solo per la trasmissione.

13.2.8 Codifiche dati

La codifica dati è applicata alla risorsa prima che questa venga trasmessa ed è quindi una proprietà del dato indipendente dal tipo di trasmissione. Poiché il canale HTTP è pulito a 8

bit, la codifica dati non è finalizzata a prevenire eventuali problemi di trasmissione, è invece generalmente usata per comprimere il dato.

L'informazione riguardante la codifica dati utilizzata è trasmessa insieme al dato ed è indicata separatamente dall'informazione sul tipo di dato: permette così di conservare il MIME-type della risorsa e di applicare la decompressione opportuna al dato ricevuto. Ad esempio, supponiamo che un file di testo sia salvato sul disco del server in formato zip. Se questo file venisse inviato con l'indicazione `Content-Type=gzip`, il ricevente non avrebbe alcun elemento per dedurre quale sia il formato originale. Occorrono dunque degli header che permettano di indicare qual è la codifica dati usata:

Content-Encoding è l'header con cui il server indica al client quale tipo di codifica è stata applicata alla risorsa richiesta;

Accept-Encoding è l'header con cui il client indica al server quale tipo di codifica è in grado di trattare e quindi può accettare.

I possibili valori di questi campi sono:

gzip è il formato di compressione definito dalla FSF e indica che è stato usato il formato GNU zip(è accettato anche x-gzip, utilizzato prima che gzip diventasse uno standard); gli algoritmi usati sono LZ-77 + CRC-32.

compress è il formato storico disponibile negli ambienti Unix, oggi disponibile anche su Linux (è accettato anche x-compress, usato prima che compress diventasse uno standard).

deflate è il formato usato dalla libreria `zlib` oppure tramite il metodo `deflate`

identity indica che non viene usata nessuna codifica ed è il default nel caso non siano presenti uno o più degli header `*-Encoding`.

13.2.9 Codifiche di trasmissione

La codifica specifica le modalità di trasferimento della risorsa, cioè come questa viene codificata nel momento in cui è trasmessa ed è quindi una propietà di questa trasmissione e non del dato.

Gli header TE e Transfer-Encoding si riferiscono alla codifica di trasmissione e danno un'indicazione simile a quella dell'header MIME Content-Transfer-Encoding. In questo caso però, dato che il canale HTTP è pulito a 8 bit, l'unica difficoltà è determinare la lunghezza del messaggio.

I campi TE e Transfer-Encoding possono assumere anche più di un valore. I valori possibili sono gli stessi degli header per la codifica dati (cioè `gzip`, `compress`, `deflate`, `identity`) più `chunked`, una codifica che non ha niente a che fare con la compressione, perciò si può usare insieme ad uno dei tipi precedenti. Deve, se presente, essere l'ultima.

Codifica chunked

Un header del tipo:

```
Transfer-Encoding: chunked
```

indica che la risposta è frammentata in diversi pezzi, detti chunk.

Tale codifica è utile per server dinamici che non possono sapere a priori le dimensioni della risposta, dato che questa dipende dall'esecuzione degli script di una pagina ASP, PHP o JSP. Il client quindi non avrebbe modo di sapere se la risposta è completa o no. Per ovviare a questo problema, il server frammenta la risposta in diversi pezzi, detti appunto *chunk*, per ognuno dei quali indica la dimensione. Obbligatoriamente l'ultimo chunk è vuoto: quando lo riceve, il client sa che la risposta è completa. La codifica `chunked` sarebbe stata meno utile in HTTP/1.0 perché in tale protocollo la fine dei dati era implicita nella chiusura del canale(anche se comunque c'erano dei rischi perchè il caso errore era indistinguibile dalla chiusura del canale), invece in HTTP/1.1, proprio perchè il canale non viene chiuso, prima di fare una nuova richiesta bisogna sapere se è stato ricevuto tutto.

La sintassi del body prevede che siano trasmessi prima i chunck in cui è diviso il testo (possono essere o zero o più) seguiti dall'ultimo chunck vuoto, con dimensione 0; esso è obbligatorio e serve per indicare che il flusso di dati è terminato. Poi seguono eventualmente delle *entity-header* (intestazioni riguardanti il formato dei dati), che vengono allegate in coda poiché le informazioni che trasportano non sono note prima dell'esecuzione degli script della pagina richiesta.

Ogni chunk ha la seguente struttura:

- chunk-size [chunk-extensions] CRLF;
- chunk-data CRLF.

L'attributo chunk-size che precede la trasmissione dei dati veri e propri del chunk è un numero esadecimale che indica la dimensione del chunck espressa in byte. Esso può eventualmente essere seguito da alcune estensioni specifiche del chunk. Il browser prenderà tutti i chunck e li ricomporrà in un unico file sommando la dimensione. Un esempio di utilizzo della codifica chuncked è rappresento e commentato in Fig. 13.6.

13.2.10 Prestazioni di HTTP/1.1

Il protocollo HTTP/1.1 è nato per migliorare le prestazioni. E' dunque importante dimostrare in quale misura tale miglioramento teorico trova riscontro nella realtà empirica.

I vantaggi delle tecnologie nate per migliorare il web sono finora stati evidenti. Esse riducono la dimensione dei file da inviare e di conseguenza anche l'aumento del traffico in rete e i tempi di ricezione percepiti dal client. Esempi di tecnologie che hanno ottenuto risultati validi in questo senso sono il PNG e il CSS.

Il PNG (Portable Network Graphic) è un formato per le immagini che è stato pensato appositamente per il Web. Originariamente si usava il formato GIF e in seguito è stato inventato il JPEG che ha, però, il problema di richiedere calcoli pesanti per la visualizzazione. Il PNG non richiede calcoli ed è quindi più leggero, inoltre ha anche una compressione più efficiente del GIF. Uno degli errori fatto spesso dagli sviluppatori è caricare le immagini ad alta risoluzione così come sono sul web, senza sapare quale sarà il dispositivo client a visualizzarle e quindi appesantendo spesso inutilmente la rete.

Il CSS permette di fornire la descrizione del layout di una pagina separatamente dalla sua descrizione logica. In questo modo il codice HTML della pagina è più snello. Grazie a questo e al fatto che più pagine possono riferirsi allo stesso CSS, è possibile ridurre di molto il sovraccarico della rete e i conseguenti tempi di trasmissione.

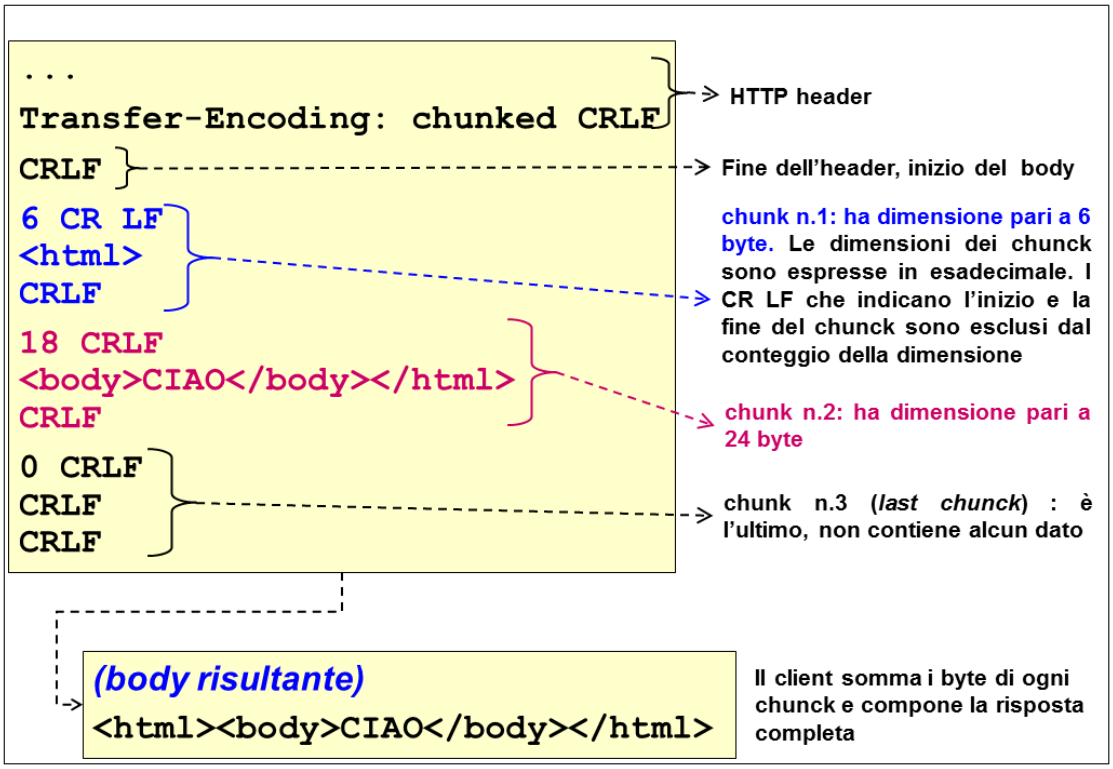

Figura 13.6: Esempio codifica chuncked.

L'impatto di altre tecnologie innovative di HTTP/1.1 non è tuttavia immediatamente determinabile, dato che tutte hanno sia effetti positivi che negativi ed inoltre le interazioni fra esse non sono facilmente prevedibili. In particolare: qual è l'effetto complessivo dell'utilizzo di connessioni persistenti, pipelining e compressione? Ad esempio la compressione ha l'effetto benefico di ridurre la quantità di dati da inviare, ma potenzialmente introduce ritardo sul server, perché il file va compresso, e sul client, perché bisogna decomprimerlo. Inoltre è vero che il pipelining velocizza il processo di invio, però se viene perso il primo pacchetto si è costretti a doverli reinviare tutti. Infine le connessioni persistenti sono sicuramente utili, perchè non richiedono di aprire e chiudere continuamente canali TCP, però si sa che portano un carico maggiore per il server.

Allo scopo di ottenere una valutazione oggettiva nel seguito viene descritto e commentato un test di confronto tra le prestazioni di HTTP/1.1 e quelle di HTTP/1.0, test nel quale i tre aspetti critici sono introdotti gradualmente proprio per poter osservare gli effetti della loro combinazione. Il test è stato svolto dal W3C.

13.2.11 Test HTTP 1.1 vs HTTP 1.0

Il test consiste nella trasmissione di una pagina HTML di 40 kB contenente 43 immagini referenziate, che hanno dimensioni complessive di 130 kB. Inizialmente il client richiede al server la pagina con una GET. Il server risponde inviando il file richiesto. A questo punto il client effettua una GET per ogni oggetto referenziato all'interno della pagina ricevuta. In totale le GET sono dunque 44, e corrispondono a ciascuno degli oggetti coinvolti.

Vengono effettuate quattro prove, in ognuna delle quali i dati vengono trasmessi in diverse modalità :

1. HTTP/1.0 con 4 connessioni simultane

2. HTTP/1.1 con 1 connessione persistente
3. HTTP/1.1 con 1 connessione persistente e pipeline
4. HTTP/1.1 con 1 connessione persistente, pipeline e compressione

In ogni prova viene registrato:

- il tempo trascorso tra la richiesta iniziale del browser e la ricezione completa di tutti gli oggetti coinvolti
- il numero di pacchetti TCP che vengono immessi in rete per portare a termine l'operazione

Il test è mirato a verificare che l'utilizzo di HTTP/1.1 sia effettivamente vantaggioso rispetto a quello di HTTP/1.0. Per questo motivo nella prima prova HTTP/1.0 è utilizzato nel modo più efficiente, cioè sfruttando il parallelismo. Gli elementi caratterizzanti di HTTP/1.1 sono invece introdotti uno alla volta, in maniera da osservarne gli effetti isolati.

I risultati del test sono commentati qui di seguito, prova per prova, e riassunti nel grafico di Fig. 13.7

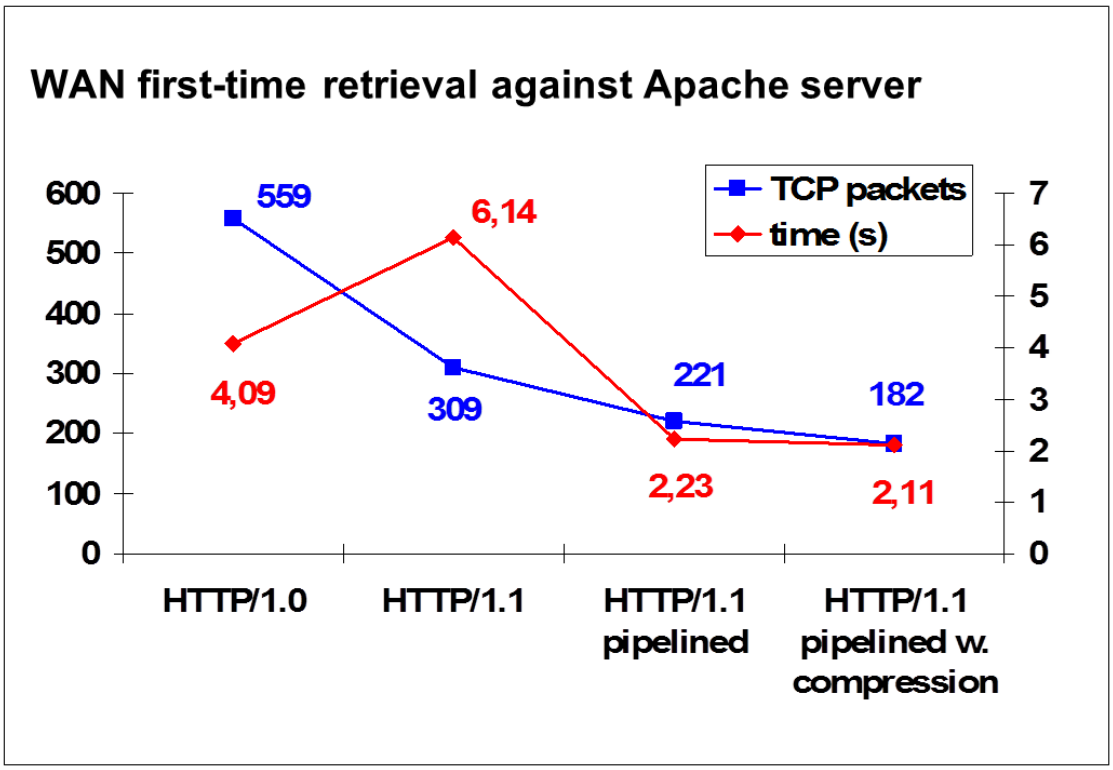

Figura 13.7: Risultati del test di HTTP/1.1 vs HTTP/1.0.

HTTP/1.0, 4 connessioni parallele

Ad ogni GET corrisponde una connessione TCP che viene chiusa nel momento in cui viene ricevuto l'oggetto richiesto. Il client richiede per prima la pagina HTML, quindi apre 4 connessioni simultanee. Quest'ultima operazione viene ripetuta finché non sono stati scaricati tutti i 43 oggetti referenziati. Ogni volta che una connessione viene aperta e chiusa si deve

attendere un certo tempo di setup. Grazie all'effetto del parallelismo il browser utilizzato ha registrato un tempo di completamento dell'operazione di indicativamente 11 (invece che 44) volte quello di scaricamento di un singolo file, ed è pari a 4.09 s, mentre i pacchetti immessi in rete sono 559 pkt.

HTTP/1.1, connessione persistente

E' usata una sola connessione persistente, sulla quale il client richiede e riceve gli oggetti uno per volta. La GET di ogni oggetto avviene solo dopo la ricezione del precedente, per cui il tempo di completamento dell'operazione sale a 6.14 s: non è infatti sfruttato il parallelismo tra canali TCP. Tuttavia tale tempo di completamento non è quadruplicato rispetto al caso precedente poiché, grazie al fatto che la connessione è persistente, l'operazione di setup del canale TCP viene eseguita una sola volta. Per lo stesso motivo il numero di pacchetti in rete scende a 309 pkt (quasi la metà di quelli registrati nella prova con HTTP/1.0).

HTTP/1.1, connessione persistente e pipeline

E' usata una sola connessione persistente, sulla quale il client può subito mandare tutte le richieste a raffica. Il server può quindi organizzare tutti i dati da inviare e, a sua volta, rispondere a raffica. Questo fatto è decisamente vantaggioso in termini di tempi di completamento (2.23 s), che sono quasi un terzo di quelli ottenuti senza pipeline. Si ha un modesto miglioramento anche per quanto riguarda il numero di pacchetti trasmessi che scende a 221.

HTTP/1.1, connessione persistente, pipeline e compressione

Il fatto che venga sfruttata anche la compressione. porta a scarsi miglioramenti. In particolare il tempo complessivo richiesto dall'operazione rimane quasi uguale al caso precedente. Questo è dovuto al fatto che le immagini sono file PNG, e quindi già compresse, per cui l'unica parte su cui la compressione porta ad un effettivo risparmio di byte da trasmettere è l'HTML. In ogni caso se non abbiamo problemi di CPU o batteria la compressione conviene, perchè è vero che il tempo è migliorato di poco, però abbiamo scaricato la rete.

Conclusioni generali

Come è possibile osservare dalla Fig. 13.7 e dalle considerazioni fatte, le prestazioni dell'HTTP/1.1 in termini di tempo sono discrete, ma variano molto in base condizioni specifiche. Per quanto riguarda il carico sulla rete invece, HTTP/1.1 è nettamente più vantaggioso di HTTP/1.0, rispetto al quale snellisce il traffico. In ogni caso questo era solo un esempio semplice di test, perché se si vuole lo si può complicare ad esempio aggiungendo il parallelismo anche in HTTP 1.1; generalmente i browser moderni utilizzano in parallelo circa 6 connessioni HTTP 1.1.

13.3 Metodi aggiuntivi introdotti da HTTP/1.1

Il protocollo HTTP/1.1 è diventato nel tempo, oltre che un protocollo per inviare/ricevere pagine HTML, un metodo di gestione remota dei dati. Ad esempio è utilizzato per permettere a tanti utenti differenti di modificare porzioni di codice open source. Tutto ciò è stato

possibile introducendo dei nuovi metodi rispetto a HTTP/1.0. Questi sono PUT, DELETE, TRACE, OPTIONS, CONNECT.

13.3.1 Il metodo PUT

PUT è un metodo pensato per caricare sul server una porzione di testo, inserito nel body, che costituirà una nuova risorsa o sotituirà una già esistente alla URI indicata. Ad esempio se bisogna collaborare da paesi diversi alla stesura di documenti Europei, questo metodo permette di evitare lunghi e inefficienti scambi di mail.

Sintassi: il termine PUT è seguito dall'indicazione di una URI e della versione HTTP usata. Terminato l'header, segue il body, in cui è trasmessa una nuova risorsa che sostituirà quella precedentemente salvata alla URI indicata nell'header.

Il metodo PUT può essere utilizzato per sostituire una risorsa già esistente con una nuova versione della stessa, ma anche per cancellarne del tutto il contenuto (se il request body è vuoto), oppure per creare una nuova risorsa (se la URI indicata è libera).

Il server che riceve una richiesta di PUT può rispondere con uno dei seguenti codici

201 può essere eventualmente seguito dal testo Created e indica che la risorsa contenuta nel body è stata creata alla URI indicata che era libera

200 può essere eventualmente seguito dal testo OK e indica che il contenuto della risorsa presente alla URI indicata è stato correttamente sostituito con quello indicato nel request body

204 può essere eventualmente seguito dal testo No content e indica che il contenuto della risorsa richiesta è stato rimpiazzato con un request body vuoto, e quindi di fatto cancellato.

Ecco un esempio di richiesta PUT con la quale si richiede di sostituire il contenuto del file `avviso.txt` con il messaggio trasmesso nel body:

```
PUT /avviso.txt HTTP/1.1
Host: lioy.polito.it
Content-Type: text/plain
Content-Length: 40

Il 31/5/2007 non ci sar\&agrave; lezione.
```

Il codice 100-continue

Quando un client invia una richiesta al server, può succedere che quest'ultimo risponda con un messaggio di errore e quindi l'operazione desiderata non vada a buon fine.

Questo può essere causa di inefficienza quando si utilizzano metodi che possono avere un request body come POST e PUT. In particolare con PUT il client corre il rischio di perdere tempo a inviare un body molto lungo al server, il quale potrebbe rispondere che non si può sostituire il contenuto del file richiesto per mancanza di spazio su disco, perché l'utente non è autorizzato o perchè magari non supporta il metodo con cui si sta inviando il file.

Per evitare problemi di questo genere, prima di inviare il request body, il client accerta la fattibilità dell'operazione tramite il codice `100-continue`.

Supponiamo che si voglia sostituire il contenuto del file `voti.pdf` con una versione aggiornata. Il client invia per prima una richiesta di PUT senza body:

```
PUT /voti.pdf HTTP/1.1
Host: www.abc.com
Expect: 100-continue
```

Tale richiesta contiene l'header `Expect` tramite il quale il client richiede conferma esplicita della fattibilità dell'operazione prima di inviare il request body. Il valore 100-continue è il codice con il quale il server autorizza il client a scrivere il file. Quindi, una volta ricevuta dal server la risposta `HTTP/1.1 100 continue`, il client procederà con la richiesta PUT vera e propria compresa di body. Se il client volesse chiedere al server se ha abbastanza spazio per contenere un certo file che gli si sta mandando, c'è un header specifico per indicare la dimensione, che si mette prima di `Expect`, cioè `Content-length`. A questo punto, se il server mi risponderà 100-continue vuol dire che oltre a supportare il metodo PUT ha anche quantità sufficiente di spazio per memorizzare questo file. E' possibile fare un'operazione simile anche in SMTP, in cui si può chiedere al server e il server tramite l'header `Size` può indicarci quale è la massima dimensione che supporta.

13.3.2 Il metodo DELETE

Si tratta un metodo che permette la cancellazione "logica" della risorsa memorizzata ad una certa URI. Il comando DELETE è seguito dall'indicazione della URI e della versione HTTP utilizzata. Ad esempio con la sintassi `DELETE /voti.html HTTP/1.1` il client sta richiedendo di cancellare il file `voti.html`.

La cancellazione tuttavia è solamente "logica" nel senso che non c'è nessuna garanzia che la risorsa venga cancellata fisicamente anche se abbiamo riveuto la risposta OK. Supponiamo ad esempio di aver memorizzato il file e una sua copia con nomi diversi in due cartelle differenti del computer; entrambi puntano alla stessa zona del disco, ma sono due puntatori diversi. Se uno dei due file viene cancellato, non viene cancellata la zona su disco, bensì solo il puntatore; fin quando esiste almeno un link attivo i dati non vengono cancellati. Questo perché potrebbero esserci più persone interessate ad operare su quel file.

Inoltre non è sicuro permettere a qualsiasi client di eliminare dati dal file system del server. Per questo motivo il server nella maggior parte dei casi non svolge alcuna cancellazione fino a quando non è stato autorizzato esplicitamente dal suo amministratore. Per tutti questi motivi, può quindi succedere che il client riceva una risposta affermativa, ma la risorsa non sia stata effettivamente cancellata.

Le risposte che può fornire il server sono:

200 (eventualmente accompagnato dal testo OK) indica che la cancellazione è stata eseguita. L'header può essere seguito da un body che fornisce ulteriori dettagli.

202 (eventualmente accompagnato dal testo Accepted) indica che la cancellazione è subordinata alla decisione dell'amministratore del sistema.

204 (eventualmente accompagnato dal testo No Content) indica che la cancellazione è avvenuta ma non fornisce dettagli.

13.3.3 Il metodo TRACE

E' un comando usato dai sistemisti quando vi sono dei problemi per fare il debug, poiché permette di vedere tutti i proxy e i gateway attraversati dalla richiesta. E' analogo al metodo `traceroute` che permette di vedere quali router sono stati attraversati da un pacchetto IP.

Ogni nodo intermedio, nel momento in cui viene attraversato dalla richiesta, vi aggiunge un header chiamato `Via`, tramite il quale si identifica. Il browser richiede dunque al server di ricevere una copia della request all'interno di un response body con tipo MIME `message/http`. Dalla lettura degli header Via è possibile conoscere il percorso.

Quest'informazione può essere molto utile per identificare i loop nella rete. Può accadere infatti che una richiesta venga inoltrata ad un nodo da cui è già passata e si ritrovi ripercorrere la stessa strada un numero di volte teoricamente infinito. In realtà il protocollo IP, sul quale si appoggia HTTP, è studiato in modo da bloccare eventuali loop, il problema però è che non c'è modo di sapere a livello applicazione dove il messaggio è stato male indirizzato. Tuttavia in questi casi, anche inviando il comando TRACE, il messaggio non verrebbe recapitato e quindi non si avrebbe alcuna risposta del server.

Per risolvere questo problema esiste l'header `Max-Forwards`, che indica il numero massimo di nodi che possono essere attraversati dalla richiesta. Ad esempio se faccio una GET e non ricevo risposta, poi faccio una TRACE e continuo a non ricevere risposta comincio a pensare che ci sia un problema di loop e perciò inserisco questo header e lo incremento progressivamente per capire dove è il problema. Nel conteggio dei forward è incluso anche il mittente.

Ad esempio, supponiamo sia inviato:

```
TRACE / HTTP/1.1
Host: www.polito.it
Max-Forwards: 3
```

Se dopo aver attraversato due proxy (o gateway) intermedi la richiesta non è giunta a `www.polito.it`, il secondo host risponde rinviando la richiesta come avrebbe dovuto fare il destinatario:

```
HTTP/1.1 200 OK
Date: Sun, 20 May 2007 21:29:05 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: message/http

47
TRACE / HTTP/1.1
Host: www.polito.it
Via: 1.0 fred, 1.1 somewhere.com

2
```

La prima intestazione è quella vera e propria della risposta. Segue l'indicazione della dimensione del body, 47 in questo caso, espressa come al solito in esadecimale(71 B in decimale), e il body, nel quale è contenuta la richiesta del client. Infine "2" è il numero di proxy e gateway attraversati.

13.3.4 Il metodo OPTIONS

E' il metodo con cui è possibile richiedere quali sono le opzioni supportate dal server. Sintassi: il comando OPTIONS è seguito dall'indicazione di una URI e della versione HTTP usata.

Se la URI indicata è generica, allora le opzioni indicate nella risposta saranno i metodi HTTP supportati dal server. Se invece la URI è il path di un file particolare, la risposta conterrà anche le opzioni specifiche come ad esempio le lingue in cui la risorsa è disponibile.

Un esempio in cui è utilizzato il metodo OPTIONS è illustrato in fig. 13.8

```
C: OPTIONS / HTTP/1.1
C: Host: www.abc.com
S:     HTTP/1.1 200 OK
S:     Date: Sun, 20 May 2007 21:51:25 GMT
S:     Server: Apache/1.3.31 (Unix)
S:     Content-Length: 0
S:     Allow: GET, HEAD, OPTIONS, TRACE
```

Figura 13.8: Esempio di utilizzo del metodo OPTIONS.

Nel caso in cui questo metodo sia utilizzato aggiungendo il request header `Max-Forward: N`, dove N è il valore deciso dall'utente, se la richiesta non giunge a destinazione prima di aver attraversato N-1 host intermedi, la risposta sarà l'elenco delle opzioni supportate dall'ultimo proxy o gateway raggiunto. OPTIONS può dunque essere opportunamente utilizzato per interrogare i nodi intermedi.

13.3.5 Il metodo CONNECT

Il metodo CONNECT verrà discusso più approfonditamente in seguito. Riguarda l'ambito della sucurezza della trasmissione dei dati ed è infatti utilizzato con dei proxy in grado di creare un canale protetto (come ad esempio SSL).

13.3.6 La trasmissione parziale

Il protocollo HTTP/1.1 offre la possibilità di richiede solo parte dei dati di una risorsa. Si tratta di una funzionalità molto utile se la trasmissione è stata interrotta senza che il client abbia ricevuto tutto ciò che aveva richiesto al server. Questo può avvenire perché è caduta accidentalmente la rete, oppure per volontà dell'utente che ad esempio ha dovuto spegnere la macchina con la quale stava scaricando. In questi casi, una volta ristabilita la connessione, grazie ad HTTP/1.1 non è necessario richiedere nuovamente l'intera risorsa come sarebbe avvenuto con HTTP/1.0, poiché è possibile specificare quali sono solo le parti mancanti.

Il server specifica se la risorsa può essere o meno spezzettata tramite l'haeder `Accept-Range` che assume il valore `none` in caso di risposta negativa, oppure il valore `bytes` se la divisione dei dati è possibile.

`Content-Range` è invece l'header tramite il quale il server indica quali sono le parti della risorsa che sta inviado al client con nel messaggio corrente. Il valore assunto da questo header è nel seguente formato: il termine `bytes` è seguito dall'indicazione del byte iniziale e di quello finale separati dal carattere `-`, segue ancora l'indicazione della dimensione totale

della risorsa richiesta preceduta dal carattere `/`. Se tali dimensioni non sono note, sono sostituite dal carattere `*`.

Alcuni esempi (il numero di byte è esprsso in esadecimale):

`Content-Range: bytes 1-15/30` indica che vengono inviati i byte dal primo al ventunesimo di una risorsa da 48 B.

`Content-Range: bytes 1-15/*` indica che vengono inviati i byte dal primo al ventunesimo di una risorsa di cui non si conoscono le dimensioni totali.

Per quanto riguarda il client invece, questi specifica gli slot di dati a cui è interessato all'interno dell'header `Range`. Gli slot richiesti possono essere multipli e non consecutivi, sono elencati separati da una virgola. I valori degli slot possono essere espressi in tre diversi formati:

start-stop in cui il byte iniziale e finale dell'intervallo sono separati da `-`

-lastN specifica che si desiderano gli ultimi N byte

startByte- indica che si desidera l'intera risorsa a partire dal byte iniziale specificato in poi.

Un header `Range` di esempio è:

```
Range: 23-46,54-63,71-
```

Questa caratteristica è utilissima nella applicazioni peer2peer, perché rende possibile lo scaricamento in parallelo da più server, indicando a ciascuno di quale pezzo si ha bisogno; questo tipicamente velocizza il trasferimento.

13.3.7 Gli entity tag

Spesso i browser prima di scaricare una risorsa accertano di non averne già una copia identica scaricata in precedenza. Il metodo utilizzato per svolgere questa verifica con HTTP/1.0 è quello di inviare una richiesta HEAD e osservare il campo `Last-Modified` della risposta, quindi procedere con una GET solo se questo campo contiene una data più recente rispetto a quella della versione memorizzata sulla cache. Questa procedura però è abbastanza inefficiente per prima cosa per via del fatto che è impiegato un invio-ricezione in più rispetto al necessario, in secondo luogo perché può avvenire che i dati risultino da riscaricare quando in realtà non sono stati modificati. Infatti la data di ultima modifica viene aggiornata anche quando la risorsa è coinvolta in operazioni che non ne cambiano in alcun modo il contenuto. Un caso tipico è quello del backup.

In un ottica di ottimizzazione dunque HTTP/1.1 definisce i cosiddetti Entity tag o *Etag*, ossia degli identificativi della risorsa che cambiano ogni volta che il contenuto della stessa viene effettivamente aggiornato. Si tratta di etichette il cui valore viene indicato dal server tramite l'header `Etag`. L'etichetta è costituita da una stringa esadecimale, opzionalmente preceduta da `W/`. Nel caso in cui la sequenza `W/` sia presente si parla di *weak Etag*, cioè debole, mentre tutti gli altri si dicono *strong Etag*, cioè forti. Due weak Etag uguali si riferiscono a oggetti equivalenti ma diversi, come ad esempio il GIF e il PNG della stessa immagine. Due risorse con uguali strong Etag invece sono identiche, hanno esattamente gli stessi byte. In questo modo è possibile riconoscere che due risorse sono uguali anche se hanno lo stesso nome. Tale stringa esadecimale è quello che si chiama un digest, cioé un riassunto,

utilizzato per identificare univocamente un contenuto, cosa che sarà molto utile quando si parlerà di sicurezza.

L'utente può dunque sfruttare questo strumento per richiedere che un'operazione sia svolta solo se c'è corrispondenza tra il valore dell'etichetta fornito dal server e quello memorizzato in cache. L' header utilizzato a questo scopo è `If-Match`. Ad esempio se l'utente desiderasse modificare il contenuto del file `abc.txt` solo nel caso in cui questo non sia ancora stato modificato, il messaggio di request sarebbe del tipo:

```
PUT /abc.txt HTTP/1.1
Host: www.def.com
Content-Type: text/plain
Content-Length: 78
If-Match: "737060cd8c284d8af7ad3082f209582d"
```

Il caso simmetrico è quello in cui l'utente voglia che la request venga soddidfatta solo se non c'è corrispondenza tra le etichette. L'Etag della versione memorizzata in cache sarà indicato all'interno dell'header `If-None-Match`. Ad esempio, un client che intenda ricevere una risorsa solo se è stata aggiornata rispetto all'ultima versione scaricata invierà una richiesta di questo genere:

```
GET /abc.txt HTTP/1.1
Host: www.def.com
If-None-Match: W/"737060cd8c284d8af7ad3082f209582d"
```

13.3.8 I request header di HTTP/1.1

Sono di seguito elencati gli header che possono esservi solo in una richiesta HTTP/1.1.

Alcuni di essi possono discrezionalmente contenere la sequenza `; q=`, con la quale è specificato il *quality factor*, cioè un valore compreso tra 0 e 1 che indica il grado di preferenza dell'opzione a cui pè riferito. Il quality factor di default è 1. Ad esempio con

```
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
```

l'utente sta richiedendo che i caratteri gli vengano mandati con la codifica ISO-8859-5: la preferenza per questa opzione è 1. Se questo set di caratteri non è disponibile, è anche accettabile la codifica UNICODE-1-1 con preferenza pari a 0.8.

In generale, il valore `*` indica uno qualsiasi dei valori che può assumere l'header.

Iniziamo con il descrivere i campi che indicano i formati accettati dal client:

Accept : formati dei dati accettabili espressi nella forma `tipo/sottotipo`. Sia il tipo che il sottotipo possono eventualmente assumere il valore `*` Es.: `Accept: image/jpeg, image/*;q=0.5`

Accept-Charset : set di caratteri accettabili. Può assumere il valore `*`

Accept-Encoding : codifiche del contenuto accettabili. Può assumere valore come `gzip, compress, chuncked`. Il client dichiara che cosa è in grado di fare così il server mandera file commisurati con la sua capacità.

Accept-Language : lingue accettabili. Può assumere il valore `*`. Es.: `Accept-Language: it, en-gb;q=0.8, en;q=0.7`.

Campi che invece indicano di eseguire la richiesta solo al verificarsi di una determinata condizione sono:

If-Match : applicare il metodo solo se la risorsa corrisponde ad uno degli Etag indicati. Gli Etag sono separati da virgola

If-Modified-Since : applicare il metodo solo se la risorsa è stata modificata dopo la data indicata

If-None-Match : applicare il metodo solo se la risorsa non corrisponde ad alcuno degli Etag indicati. Gli Etag sono separati da virgola

If-Range : riporta il valore di un Etag oppure una data e indica di inviare tutta la risorsa se è cambiata, altrimenti solo la porzione indicata specificata dal campo `Range`

If-Unmodified-Since : applicare il metodo solo se la risorsa non è stata modificata dopo la data indicata.

I restanti campi sono:

Authorization : in questo header sono fornite le credenziali per accedere a pagine protette tramite autenticazione presso l'origin server.

Expect : comportamento che il client si aspetta che il server abbia. Il server risponde con 417 se non può tenere il comportamento atteso dal client.

From : fornisce un indirizzo di posta elettronica dell'utente che sta usando il browser. Può essere importante per contattare chi ha attivato una macchina automatica che crea problemi, ad esempio i robot utilizzati per indicizzare possono entrare in un loop che impegna tutte le risorse di una pagina; ma in generale è fortemente deprecato per mantenere la privacy ed evitare lo spamming

Host : indica l'host virtuale da contattare (porta di default è la 80)

Max-Forwards : massimo numero di forward accettati. E' usato con i metodi TRACE e OPTIONS per identificare i nodi intermedi. Il suo valore è decrementato da ogni nodo

Proxy-Authorization : fornisce le credenziali per l'autenticazione presso di un proxy

Range : chiede di ricevere solo l'intervallo di dati indicato

Referer : URI assoluta o relativa della pagina che ha generato l'attuale richiesta. Tale campo è assente se la URI viene inserita da tastiera

TE : indica se sono o meno accettati i *trailers*, cioè gli header allegati in coda al il contenuto chunked, e eventuelmente quali sono le codifiche di trasferimento supportate, con opzionale specifica dell'indice di preferenza. Ad esempio, `TE:` indica che i trailers non sono accettati e non fornisce indicazioni riguardo alle codifiche di trasferimento, `TE: deflate` significa che i trailers non sono supportati e che è accettata la codifica deflate, `TE: trailers, deflate;q=0.5` significa invece che sono ammessi i trailer e la codifica deflate con preferenza 0.5

User-Agent : identifica il software che implementa il client, cioè il tipo di broswser (nella maggior parte dei casi).

13.3.9 I response header di HTTP/1.1

Accept-Ranges : indica se il server permette il download parziale

Age : indica i secondi trascorsi da quando la risorsa è stata inserita in cache sul server. Ha senso solo in proxy, gateway o server con cache, affinché l'utente sappia quanto è vecchia la risorsa che gli viene fornita da un nodo intermedio

ETag : identificativo della risorsa richiesta

Location : effettua un redirect alla URI assoluta indicata

Proxy-Authenticate : indica la sfida proposta da un proxy per autenticare un client che voglia accedere a risorse protette

Retry-After : in caso di indisponibilità del server esse segnala un errore temporaneo, indica quando riprovare fornendo una data o un numero di secondi.

Server : identifica il software che implementa il rispondente, cioè l'applicazione server

Vary : indica i campi della richiesta da cui dipende la risposta. Assume il valore `*` oppure contiene l'elenco degli header che devono essere presenti nella request affinché la risposta abbia senso. E' quindi usato per testare la validità di una copia nella cache di un proxy o un gateway.

WWW-Authenticate : sfida proposta dall'origin server per autenticare il client, analogo a `Proxy-Authenticate`.

13.3.10 General header di HTTP/1.1

I general header valgono sia per le richieste sia per le risposte.

Cache-Control : controllo della cache nella richiesta e nella risposta. Contiene direttive del client e del server verso i proxy

Connection : assume il valore `close` per indicare a server o proxy che si vuole una connessione non persistente

Date : data di invio della richiesta o della risposta

Pragma : se presente, assume necessariamente il valore `no-cache`. Indica che si intende scaricare la risorsa originale e non una sua copia in cache. In realtà , a questo scopo si preferisce usare `Cache-Control`, per cui questo header è mantenuto solo per compatibilità con la versione HTTP precedente.

Trailer : è seguito da un elenco di header separati da virgola. Si tratta dei trailers, header trasmessi in coda al body chunked. Fra questi non possono esserci `Transfer-Encoding`, `Content-Length` e `Trailer`

Transfer-Encoding : indica la codifica di trasferimento usata

Upgrade : il client chiede di passare ad un protocollo migliore. Sono elencati i protocolli proposti, tra i quali il server può scegliere a propria discrezione

Via : sequenza dei nodi intermedi (proxy o gateway). Nell'elenco, per ciascun nodo, sono indicati il protocollo utilizzato, il nome dell'host (o uno pseudonimo) ed eventualmente un commento. Insieme al nome di ciascun host, può opzionalmente essere specificata la porta con cui questi ha comunicato. Es.: `Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)`

Warning : avviso per segnalare la presenza di condizioni particolari. E' tipicamente usato dai proxy. Formato: codice di stato seguito da nome dell'agente, un testo che ne chiarisce il significato e opzionalmente la data. Il testo è codificato in ISO-8859-1 o in RFC-2047. Un elenco dei valori che può assumere questo campo e dei loro significati è riassunto nella tabella in Fig. 13.9.

codice	*testo*	*spiegazione*
110	Response is stale	la risorsa è vecchia
111	Revalidation failed	impossibile contattare origin server per rivalidare.
112	Disconnected operation	il proxy è scollegato dalla rete
113	Heuristic expiration	il proxy ha scelto euristicamente una scadenza
199	Miscellaneous warning	dovrebbe normalmente essere abbinato ad un testo che spieghi all'utente quali condizioni si sono verificate
214	Transformation applied	il proxy ha cambiato la codifica dei dati trasmessi
299	Miscellaneous persistent warning	simile a 199 ma persistente

Figura 13.9: Gli status code dell'header Warning.

13.3.11 Gestione della cache

All'interno del campo `Cache-Control`, come si è detto, vengono indicate le direttive da parte del client e del server riguardo alla gestione della cache nei nodi intermedi. Generalmente il client dichiara se desidera o meno ricevere risposte provenienti da una cache, mentre il server indica se è il caso o no di memorizzare in cache una certa risposta.

Le direttive fornite dal client sono:

no-cache La risposta deve arrivare dall'origin server e non da un proxy intermedio.

no-store Non bisogna memorizzare nulla nei nodi intermedi: né la richiesta né la risposta e nemmeno parte di esse.

max-age=X La risposta non dev'essere più vecchia del numero di secondi (X) indicato.

max-stale Sono ammesse anche risposte scadute. Opzionalmente può seguire la sequenza `=X` , dove X è il numero massimo di secondi che possono essere trascorsi dalla scadenza affinché la risposta sia considerata ancora valida.

min-fresh=X La risposta dev'essere ancora valida tra X secondi.

no-transform Ai proxy è vietato effettuare trasformazioni della risorsa. Ad esempio potrebbero trasformare un GIF in un PNG per risparmiare spazio.

only-if-cached Direttiva utile con reti sovraccariche: indica che non bisogna contattare l'origin server, la risposta deve arrivare da una cache intermedia.

Le direttive fornite dal server riguardo alla gestione della cache sono:

public La risposta può essere messa in cache sia condivise da più client (shared) che visibili solo a chi ha mandato la richiesta (private).

private La risposta non può essere salvata in una cache condivisa, ma solo essere memorizzata in cache accessibili unicamente dal client che ha inviato la richiesta. L'attributo può essere seguito dall'elenco dei particolari header che sono privati. Ad esempio `Cache-Control: private=''Date,Server''` indica che gli header `Server` e `Date` non devono essere memorizzati, mentre `Cache-Control: private` indica che l'intera risposta non dev'essere memorizzata .

no-cache La risorsa non va salvata in nessuna cache, né private, né shared. Può essere seguito da `=` e l'elenco degli header da non memorizzare.

no-store La risposta non è da salvare su disco, ma può essere mantenuta in RAM. E' riferito a cache sia shared sia private.

no-transform Il server indica agli intermediari di non trasformare la risorsa.

must-revalidate Quando la risorsa in cache diventa stale (scaduta), chi ha effettuato la richiesta (il proxy o il client) deve "rivalidare" tale risorsa presso l'origin server, cioè verificare se è effettivamente da riscaricare o no.

proxy-revalidate E' come `Must-Revalidate`, ma è riferito solo al proxy

max-age=X Indica tra quanti secondi (X) scadranno i dati trasmessi

s-maxage=X Indica tra quanti secondi (X) scadranno i dati trasmessi e memorizzati in una shared cache ed ha priorità su max-age. Ovviamente si riferisce soltanto alle shared cache.

Nota: `max-age` e `s-maxage` hanno priorità su un eventuale header `Expires` della risorsa, i cui valori possono essere in contraddizione con quanto indicato dai primi due.

13.3.12 Gli Entity header

Gli entity header sono altri header che sono inseriti sia nella richiesta che nella risposta e che definiscono i formati del dato che viene trasferito.

Allow : indica i metodi permessi su una determinata entità . Viene fornito dal server a completamento del warning `405 Method Not Allowed`. Dal client è invece inserito nel messaggio PUT per suggerire i metodi con cui la risorsa che viene caricata dovrà essere accessibile ai richiedenti.

Content Encoding : specifica la codifica dati

Content-Language : indica il linguaggio del contenuto della risorsa

Content-Length : numero (in base 10) di byte del body

Content-Location : indica la URI assoluta o quella relativa dove è possibile trovare la risorsa. Utile nel caso in cui una risorsa fornita a seguito di una negoziazione sia anche accessibile direttamente. Ad esempio il client richiede la pagina `guide.html` specificando che preferisce la lingua italiana (con `Accepted-Language:it, fr;q=0.8, *;q=0.6`); il server può allegare alla risposta `Content-Location: /guideit.html`, in modo che in seguito il client possa richiedere direttamente l'indirizzo della versione italiana.

Content-MD5 : fornisce un *MD5-digest*, un *riassunto in MD5* (l'MD5 è una funzione crittografica). Tale riassunto ha lo scopo di proteggere l'integrità dei dati trasferiti da errori casuali, non da attacchi

Content-Range : intervallo di byte della risorsa trasmessi

Content-Type : MIME-type della risorsa

Expires : data di scadenza della risorsa

Last-Modified : data di ultima modifica

Capitolo 14

Il linguaggio PHP

14.1 Introduzione

14.1.1 Cos'è PHP?

PHP è un acronimo ricorsivo. Infatti, PHP significa "PHP Hypertext Preprocessor". Le principali informazioni e la documentazione possono essere reperite all'indirizzo http://www.php.net.

PHP è un linguaggio di scripting open-source molto diffuso e multiuso. In particolare, si può usare per:

- generare script eseguibili da riga di comando;
- creare applicazioni desktop (grazie all'estensione GTK che consente di creare la grafica), anche se altri linguaggi – come Python – sono più adatti a questo scopo;
- inserire script server-side all'interno di pagine web.

Nonostante ciò sia vero, bisogna dire che il PHP è particolarmente adatto ed è usato soprattutto come script server-side per generare dinamicamente pagine web. Le altre applicazioni sono possibili, ma la loro reale diffusione è marginale.

Inoltre, il codice scritto in PHP è altamente portabile. Ciò significa che è indipendente dall'hardware e dal software impiegato. Infatti, è multi-piattaforma (si può utilizzare con qualsiasi sistema operativo) e non dipende dal web server impiegato, purché sia stato installato correttamente.

14.1.2 Installazione

PHP può essere installato in due modi, come modulo del server web o come interprete CGI. Tuttavia, la seconda opzione è sconsigliata: le applicazioni CGI sono eseguite come processi indipendenti e hanno i problemi già visti. Per cui, un elevato numero di connessioni implicherebbe la generazione di molti processi, il che crea due problemi: c'è un elevato dispendio di risorse (RAM e CPU) e un certo tempo di attivazione del processo che aumenta la latenza. Inoltre, un processo non può comunicare facilmente con ciò che è esterno ad esso: quindi è complicata anche l'interazione con il server.

Per tutte queste ragioni le prestazioni peggiorano notevolmente. Infatti, avevamo visto che il paradigma CGI è adatto ad applicazioni con pochi utenti e che richiedono tempi si risposta elevati e ingenti risorse di calcolo. In questi casi, si sfruttano le migliori prestazioni nei calcoli da parte di un eseguibile: il risparmio di tempo ottenuto così controbilancia gli altri problemi. Però, il codice PHP non genera un eseguibile, ma viene interpretato di volta in volta. Quindi, non si ha neppure questo vantaggio. In virtù di queste considerazioni, PHP non viene quasi mai installato come interprete di CGI.

Normalmente, PHP è installato come modulo del server web. Tutti i maggiori server web hanno un modulo PHP disponibile (anche IIS). Un pacchetto molto semplice da installare è EasyPHP (`http://www.easyphp.org`): contiene il web server Apache, con un modulo PHP già installato, e il database mysql. Il principale limite di questo pacchetto è la disponibilità solo per sistemi operativi Windows.

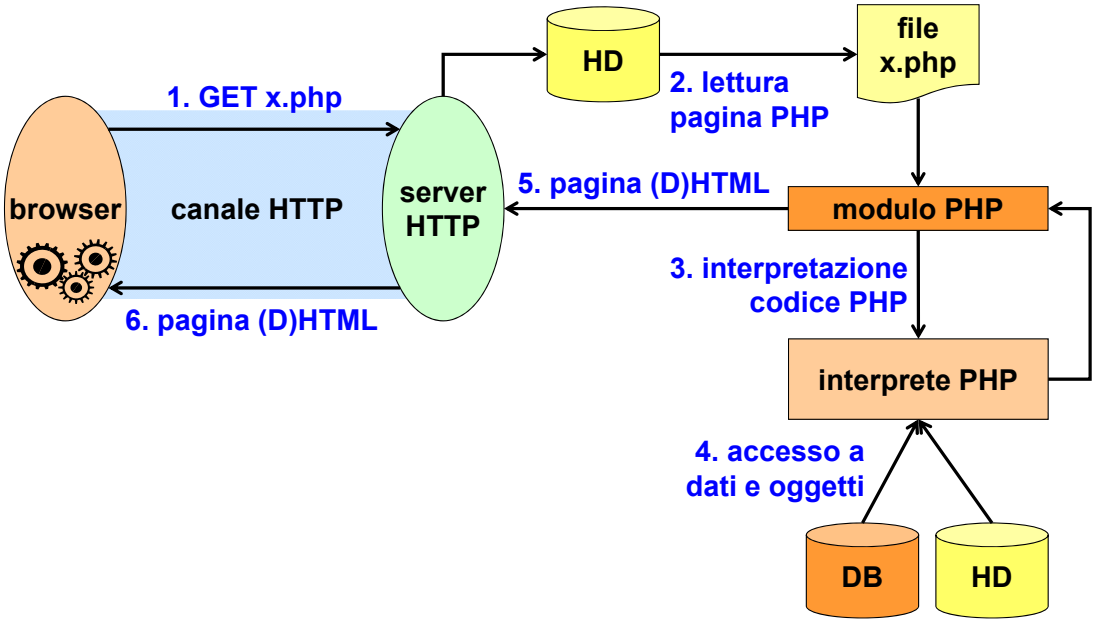

Figura 14.1: Architettura di un server HTTP con un modulo PHP.

Il modulo PHP comunica con il server HTTP per mezzo dell'interfaccia SAPI (acronimo di Server Application Programming Interface). In questo modo, il server HTTP fornisce i file PHP al modulo PHP, che li interpreta e restituisce al server una pagina DHTML o HTML. Si definisce DHTML una pagina costruita con HTML, CSS e Javascript in modo tale da cambiare in maniera dinamica il contenuto e la rappresentazione del contenuto.

In figura 14.1 si può notare come avviene l'interazione. Il client invia una richiesta HTTP GET per una pagina PHP (nell'esempio è `x.php`). Il server HTTP legge dall'hard disk la pagina indicata e la invia al server HTTP. Quest'ultimo si interfaccia con il modulo PHP, che la invia all'interprete: esso accede ai vari oggetti che sono indicati nel file PHP, come database o file, e restituisce un file (D)HTML. Infine, il server HTTP invia la pagina così ottenuta al browser del client.

Quindi, se il file x.php richiesto è:

```
<html>
<head>
<title>Saluti</title>
</head>
```

```
<body>
<?php
for ($i=1; $i<=5; $i++)
printf ("<h%d>Ciao!</h%d>\n", $i, $i);
?>
</body>
</html>
```

L'interprete PHP restituisce al server HTTP la seguente pagina HTML:

```
<html>
<head>
<title>Saluti</title>
</head>
<body>
<h1>Ciao!</h1>
<h2>Ciao!</h2>
<h3>Ciao!</h3>
<h4>Ciao!</h4>
<h5>Ciao!</h5>
</body>
</html>
```

E questa pagina è ciò che il server invia al browser del client.

I file PHP sono semplici file di testo. Il motivo per cui sono interpretati diversamente è la loro estensione. Infatti, nei file di configurazione del server HTTP sono specificate le estensioni che devono essere interpretate come file PHP (normalmente l'estensione è .php, ma si possono avere anche .php3, .phtml e altre). Un file PHP contiene HTML, CSS e Javascript, ma anche script racchiusi tra tag speciali (come `<?php` e `?>`): il modulo PHP prende ciò che è tra questi tag e lo interpreta come script PHP, restituendo una pagina HTML.

Solitamente, per controllare la corretta installazione di PHP, si crea una pagina di test contenente la sola istruzione `<?php phpinfo(); ?>`. Se il PHP è installato correttamente, questa istruzione restituisce una pagina contenente molte informazioni rilevanti riguardo il PHP installato e la sua configurazione. Ad esempio, riporta la versione del PHP installata.

14.2 Le funzioni di output

L'output in PHP è e deve essere codice HTML (o CSS o Javascript). L'output generato dal PHP non è ciò che viene mostrato all'utente ma costituisce il codice sorgente che il browser del client interpreta.

Le funzioni più semplici per generare output sono `echo()` e `print()`. In entrambi i casi le parentesi non sono necessarie e le due funzioni sono sinonimi. Esistono anche le funzioni come `printf()` (come visto in un esempio precedente) e affini, che sono analoghe a quelle usate dal linguaggio C. Con la funzione `printf()`, si possono anche usare i parametri posizionali: con la sintassi %numero$formato si indica uno specifico parametro. In questo caso bisogna numerare tutti i parametri: non se ne può numerare uno solo. Tale sintassi è conveniente se ci sono varie ripetizioni o se non si può o vuole cambiare l'ordine degli argomenti. Un esempio interessante è dato dal formato per generare l'output di una data: è necessario che

cambi da paese a paese, ma i dati sono sempre gli stessi. Per cui si può usare una variabile che indichi il formato e cambiarla in maniera opportuna:

```
<?php
$format = '%2$02d/%1$02d/%3$4d';
printf ($format, 31, 1, 2013);
?>
```

Se nel file di configurazione del PHP l'attributo `short_open_tag` è settato a `On`, c'è un'ulteriore possibilità. Anziché usare una funzione `echo`, si può usare la sintassi più breve `<?=$var ?>` che restituisce in output il valore della variabile `$var`. In questo caso, però, bisogna stare molto attenti a non inserire alcuno spazio tra il tag di apertura e `=`.

14.3 Le variabili e i tipi

14.3.1 Introduzione

Il linguaggio PHP ha una sintassi simile al C, ma con molte varianti e aggiunte. È case-insensitive per i nomi delle funzioni, mentre è case-sensitive per i nomi delle variabili. Il segnale di fine comando è dato dal punto e virgola (`;`) e non dal semplice ritorno a capo (come per Javascript). I commenti possono essere inseriti sia con la sintassi `//`, che commenta fino alla fine della linea, sia con la sintassi `/* ... */` che consente di commentare su più linee (esattamente come in Java).

Attualmente si è alla versione 5.4.14, ma sono ancora usate anche la versione 3 e 4. Nonostante le varie versioni siano sostanzialmente compatibili per script molto semplici, ci sono sostanziali differenze. Ad esempio, la sintassi per la definizione degli attributi nelle classi prevede la parola chiave `var` seguita dal nome dell'attributo in PHP 4. Invece, in PHP 5 questa parola chiave non è più necessaria ed è possibile gestire la visibilità con le parole chiave `private`, `protected` e `public`: in PHP 4 tutti gli attributi e i metodi sono pubblici. Inoltre varie funzioni sono diventate deprecate o non sono più supportate ed alcune di esse sono state sostituite da nuove funzioni analoghe.

Come accennato, gli script PHP devono essere racchiusi tra appositi tag. Il tag più usato è `<?php ... ?>`. Questo tag è sempre disponibile; pertanto è raccomandato per la portabilità. Altro tag sempre disponibile, ma meno usato perché più lungo è

```
<script language="php"> ... </script>
```

Gli altri tag sono utilizzabili sono su server configurati opportunamente. Il tag `<? ... ?>` funziona solo su server che abbiano l'attributo `short_open_tag` settato a `On` sul file di configurazione. Allo stesso modo, il tag `<% ... %>` è riconosciuto solo dai server che abbiano `asp_tag=On`. Questi attributi sono visualizzabili nella schermata restituita da `<?php phpinfo(); ?>`, ma possono essere modificati solo editando il file di configurazione.

Gli identificatori possono contenere caratteri alfanumerici o _. Sono considerati alfabetici tutti i caratteri ASCII tra il 127 e il 255; però, si sconsiglia l'uso di caratteri particolari che siano difficilmente riproducibili su tutte le tastiere. Un identificatore può iniziare solo con un carattere alfabetico o _. Perciò, `9nove` non è un identificatore valido, mentre lo sono `voto2`, `totale`, `_27`.

Le variabili PHP sono degli identificatori preceduti dal carattere `$`. Gli identificatori sono case-sensitive, ma a differenza del C, non è necessario dichiarare una variabile prima

di utilizzarla. Se non sono state inizializzate, le variabili assumono un valore di default che dipende dal tipo: ad esempio, se usata come stringa, una variabile non inizializzata è una stringa vuota, mentre come intero assume il valore 0. Per verificare se una variabile è stata inizializzata o dichiarata, si può usare la funzione `isset()`, che restituisce un valore Booleano: `true` se la variabile esiste, `false` altrimenti. Per distruggere una variabile si può usare `unset()`. Il nome `$this` è riservato: indica la pseudo-variabile disponibile all'interno di un metodo di una classe che identifica l'oggetto chiamante (che solitamente è l'oggetto a cui appartiene il metodo).

PHP consente di usare delle variabili come nome di altre variabili. Dato che il concetto è complesso da spiegare, ma piuttosto intuitivo, ecco un paio di esempi:

```
<?php
$var = "pippo";
$pippo = "pluto";
echo $$var; // restituisce pluto
?>

<?php
$var = 2;
$pippo2 = "pluto";
echo $pippo$var; // genera errore invece
?>
```

Al contrario del C, le variabili non sono tipate (come in Javascript). Ciò significa che assumono il tipo adeguato al momento dell'uso e possono cambiare tipo durante l'esecuzione di uno script: c'è una conversione automatica di tipo al momento dell'uso. In PHP sono disponibili tre tipologie di tipi di dati: i tipi scalari, i tipi composti e i tipi speciali.

Per conoscere il tipo di una variabile sono disponibili varie funzioni. La funzione `gettype()` restituisce il tipo dell'espressione indicata come parametro; `is_integer()`, `is_string()`, ... restituiscono un valore Booleano che indica l'appartenenza o meno al rispettivo tipo; infine, la funzione `var_dump()` è molto utile in fase di debug: restituisce, infatti, sia il tipo che il valore del parametro. Ad esempio,

```
<?php
var_dump(1); //restituisce int(1)
?>
```

Inoltre, è possibile effettuare il cosiddetto type cast. Questa operazione spesso non è necessaria, perché viene eseguita automaticamente. Ad ogni modo, se si desidera si può forzare il tipo di un'espressione precedendola con il tipo voluto messo tra parentesi. Perciò, se si ha una stringa contenente `"5"` la si può rendere un intero con l'espressione `(int)"5"`.

I tipi scalari sono:

- Booleani (Boolean o Bool);
- interi (integer o int);
- numeri razionali (float o double), che sono sempre memorizzati come double;
- stringhe (string).

I tipi composti sono:

- vettori (array) ;
- oggetti (object).

I tipi speciali sono:

- resource;
- NULL.

14.3.2 I valori Booleani

In PHP i valori Booleani sono espressi con le parole chiave `TRUE` e `FALSE`. Entrambe sono case-insensitive. Nella conversione a Booleano (può avvenire automaticamente o per casting esplicito tramite `(Bool)` o `(Boolean)`), i seguenti valori sono considerati `FALSE`:

- il Booleano `FALSE` stesso;
- l'intero 0 e il double 0.0;
- la stringa vuota e la stringa `"0"`;
- un array vuoto (ossia con zero elementi);
- il valore `NULL` e le variabili non definite (valore `undefined`).

Qualsiasi altro valore è considerato `TRUE`, incluse tutte le variabili di tipo `resource`. Perciò, si ha:

```
<?php
var_dump((Bool) "");        // Bool(false)
var_dump((Bool) 1);         // Bool(true)
var_dump((Bool) -2);        // Bool(true)
var_dump((Bool) array(12)); // Bool(true)
var_dump((Bool) array());   // Bool(false)
var_dump((Bool) "false");   // Bool(true)
?>
```

14.3.3 Gli interi

Un intero è un numero relativo (quindi ha sempre un segno). La dimensione di un intero non dipende dalla piattaforma e può essere determinata usando la costante `PHP_INT_SIZE`; il massimo valore può essere, invece, determinato con `PHP_INT_MAX`. Se avviene overflow, l'intero è interpretato come un double.

Sebbene sia possibile convertire un numero razionale in intero, ciò è sconsigliato, perché il comportamento che tale casting produce è impredicibile. Pertanto è meglio affidarsi alle funzioni `round()`, `floor()` e `ceil()` che arrotondano rispettivamente all'intero più vicino, al minor intero maggiore o uguale e al maggiore intero minore o uguale.

I numeri interi sono esprimibili in varie basi. La notazione ottale si ottiene ponendo uno 0 di fronte al numero (ad esempio `012`). La notazione esadecimale prevede che il numero sia preceduto da `0x`. Infine, i numeri binari sono preceduti da `0b`.

14.3.4 Le stringhe

Le stringhe possono essere rappresentate con quattro notazioni differenti. La prima è la notazione single-quote, che significa scrivere la stringa tra singoli apici. Per rappresentare il singolo apice all'interno di una stringa di questo tipo bisogna anteporvi un backslash che funga da escape. Stessa cosa bisogna fare per scrivere il carattere backslash stesso. Non sono ammessi altri escape, per cui anche `\n` e `\r` così come i doppi apici sono rappresentati così come sono. Inoltre non espande le variabili. Ad esempio:

```
<?php
$stringa = 'Esempio di una stringa con i caratteri ", \n e \r';
echo $stringa; // restituisce: Esempio diuna stringa con i caratteri ",
   \n e \r
?>
```

Le stringhe double-quoted, invece, riconoscono le sequenze di escape e espandono le variabili. Tuttavia, per variabili complesse è consigliabile usare la sintassi `{$var}` (attenzione a non inserire alcuno spazio tra la graffa e il dollaro). Le sequenze di escape riconosciute sono le stesse del C, con l'aggiunta di `\$` e `\"`. Quindi, si ha:

```
<?php
$var="variabile";
$stringa = " \$var contiene $var";
echo $stringa; // restituisce: $var contiene variabile
?>
```

Con entrambe le sintassi precedenti è possibile effettuare concatenazioni di stringhe e variabili. Per concatenare due stringhe si usa l'operatore `.` (punto).

Un'altra sintassi utilizzabile per rappresentare le stringhe è *heredoc*. Essa prevede la presenza dell'operatore `<<<` seguito da un identificatore (che può essere racchiuso tra double quote o no) – detto marker – e quindi da un ritorno a capo. Segue la stringa stessa e quindi lo stesso identificatore usato in apertura per determinarne la chiusura. L'identificatore di chiusura deve iniziare nella prima colonna della linea e segue le stesse regole degli identificatori già viste. Heredoc si comporta come una stringa double-quoted. Ad esempio:

```
<?php
$name = 'MyName';
$foo[0]="bar";
echo <<<EOT
My name is "$name".
Now, I am printing some {$foo[0]}.
EOT
// restituisce: My name is "MyName". Now, I am printing some bar.
?>
```

Infine, c'è la rappresentazione *nowdoc*. È analoga alla precedente ma il marker è una stringa single-quote e la stringa rappresentata è di tipo single-quote.

14.3.5 Gli array

Gli array in PHP sono in realtà delle mappe ordinate. Una mappa è un tipo che associa dei valori a una chiave. La chiave può essere sia un intero, sia una stringa. Tutti gli altri

tipi, se usati come chiavi, subiscono un casting a uno di questi due tipi. Un array può anche contenere allo stesso tempo chiavi intere e di tipo stringa. Un valore di un array può essere un qualsiasi tipo, anche a sua volta un array: perciò è possibile creare array multidimensionali.

Esistono diverse funzioni per gestire gli array (sia per l'ordinamento che per altre funzionalità). Per crearli, invece si può usare il costrutto `array()` o si può direttamente popolare l'array. Ad esempio:

```
<?php
$array1=array(
    "key1"  => "value1",
    "key2" => "value2",
    "key3" => "value3"
)
$array2["key1"]="value1";
$array2["key2"]="value2";
$array2["key3"]="value3";
// $array1 e $array2 sono identici
?>
```

L'accesso a un elemento in un array è effettuato indicando tra parentesi quadre la chiave corrispondente (così come in C). L'aggiunta di un elemento, invece, può essere effettuata in due modi. O con il metodo già visto nell'esempio precedente a proposito del secondo array o, nel caso di array con indici interi, si può usare `$array[]` per aggiungere un elemento con indice pari all'intero successivo alla maggiore delle chiavi presenti. Ad esempio:

```
<?php
$array[0]="value1";
$array[3]="value2";
$array[]="value3";
// avremo $array[4]="value3"
?>
```

Una funzione interessante per il debug nel caso di array è `print_R()`. Essa restituisce il contenuto dell'array passato come parametro, formattato in maniera facilmente comprensibile. È molto utile soprattutto nel caso di array multidimensionali per capirne meglio la struttura.

14.4 Le variabili predefinite

In PHP ci sono alcune variabili predefinite, che esistono senza che il programmatore le istanzi o le dichiari esplicitamente. Queste variabili sono chiamate "superglobal" (o superglobali in italiano). Il nome indica il fatto che sono visibili in ogni punto dello script (anche all'interno di funzioni o classi).

Quindi, le variabili superglobali sono variabili precostituite – sotto forma di array associativi – che sono sempre disponibili. Esse sono:

- `$GLOBALS`: tiene traccia di tutte le variabili globali disponibili nello script (i nomi delle variabili sono le chiavi di questo array);
- `$_SERVER`: contiene informazioni definite dal web server come gli headers, i percorsi, ecc.; queste variabili sono create dal web server e non c'è alcuna garanzia su quali siano fornite da un web server;

- `$_POST`: contiene le variabili passate allo script corrente con il metodo HTTP POST;
- `$_GET`: contiene le variabili passate allo script corrente con il metodo HTTP GET;
- `$_REQUEST`: contiene l'unione di `$_GET`, `$_POST` e `$_COOKIE`;
- `$_FILES`: contiene una lista degli oggetti caricati allo script corrente con il metodo HTTP POST;
- `$_COOKIE:` contiene le variabili passate allo script corrente come cookie;
- `$_SESSION`: contiene le variabili di sessione disponibili;
- `$_ENV`: contiene le variabili di ambiente fornite dal sistema in cui è eseguito PHP.

`$_GET`, `$_POST` e `$_REQUEST` sono particolarmente importanti perché contengono le informazioni trasmesse dal client in un form. I form hanno un attributo (`method`) in cui bisogna indicare con quale metodo si intende passare i dati inseriti dall'utente negli appositi campi di input: `$_GET` e `$_POST`, ovviamente, contengono rispettivamente i campi trasmessi con il metodo GET e il metodo POST.

`$_GET`, `$_POST` e `$_REQUEST` sono dei normali vettori associativi. Le chiavi sono i nomi (cioè i valori dell'attributo `name`) dei tag `input` definiti nel form da cui si arriva: ad essi sono associati i valori che questi hanno assunto ad opera dell'utente o meno (si pensi ad esempio agli elementi `input` di tipo `hidden`).

Tutti gli elementi input del form da cui si arriva sono sempre presenti e disponibili, anche se vengono lasciati vuoti. Gli unici che fanno eccezione sono gli elementi `submit`: se, ad esempio, ci sono due pulsanti submit diversi all'interno dello stesso form, nel vettore `$_GET` o `$_POST` ci sarà solo quello che è stato premuto dall'utente.

Facciamo un esempio per spiegare meglio questi concetti. Sia questo il codice del file `x.html`:

```
<!-- header del file -->
<form action="y.php"  method="GET">
   <p>Inserisci il tuo nome:</p><input type="text" name ="nome">
   <input type="submit" name="conferma" value="Conferma">
   <input type="submit" name="altraConferma" value="Altra conferma">
</form>
```

Ora inseriamo come nome Pippo e clicchiamo sul pulsante "Conferma". Fare ciò equivale a inviare una richiesta HTTP di questo tipo:

```
GET /y.php?nome=Pippo&conferma=Conferma HTTP/1.1
...
```

Sia questo il contenuto del file `y.php` contenuto nella stessa cartella:

```
<!-- header HTML -->
<?php
// qui dovremmo fare gli opportuni controlli sui campi inseriti
   dall'utente...
echo "<p>Ciao, {$_GET['nome']}! Hai cliccato il pulsante ";

if(isset($_GET["conferma"])){
```

descrizione	*simbolo*
addizione	`+`
incremento unitario	`++`
sottrazione	`-`
decremento unitario	`--`
moltiplicazione	`*`
divisione (floating-point)	`/`
modulo (resto di una divisione tra interi)	`%`

Tabella 14.1: Gli operatori aritmetici

```
    echo $_GET["conferma"]."</p>";
}
if(isset($_GET["AltraConferma"])){
    echo $_GET["AltraConferma"]."</p>";
}
?>
```

Se abbiamo scritto Pippo nel campo di testo del file `x.html` e abbiamo cliccato su "Conferma", restituisce: "Ciao, Pippo! Hai cliccato il pulsante Conferma".

14.5 Gli operatori

14.5.1 Gli operatori aritmetici

Gli operatori aritmetici sono gli stessi che siamo abituati a incontrare in C. L'unica differenza è che la divisione è sempre floating-point in PHP. Cosa vuol dire? Significa che, anche se si effettua una divisione tra interi, in PHP essi vengono automaticamente convertiti e considerati come double e quindi la divisione è sempre effettuata tra numeri double.

In tabella 14.1 c'è l'elenco degli operatori.

14.5.2 Gli operatori di assegnazione

Anche gli operatori di assegnazione sono analoghi a quelli del linguaggio C. Tra i tipi composti, l'unica novità è costituita dall'assegnazione con concatenazione. Come detto, la concatenazione in PHP è eseguita con l'operatore `.`: la sintassi è, quindi, analoga alle altre.

Segue la tabella (14.2) degli operatori di assegnazione:

14.5.3 Gli operatori logici

L'ultima categoria di operatori è quella degli operatori logici. Anche in questo caso, molti sono analoghi al C, ma ci sono anche notevoli differenze. Ad esempio, in PHP si possono usare come operatori le parole chiave `and` e `or`, al posto dei rispettivi simboli. Gli operatori `and` e `or` non sono però dei semplici doppioni. Infatti, essi hanno un diverso livello di

descrizione	*simbolo*	*esempio*
assegnazione	=	$a = 5
ass. con somma	+=	$a+=5
ass. con sottrazione	-=	$a-=5
ass. con prodotto	*=	$a*=5
ass. con divisione	/=	$a/=5
ass. con modulo	%=	$a%=5
ass. con concatenazione	.=	$s.="!"

Tabella 14.2: Gli operatori di assegnazione

descrizione	*simbolo*
uguaglianza (solo valore)	==
identità (valore e tipo)	===
disuguaglianza (solo valore)	!=
non identità (valore e tipo)	!==
minore / minore o uguale	< <=
maggiore / maggiore o uguale	> >=
AND	&& and
OR	|| or
NOT	!
EX-OR	xor

Tabella 14.3: Gli operatori logici

precedenza: `&&` e `||` hanno sempre la precedenza rispetto a loro. Quindi, l'espressione `(true || true and false)` corrisponde a `((true||true) && false)`.

La tabella 14.3 riporta gli operatori logici.

Facciamo qualche esempio ricordando quanto detto prima a proposito delle variabili di tipo Booleano. Il confronto `false==0` restituisce vero, mentre il confronto `0===false` restituisce falso, perché tra 0 e `false` c'è uguaglianza di valore ma non di tipo.

14.6 Controlli di flusso

14.6.1 Il costrutto if-else

I controlli di flusso servono a eseguire un programma in modo non sequenziale. La maggior parte dei controlli sono gli stessi del linguaggio C.

Uno di questi è il costrutto `if()...else`. È possibile anche inserire diversi `elseif()` dopo il primo `if` e prima dell'`else` finale. Un esempio è:

```
//definire qui la variabile $temperatura...
if ($temperatura < 0)
   echo "l'acqua e' ghiacciata";
elseif ($temperatura > 100)
   echo "l'acqua e' vapore";
else
   echo "l'acqua e' allo stato liquido";
```

14.6.2 Lo switch

Il controllo di flusso `switch` serve a testare vari possibili valori di un espressione. La parola chiave `case` introduce il valore di cui controllare l'uguaglianza, mentre dopo `default` c'è l'insieme di istruzioni da eseguire se non si ricade in nessuno dei casi elencati con i costrutti `case`. Pertanto, si può pensare lo `switch` come una forma abbreviata di una serie di `if...else`.

Ecco un esempio.

```
switch ($dado)
{
case 1:
case 3:
case 5:
   echo "Numero dispari";
   break;
default:
   echo "Numero pari";
}
```

14.6.3 I cicli while e do-while

Il ciclo `while` serve a ripetere un blocco di istruzioni fino a che una data condizione rimane vera. Quindi, le istruzioni del ciclo possono essere eseguite zero (se la condizione non è mai verificata) o più volte (anche infinite se la condizione rimane sempre vera).

Se, invece, si vuole che le istruzioni siano comunque eseguite almeno una volta, si può usare il costrutto `do-while`. Per il resto, i due costrutti sono analoghi.

Vediamo un esempio per entrambi i costrutti:

```
<h1>Tabellina del 7</h1>
<?php
$x = 1;
while ($x <= 10)
{
   echo "<p>7 * ".$x." = ".($x*7)."</p>";
   $x++;
}

/* oppure, con il do-while
do {
   echo "<p>7 * ".$x." = ".($x*7)."</p>";
   $x++;
} while ($x <= 10)
*/
?>
```

14.6.4 I cicli for e foreach

Il ciclo `for` è simile al `while`. Il blocco di istruzioni che esso contiene è ripetuto fino a che una certa condizione rimane vera. In più, si può definire una condizione di inizializzazione e un'azione da eseguire al termine dell'esecuzione di un'iterazione.

Ecco un esempio:

```
// calcolo della somma
// dei primi 10 numeri naturali
$totale = 0;
for ($i = 1; $i <= 10; $i++) {
        $totale += $i;
}
echo "Somma dei numeri [1...10] = ".$totale;
```

Il ciclo `foreach`, invece, serve a scandire tutti gli elementi di un vettore. Questo è particolarmente importante in PHP, visto che un array è una mappa e le sue chiavi possono essere delle stringhe: quindi è impossibile iterare su di essi con un normale ciclo `for` o `while`. Inoltre, anche nel caso di array che usano un indice numerico non c'è alcuna garanzia che tutti gli indici esistano. Perciò bisognerebbe testare l'esistenza di ogni elemento. Con il ciclo `foreach` tutto ciò non è necessario, perché itera su tutti e soli gli elementi definiti.

Il ciclo `foreach` è disponibile solo dalle versioni successive a PHP 4. Esso premette di accedere solo ai valori dell'array oppure anche alle chiavi corrispondenti. Le due rispettive sintassi sono:

```
foreach ($vettore as $valore) {
        //istruzioni;
};
foreach ($vettore as $chiave => $valore) {
        //istruzioni;
}
```

Un esempio del secondo tipo di sintassi è il seguente:

```
$vocab = array (
        "giallo"=>"yellow",
        "rosso"=>"red",
        "verde"=>"green");
foreach ($vocab as $key => $val)
        printf ("<p>(IT)%s = (EN)%s</p>\n", $key, $val);
```

14.6.5 Funzioni utili per le iterazioni

Molto importante per il ciclo `for` è la funzione `count()`. Essa riceve come parametro il nome di un array e ne restituisce la dimensione, ovvero il numero di elementi di un vettore. In realtà, la funzione si può applicare anche a un oggetto. In questo caso, se l'oggetto implementa l'interfaccia `Countable`, che contiene un metodo – `count()` – che restituisce il valore ritornato dalla funzione `count`. In tutti gli altri casi, `count()` restituisce 1 (0 se l'oggetto è NULL).

Se il parametro opzionale è impostato a `COUNT_RECURSIVE` o 1, l'array viene contato in maniera ricorsiva. Questo è utile per contare gli elementi presenti in un array multidimensionale.

Vediamo un esempio:

```
$colori = array (
   "giallo"=>"yellow",
   "rosso"=>"red",
   "verde"=>"green");
$numeri = array (
   "uno"=>"one",
   "due"=>"two");

$vocab = array ($colori,$numeri );

echo "<p>Numero di categorie: ".count($vocab)."</p>";
echo "<p>Numero di parole: ".count($vocab, COUNT_RECURSIVE)."</p>";
/* L'output generato è:
Numero di categorie: 2
Numero di parole: 7
*/
```

Infine, esistono funzioni che consentono di scorrere un qualsiasi array associativo o anche numerico con indici sparsi. Infatti, ogni array associativo ha un indice implicito, nascosto. Le seguenti funzioni servono a interagire con questo indice:

- `reset()`: si posiziona all'inizio, al primo elemento;
- `end()`: si posiziona all'ultimo elemento;
- `next()`: passa al prossimo elemento;
- `prev()`: passa all'elemento precedente;
- `key()`: restituisce la chiave dell'elemento corrente (o NULL, se l'array è finito);
- `current()`: restituisce il valore dell'elemento corrente;
- `each()`: restituisce un array contenente la coppia chiave-valore corrente; la chiave è accessibile agli indici `key` o 0 e il valore agli indici `value` o 1.

Un esempio in cui sono usate queste funzioni è il seguente:

```
$vocab = array (
        "giallo"=>"yellow",
        "rosso"=>"red",
        "verde"=>"green");
// elenca in ordine crescente
for (reset($vocab); $k=key($vocab); next($vocab)) {
        $val = $vocab[$k];
        printf ("<p>(IT)%s = (EN)%s</p>\n", $k, $val);
}
```

14.7 Parte non trascritta

14.8 Gestione delle sessioni

La maggior parte dei siti web attuali è composto da più pagine. Navigando, spesso si passa da una pagina ad un'altra. E' possibile che per poter visualizzare le pagine di un certo sito siano richiesti username e password, oppure venga chiesta la lingua in cui le si vuole visualizzare. Se ad ogni pagina di uno stesso sito web che un utente visita queste informazioni gli vengono richieste, dopo poco tempo gli passerebbe la voglia di continuare a navigare su questo sito. Come è possibile non fare annoiare con continue richieste identiche chi visita un sito web, o meglio, come fa questo a ricordarsi dei dati di navigazione (inseriti da un utente) passando da una pagina ad un'altra? Esistono due possibili soluzioni: usare i cookie oppure le sessioni.

L'idea che sta alla base delle sessioni è quella di associare ad una comunicazione (non ad una sola connessione) tra un determinato client ed un determinato server delle informazioni che vengano ricordate fino alla chiusura del browser o fino a quando tale sessione non viene cancellata. Il server associa ad un client un certo spazio di memoria, su questo andrà a memorizzare le informazioni ad esso associate in modo da non doverle richiedere ogni volta che questo si sposta da una pagina ad un'altra. In questo modo il server mantiene informazioni/ preferenze/ impostazioni di un client (il quale può anche essere anonimo), garantendo una navigazione 'più gradevole'. Tali informazioni vengono mantenute in memoria sul server, si crea quindi un collegamento logico tra esso ed il client, fino alla chiusura del browser e non solo per la singola connessione. Se il browser non viene chiuso e si è quindi sempre all'interno della stessa sessione, è possibile visitare altri siti e poi tornare su di uno visitato in precedenza senza dovere reinserire i dati/ riselezionare preferenze già specificate.

Per una certa iterazione tra un client ed un sito web può venir creata una sessione affinché siano ricordate determinate informazioni. Il server che gestisce il sito attribuisce a tale sessione un identificativo univoco, il quale è associato ad un cookie volatile denominato `PHPSESSID`, per questo motivo le sessioni sono in ogni caso cancellate alla chiusura del browser. Ogni volta che viene visitata una pagina di questo sito durante la sessione corrente il browser invia al server che lo gestisce questo cookie, contenente il numero che la identifica. Sul server è presente una tabella che collega ogni numero ad una sessione, alla quale sono associati a sua volta dei dati riguardanti uno specifico client.

In questo modo è possibile per un utente navigare tra diverse pagine di uno stesso sito web (che richiede autenticazione tramite user e password) senza dover inserire e trasmettere sulla rete i dati ogni volta che viene caricata una pagina. Questi dati sono inseriti dal client nella prima pagina che viene visitata. Il server salva le informazioni di autenticazione in memoria, associando ad essi un certo numero di sessione e setta il cookie volatile `PHPSESSID` sul browser. Quando l'utente passa ad un'altra pagina di questo sito, manda il cookie con l'identificativo della sessione al server. Esso prende dalla propria memoria i dati relativi ad user e password dell'utente, in modo da non doverglieli nuovamente chiedere.

Inoltre si può usare l'ID della sessione come inidce in una tabella per ricordare la selezioni di alcuni prodotti (ad esempio per un sito di e-commerce) e rendere disponibile all'utente una schermata finale riepilogativa con tutti i prodotti selezionati dalle varie pagine del catalogo online.

14.8.1 Cookie o sessioni?

Al posto di utilizzare le sessioni è possibile ottenere qualcosa di molto simile, dal punto di vista del risultato finale, con i cookie. Vi sono però delle differenze molto importanti tra questi.

L'uso dei cookie è un metodo poco efficace: ogni volta che un utente passa da una pagina ad un'altra il client manda tutti i cookie al server. Questo porta a sovraccaricare la rete perché vengono fatti passare su di essa moltissimi dati. Inoltre è un metodo non sicuro perché i cookie sono trasmessi in rete (sono quindi facilmente leggibili o alterabili da chi sniffa o controlla la rete) ed i cookie sono salvati su client (che può modificarli, per esempio inserendo il session-id di un altro utente per accedere alla sua sessione).

L'uso delle sessioni è invece più efficiente perché ogni volta che un utente passa da una pagina ad un'altra viene trasmesso un solo cookie (il session-id) e quindi la rete non viene sovraccaricata. Inoltre il meccanismo delle sessioni è un metodo parzialmente più sicuro perché i dati sono salvati sul server e mai trasmessi direttamente in rete (quindi non sono modificabili a piacere sul client). Si noti che se l'utente disabilita i cookie sul browser, il session-id viene passato come parametro di una GET (come se in ogni pagina vi fosse un form hidden) e compare quindi nella URI. A questo punto un malintenzionato potrebbe facilmente ottenere l'identificativo di questa sessione e connettersi al posto del legittimo utente ma almeno si evita che possa accedervi in futuro visto che le sessioni terminano con la chiusura del browser.

14.8.2 Funzioni per la gestione delle sessioni

Le sessioni sono disponibili dalla versione 4 di PHP, sono state pensate una serie di funzioni specifiche per poterle gestire in modo semplice.

`int session_status (void)`

E' usata per avere informazioni sullo stato attuale della sessione; possibili risultati sono `PHP_SESSION_DISABLED` (le sessioni non sono abilitate), `PHP_SESSION_NONE` (le sessioni sono abilitate ma non c'è una sessione attiva) e `PHP_SESSION_ACTIVE` (c'è una sessione attiva).

`bool session_start (void)`

Crea una sessione (o riprende quella corrente basata sull'id di sessione che viene passato attraverso una variabile GET o cookie). Va chiamata prima di inserire un altro qualsiasi tipo di output (es. echo, printf) perché modifica gli header della risposta HTTP inserendo il cookie relativo al session-id.

`$_SESSION["var"] =` *valore*

Registra la variabile `$var` nella sessione corrente assegnandole il valore indicato. Tutte le variabili attivate per la sessione in corso sono contenute all'interno dell'array associativo `$_SESSION`. La figura 14.2 contiene un esempio di uso delle variabili di sessione.

`unset($_SESSION["var"]`

Cancella la variabile `$var` dalla sessione corrente. Quando si hanno variabili molto grandi che non sono più utilizzate consente di liberare della memoria. Non restituisce alcun valore, è una semplice istruzione.

```
// da mettere in ciascuna pagina
if (session_status() !== PHP_SESSION_ACTIVE)
   session_start();
if (!isset($_SESSION['npag'])) {
   $_SESSION['npag'] = 1;
} else {
   $_SESSION['npag']++;
}
...
printf ("<p>Hai sinora visitato %s pagine.</p>", $_SESSION['npag']);
```

Figura 14.2: Esempio di uso delle variabili di sessione per contare le pagine visitate.

`void session_write_close (void)`

Termina la sessione corrente e ne salva i dati. Anche se questa funzione non viene invocata esplicitamente, l'operazione viene fatto in automatico al termine dello script. I dati di sessione vengono bloccati (lock dei dati) per prevenire scritture contemporanee (un solo script può operare su di una sessione in qualsiasi momento). Utile chiamarla quando vi sono problemi di accesso concorrente, ad esempio se si utilizzano i frameset, per ridurre il tempo di caricamento. Infatti i frame vengono caricati uno alla volta, il caricamento di un frame può iniziare solamente quando nel frame caricato in precedenza è stata chiamata questa funzione e sono state liberate le variabili di sessione. Chiamando questa funzione esplicitamente prima del termine del frame il browser può iniziare a caricare il successivo anche se quello attuale non è ancora stato caricato completamente.

`bool session_destroy (void)`

Cancella tutti i dati associati alla sessione corrente ma non cancella il session-id e non cancella le variabili di sessione nell'aray `$_SESSION`. Ritorna TRUE in caso di successo e FALSE in caso di fallimento.

`string session_id (void)`

Restituisce l'identificativo della sessione (se esiste) altrimenti la stringa vuota. Il session-id è disponibile anche tramite la costante predefinita `SID`.

`string session_id (string id)`

Ha lo stesso nome della precedente ma firma diversa; fissa l'identificativo della sessione, Da chiamarsi prima di `session_start( )` altrimenti questa riprende di default la sessione basata sull'id che viene passato attraverso una variabile GET o cookie.

`bool session_regenerate_id ( [ bool delete_old_session ] )`

Crea un nuovo identificativo della sessione ed opzionalmente cancella i dati associati alla vecchia sessione (se viene passato come paramatero True, il default è False). Inserita per evitare attacchi di tipo session fixation.

`int session_cache_expire ( [ int nuova_scadenza ] )`

Ritorna il valore corrente di scadenza della sessione (espresso in minuti, il default è 180'). Se si passa un valore in parentesi, il valore corrente è sostituito da questo; un valore piccolo (es. minore di 4') fa perdere lo stato, troppo grande sovraccarica il server ed espone a rischi di furto di sessione. Da chiamare prima di avviare la sessione.

```
// cancella le variabili della sessione
$_SESSION = array();
// cancella il cookie
$params = session_get_cookie_params();
setcookie(session_name(), '', time()-42000,
$params["path"], $params["domain"],
$params["secure"], $params["httponly"]
);
// cancella la sessione da disco
session_destroy();
```

Figura 14.3: Esempio di cancellazione di una sessione.

```
MAX_IDLE_T = 1800; // 30 minuti
if (!isset($_SESSION['idle_time']))
   $_SESSION['idle_time'] = time() + MAX_IDLE_T;
else {
   if ($_SESSION['idle_time'] < time()) {
      // cancella dati vecchia sessione + crea nuova
      session_regenerate_id (TRUE);
   }
   $_SESSION['idle_time'] = time() + MAX_IDLE_T;
}
```

Figura 14.4: Esempio di impostazione manuale del timeout di una sessione.

Capitolo 15

PHP e database

PHP supporta una vastissima gamma di DBMS(dbm, dBase, FrontBase, filePro, Informix, Interbase, Ingres II, Microsoft SQL Server, mSQL, MySQL, ODB, Oracle 8, Ovrimos, PostgreSQL, SESAM, SQLite, Sybase); permette di interfacciarsi con essi usando una delle specifiche API per il database o utilizzando un livello di astrazione PDO/PEAR DB o ancora connettendosi al database facendo uso di driver ODBC15.1.

15.1 Collegamento al DBMS

Un server DBMS può essere costituito da un insieme assai complesso di programmi software che controllano l'organizzazione, la memorizzazione e il reperimento dei dati (campi, record, archivi) in un database. Il DBMS accetta richieste di dati da parte del programma applicativo e "istruisce" il sistema operativo per il trasferimento dei dati appropriati.

Per ciascun DBMS esiste un'API dedicata (simile ma non identica per tutti i DBMS) con la quale interfacciarsi ad esso. Usare una di queste API (MySQL iface, Oracle iface, Postgres iface ...) per sviluppare un applicazione o un sito web ha come conseguenza quella di vincolarsi ad un certo DBMS in quanto se ne usano le specifiche proprie di ognuno. Qualora un giorno, per un qualsivoglia motivo, si volesse cambiare database bisognerebbe riscrivere tutto il codice utilizzando l'API relativa al nuovo DBMS.

Per poter sviluppare un applicazione in grado di interfacciarsi con qualsiasi tipo di DBMS (quindi un applicazione più portatile) è possibile passare ad un primo livello di astrazione ed usare i driver delle interfacce JDBC e ODBC. Queste contengono infatti specifiche classi che permettono di interfacciarsi con praticamente qualsiasi tipo di database, rendendo più facile eventualemnte cambiare DBMS usato per l'applicazione o il sito web.

Ad un ulteriore livello di astrazione esistono librerie di alto livello (es. PEARDB, PDO, ADO) per accedere in modo uniforme a qualsiasi DBMS. Il compito di capire quale database viene interrogato ed il modo specifico con cui interrogarlo e lasciato alle interfacce di queste librerie, di cui il programmatore deve solo conoscerne le specifiche (comuni per tutti i database). In questo caso se viene cambiato il DBMS non si vanno ad apportare modifiche sul codice dell'applicazione o del sito web.

15.2 Terminologia

Connessione. Canale tra l'interprete PHP ed uno specifico DBMS.

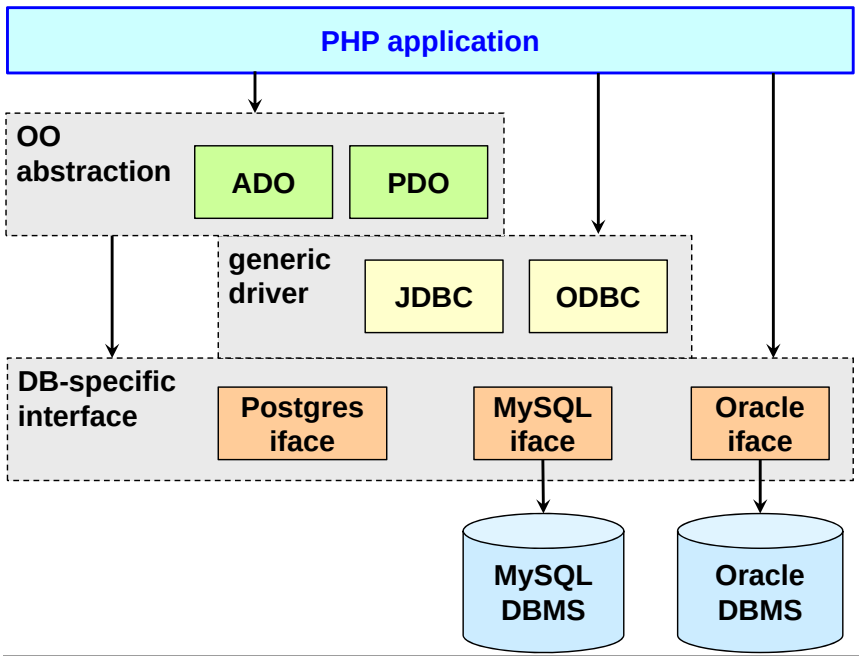

Figura 15.1: Collegamento tra PHP e DBMS.

Query. Comando SQL emesso dall'interprete PHP verso il DBMS tramite una connessione aperta.

Result set. Una tabella (virtuale) con i risultati di una query; un result set non è trasferito automaticamente all'interprete PHP e non c'è garanzia che sia memorizzato nel DB (ossia è volatile).

Riga (row). la componente elementare di un result set.

15.3 Collegamento al DBMS con PHP

Il modo con cui interfacciarsi ad un DBMS, sviluppando una pagina web o un'applicazione, facendo uso di PHP dipende dalle funzionalità che vogliamo avere:

- PEARDB/PDO
  - permettono di astrarre dai singoli DBMS;
  - adatti per programmi che devono essere molto portatili, dove non si ha la possibilità di sapere a priori con quale DB si andrà a comunicare;
  - garantiscono una bassa gamma di alternative, permettono infatti di usare solo le funzionalità comuni a tutti i DBMS;
  - non sono adatti per programmi con molti utenti o nei quali si desidera un accesso veloce in quanto vanno ad eseguire delle funzioni per capire a quale DB ci si vuole interfacciare e "adeguano" le richieste (ho perdita di tempo!).
- DB, SPECIFICHE API
  - permettono di avere a disposizione tutte le funzionalità offerte dal DBMS;
  - vincolano al database il programma/pagina web;
  - adatte quando si desidera un accesso rapido.

15.4 Aspetti principali dell'interfaccia MySQLi

L'interfaccia MySQLi è la versione migiorata (la "i" sta per improved) dell'interfaccia di basso livello MySQL, in particolare ne migliora le prestazioni quando ci sono molte chiamate al DBMS.

Tra gli aspetti più significativi di MySQLi si nota l'accesso diretto ad SQL e la doppia interfaccia (una procedurale, l'altra ad oggetti). Un altra novità chiave sono i prepared statement, non supportati da MySQL, molto importanti perchè migiorano le prestazioni e alzano il livello di sicurezza(sono usati per prevenire attacchi del tipo SQL injection!)

15.5 Funzioni PHP per la connessione al DBMS

PHP possiede numerose funzioni che sfruttano l'interfaccia MySQLi per far accesso al DBMS:

- [$con=] mysqli_connect(server, username, password, database, port, socket) ;

Crea una connessione al DBMS;

Restituisce un oggetto che rappresenta tale connessione(o NULL in caso di errore), è utile salvarla in una variabile in modo da poterla utilizzare più volte;

PORT e SOCKET possono essere non specificati, in questo caso vengono usati quelli di default.

- int mysqli_connect_errno(void) ;

restituisce il numero associato all'errore generato dalla funzione mysqli_connect (zero se non ci sono errori).

- string mysqli_connect_error(void) ;

restituisce la descrizione testuale dell'errore generato dalla funzione mysqli_connect.

- int mysqli_close($con) ;

chiude la connessione che gli viene passata come parametro.

ESEMPIO_1: Apertura e chiusura di una connessione

```
$con = mysqli_connect(
"dbserver.polito.it","lioy","123","studenti");
if ( mysqli_connect_errno() )
printf ("<p>errore - collegamento al DB
impossibile: %s<p>\n", mysqli_connect_error());
else
{
// operazioni sul DB£! \emph{(ESEMPIO\_2{*}~\ref{sec:es2})} !£
...
// rilascio della connessione al DB
mysqli_close($con);
}
```

15.6 Organizzazione di una query

Ciascuna query è composta di quattro fasi (anche dette $3+1$ perché la quarta fase è relativa alla presentazione dei risultati e quindi non è strettamente legata all'esecuzione della query):

1. preparazione: costruzione di una stringa contenente il comando SQL da eseguire;
2. esecuzione: invio del comando SQL al DBMS, generazione e ricezione del result set;
3. estrazione: lettura dei dati presenti nel result set;
4. formattazione: costruzione del codice HTML per visualizzare i risultati nella pagina.

Estrazione e formattazione solitamente sono organizzate in un ciclo (while o for). La formattazione dei dati contenuti nel result set è opzionale, dipende dal fine con cui l'applicazione a fatto la specifica query al database.

Ad esempio un server di un sito di e-commerce potrebbe eseguire una query per chiedere l'elenco dei prodotti disponibili con determinate caratteristiche. Il result set della query in questo caso necessita di formattazione, in modo da renderlo visibile ad un utente. Nel caso un client facesse il login ad un sito web, anche questa volta il server fa una query al database ma questa non necessita di formattazione, il server usa infatti il risultato della query per verificare username e password inseriti senza mostrare nulla lato client.

15.6.1 Query: preparazione

- comando SQL fisso

  esempi:

  ```
  $query = "SELECT user FROM utenti";
  $query = "INSERT INTO utenti (user, pwd) VALUES ('pippo', 'xyz')";
  ```

- comando SQL con valori variabili

  esempio:

  ```
  $query = "SELECT user FROM utenti WHERE pwd= ' ". $password ." ' " ;
  ```

  si usa l'operatore di concatenazione (.) per inserire variabili PHP nella query SQL (N.B. usare gli apici singoli intorno ad ogni stringa); rischio di attacchi SQL injection.

15.6.2 Query: esecuzione

Una volta preparata, è sufficiente inviare la query SQL al DBMS, usando la connessione attiva ed il database attualmente selezionato con la funzione

- `$result = mysqli_query($con, $query)`

  per le query di tipo INSERT, DELETE o UPDATE il lavoro è terminato ($result non è utile)

  per le query di tipo SELECT occorre estrarre ed analizzare il result set salvandolo in una variabile ($result)

  il valore di ritorno è un "riferimento" al result set attraverso cui si arriva al risultato vero e proprio che dovrà quindi essere estratto (con apposite funzioni)

- `mysqli_free_result($result)`

  Questa funzione che agisce sul DBMS per liberare memoria cancellando il result set passato come parametro. E' importante inserirla sempre quando si finisce di usare il risultato per non occupare spazio di memoria inutilmente

ESEMPIO_2: Esecuzione query

```
$result = mysqli_query($con,"SELECT * FROM 01NBE")
if (! $result)
   printf ("<p>errore - query fallita: %s</p>\n", mysqli_error($con));
else
{
   // operazioni sul result set
   . . .
   // rilascio della memoria associata al result set
   mysqli_free_result($result);
}
```

15.6.3 Query: estrazione

Una volta eseguita la query e salvato il puntatore al risultato si chiama la funzione

- `array mysqli_fetch_assoc($result)`

  riceve come parametro (`$result`) il puntatore ottenuto dalla funzione `mysqli_query` e restituisce come risultato un array associativo:

  - indici = nomi dei campi
  - valore = dato letto dal database

  Ad ogni chiamata restituisce:

  - un array coi valori della prossima riga del result set
  - NULL quando è terminato il result set

- `int mysqli_num_rows($result)`

  restituisce il numero di righe nel result set

15.6.4 Query: formattazione

Solitamente è organizzata in un ciclo (for o while, tipicamente lo stesso con cui viene estratto il risultato) in cui sono presenti funzioni di output e sintassi HTML.

Può essere organizzata in modi diversi a seconda dello scopo, ad esempio visualizzando il risultato come paragrafi (figura 15.2) o come elenco puntato (figura 15.3).

```
while ($row = mysqli_fetch_assoc($result))
{
   printf("<p>ID:%s, cognome: %s, nome: %s</p>\n",
      $row["id"], $row["surname"], $row["name"]);
}
```

Figura 15.2: Esempio di estrazione e formattazione del risultato di una query.

```
$nrow = mysqli_num_rows($result);
printf ("<p>Trovati %d record:</p>\n<ul>\n", $nrow);
for (i=1; i<=$nrow; i++)
{
   $row = mysqli_fetch_assoc($result);
   printf("<li>record %d = ID:%s, cognome: %s</li>\n",
      $i, $row["id"], $row["surname"]);
}
echo "</ul>\n"
```

Figura 15.3: Esempio di estrazione e formattazione del risultato di una query.

15.7 Prepared statement

I prepared statement sono query SQL con struttura fissa ma dati variabili (di input e/o di output); MOLTO utile per prevenire attacchi SQL injection in quanto il tipo di variabile che ci aspettiamo venga inserita è specificato attraverso delle particolari funzioni. Ad esempio una stinga con istruzioni SQL per modificare un DB inserita in un campo dove ci si attende un numero non viene accettata come valore valido.

Una serie di funzioni permettono di creare e gestire iterazioni con il DBMS attraverso prepared statement

- dati vengono associati alle variabili tramite

  `mysqli_stmt_bind_param (stmt, types, vars)`

  riceve come parametri:

  stmt uno statement creato dalla funzione `mysqli_prepare`;

  types una stringa che specifica ordinatamente il tipo delle variabili (i = integer, d = double, s = string);

  vars l'elenco delle variabili (una per ogni punto interrogativo della query).

- esecuzione di un prepared statement:

  `mysqli_stmt_execute (stmt)`

  esegue lo statement passato come parametro.

- associazione del risultato (tutti i campi!) a variabili:

  `mysqli_stmt_bind_result(stmt, vars)`

  associa a ciascun campo del risultato trovato dall'esecuzione dello statment la/le variabile vars.

```
$stmt = mysqli_prepare ($con, "SELECT surname FROM 01NBE WHERE id=?");
mysqli_stmt_bind_param ($stmt, "i", $matricola);
$studenti = array (12345, 11223, 54321);
foreach ($studenti as $matricola)
{
   mysqli_stmt_execute($stmt);
   mysqli_stmt_bind_result($stmt, $cognome);
   mysqli_stmt_fetch($stmt);
   printf ("matr. %d = %s\n", $matricola, $cognome);
}
mysqli_stmt_close($stmt);
```

Figura 15.4: Esempio di prepared statement.

- caricamento dei valori del risultato nelle variabili:

  `mysqli_stmt_fetch(stmt)`

  carica i valori del risultato dello statement che viene passato come parametro nelle variabili specificate nella funzione mysqli_stmt_bind_result.

- per terminare l'uso di un prepared statement:

  `mysqli_stmt_close(stmt)`

  per dichiarare che lo statement (che viene passato come parametro) non serve più, è quindi possibile rimuoverlo dalla memoria.

- funzioni utili per gestire gli errori

  - `int mysqli_stmt_errno(stmt)`
    restituisce il numero dell'errore generato dallo statement passato come parametro.
  - `string mysqli_stmt_error(stmt)`
    restituisce la descrizione dell'errore generato dallo statement che viene passato come parametro.

Capitolo 16

Sicurezza web

16.1 Introduzione

Il termine "sicurezza" è un termine difficile da identificare e definire univocamente. Ogni persona ha infatti un proprio concetto di sicurezza inteso come il rischio che è disposto a correre per ottenere qualcosa. Ad esempio alcune persone non mangiano le cozze perché sono ritenute "gli spazzini del mare" e quindi portatrici di malattie; altre persone per lo stesso motivo fanno bollire le cozze prima di mangiarle; altre ancora semplicemente decidono di ignorare i rischi e le mangiano crude. Chi ha ragione? Nessuno. Ognuno ha un diverso grado di accettazione dei rischi e quindi anche la soglia minima di sicurezza che è disposto a correre cambia. Allo stesso modo nei sistemi informativi non si può affermare di volere un'applicazione "sicura": bisogna specificare esattamente quali proprietà di sicurezza si considerano necessarie in base al proprio business. Di seguito verrà declinata la sicurezza nelle sue principali proprietà.

16.2 Proprietà di sicurezza

16.2.1 Autenticazione (della controparte)

Quando due parti vogliono comunicare tra loro, è buona norma verificare l'identità dell'interlocutore. Il processo di verifica si definisce *autenticazione* (e non autentificazione!). Ad esempio nella figura 16.1 la signorina Barbara finge di essere Alice per poter accedere al sistema, il quale non le crede e le chiede di autenticarsi, ovvero di dimostrare la propria identità.

16.2.2 Mutua autenticazione

Si parla di *mutua autenticazione* quando entrambe le parti verificano l'identità dell'interlocutore reciprocamente, come illustrato in figura 16.2 in cui Barbara desidera accedere al sito della propria banca: il sito richiede lo username e la password della ragazza per essere sicuro che sia proprio lei e non un impostore, allo stesso modo anche Barbara vuole essere sicura di comunicare con il vero sito della banca e non con un server fittizio (che ha copiato l'aspetto grafico del vero server). E' importante notare che la grafica di una pagina web è molto facile da copiare, quindi all'apparenza non si può distinguere un sito ufficiale da uno fittizio, ma bisogna controllare che il nome host o l'indirizzo siano attendibili.

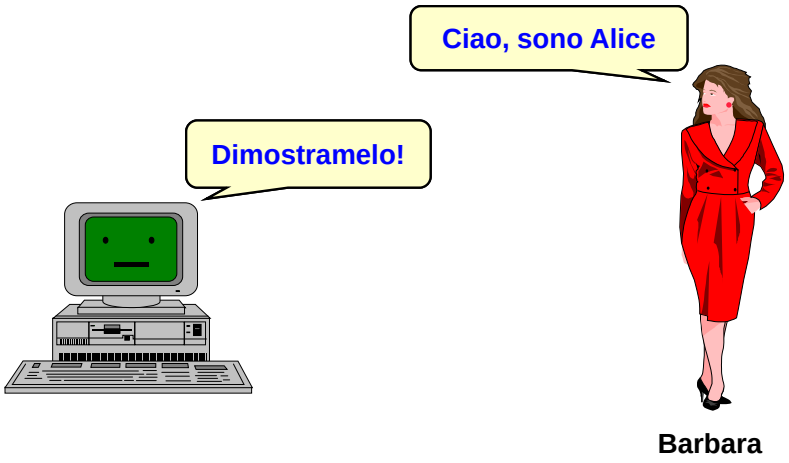

Figura 16.1: Autenticazione (semplice) della controparte.

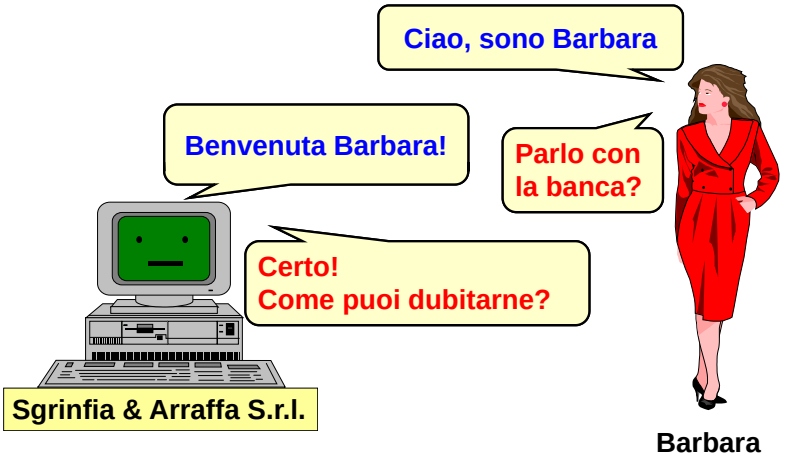

Figura 16.2: Mutua autenticazione delle controparti.

16.2.3 Autenticazione (dei dati)

Autenticazione non significa solo identificare chi vuole comunicare, ma anche verificare l'autenticità dei dati scambiati (figura 16.3). Ad esempio con il fine di frodare gli utenti è possibile creare delle fake mail: sono e-mail che sembrano essere attendibili all'apparenza, ma sono inviate per ottenere dal destinatario informazioni sensibili che vengono poi usate illegittimamente. Tramite mail il collegamento non è in tempo reale (come invece accade attraverso l'uso di username e password per l'accesso ad un sito), ma è un collegamento asincrono e per questo non si può avere certezza dell'identità del mittente.

16.2.4 Autorizzazione

L'autenticazione può essere un mezzo di verifica dei permessi ad entrare in determinate aree o svolgere specifiche azioni. Questo controllo degli accessi viene definito come *autorizzazione*. L'esempio illustrato in figura 16.4 mostra la signorina Barbara che reclama la macchina di Alice. Il sistema deve verificare se Barbara è stata autorizzata da Alice a guidare la sua auto. Attraverso l'autenticazione quindi si può accedere a determinati oggetti e svolgere operazioni su di essi, ma solo se si è autorizzati.

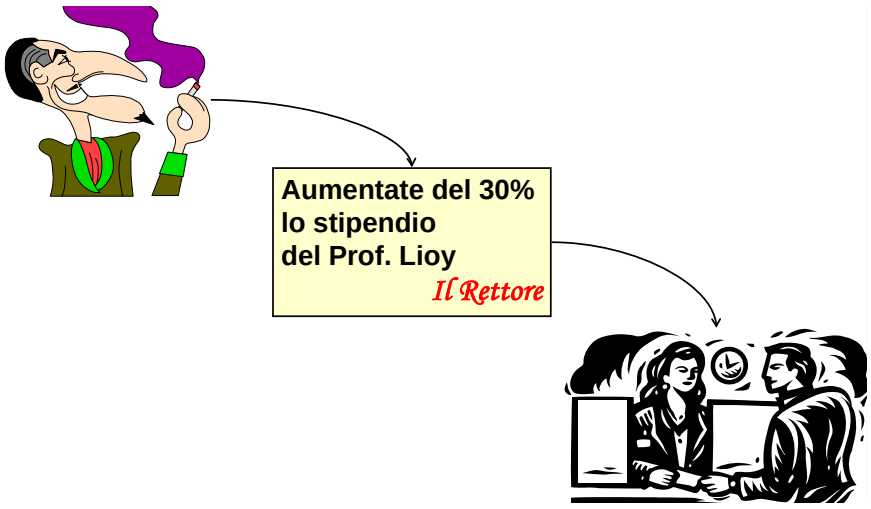

Figura 16.3: Autenticazione dei dati.

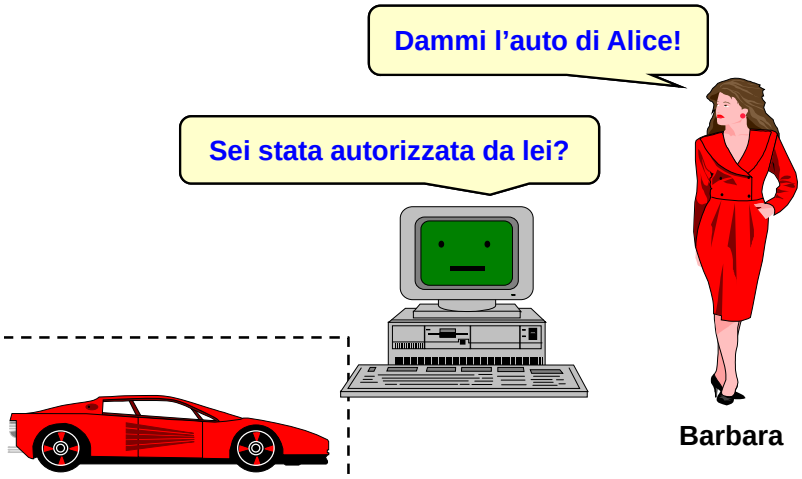

Figura 16.4: Autorizzazione (controllo accessi).

16.2.5 Riservatezza

Come per l'autenticazione, anche la *riservatezza* è un concetto generico se non si specifica cosa deve essere riservato.

Riservatezza delle trasmissioni

Si parla di *riservatezza delle trasmissioni* (figura 16.5) per indicare che la comunicazione in rete tra due utenti deve essere riservata: le informazioni che i due utenti si scambiano non possono essere lette da terzi. La riservatezza delle trasmissioni si applica in generale a qualunque tipo di comunicazione, che essa avvenga tra due pari o tra un client e un server.

16.2.6 Riservatezza dei dati

Si esprime come *riservatezza dei dati* (figura 16.6) la modalità di protezione dei propri dati al di fuori di una trasmissione in rete. Ad esempio i dati presenti sul proprio computer, le ricerche che si effettuano su Internet o la posizione ottenuta tramite l'uso di dispositivi mobili sono dati riservati e come tali non possono essere decifrati da terzi. Questo argomento

Figura 16.5: Riservatezza delle trasmissioni.

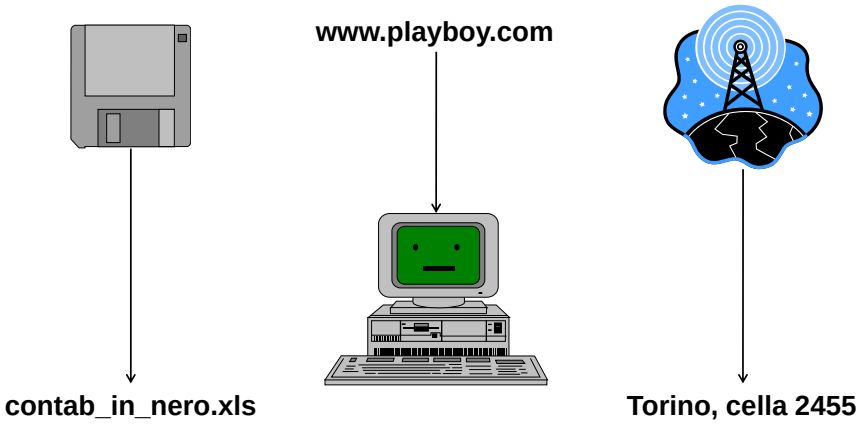

Figura 16.6: Riservatezza (di dati, azioni e posizione).

è di notevole importanza per via dell'aumento dei furti di portatili e altri dispositivi mobili. Esistono quindi delle norme e procedure per proteggere i dati e le azioni. Un esempio è il Decreto Legge 155/2005, meglio noto come "Legge Pisanu", secondo cui tutti gli Internet Provider sono obbligati a tenere memoria per 7 anni degli accessi dei propri utenti, rendendoli quindi tracciabili in caso di indagine. Dunque è necessario identificare vari livelli di riservatezza delle azioni e decidere in base alla tipologia se devono essere protette, pubbliche o visualizzabili solo dalle forze dell'ordine.

16.2.7 Integrità

Modifica dei dati

Quando si comunica in rete è doveroso controllare se le informazioni sono state alterate durante la trasmissione e quindi se è nata una discrepanza tra il messaggio inviato e quello ricevuto. Nell'esempio in figura 16.7 il messaggio, contenente la somma da versare di 1000 Euro, durante la trasmissione in rete viene intercettato e modificato, aumentando la cifra a 10000 Euro. Se non viene fatto alcun controllo sull'integrità del pacchetto, la cifra che verrà versata sarà di 10000 Euro anzichè di 1000 Euro come deciso dal mittente.

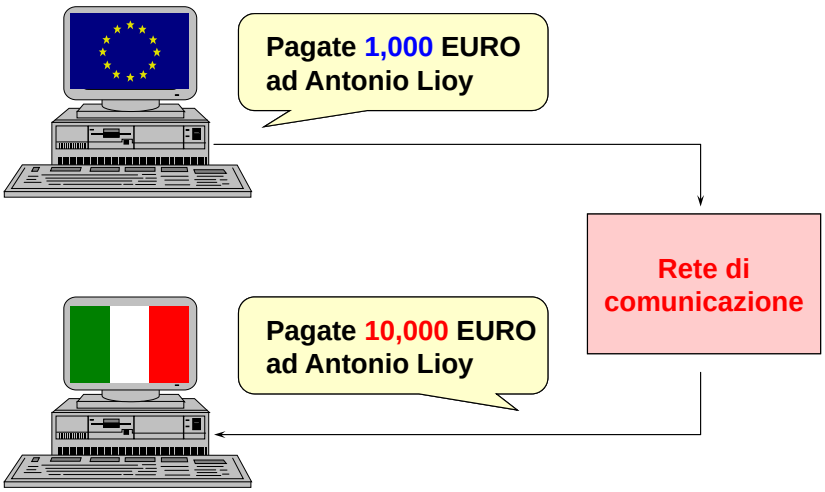

Figura 16.7: Integrità (modifica dei dati).

Cancellazione dei dati

Esiste una forma molto più radicale di modifica dei dati: la *cancellazione*. La cancellazione dei pacchetti potrebbe sembrare un problema di lieve entità, ma al contrario è un'azione subdola: un dato cancellato è difficile da individuare senza un controllo approfondito e di conseguenza il mittente non si preoccupa di doverlo inviare nuovamente. Nella figura 16.8 si può osservare che il messaggio di trasferimento della somma di 2500 Euro al conto della Rolex è stato cancellato durante il suo passaggio in rete. Se la transazione non è controllata in altri modi, la Rolex non si accorgerà della mancanza di un pagamento.

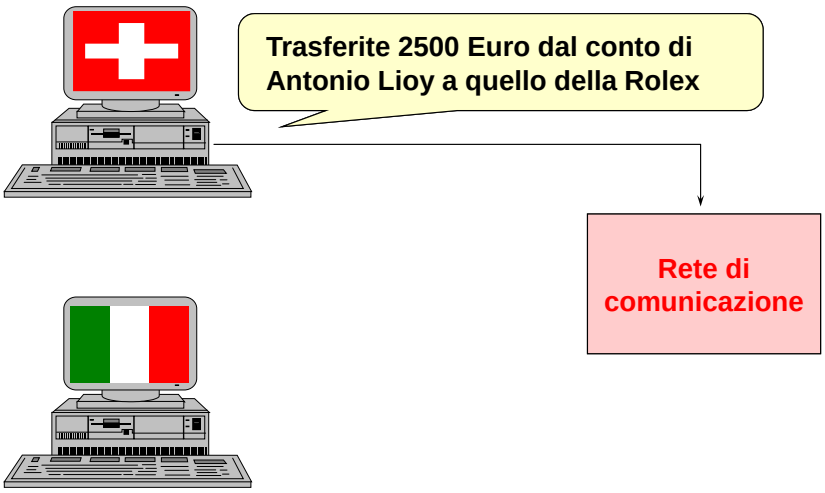

Figura 16.8: Integrità (cancellazione dei dati).

Replay attack

Un caso opposto alla cancellazione ma altrettanto dannoso è la ripetizione di dati. Si parla di *replay attack* quando il dato originale circolante in rete viene replicato e inviato più volte. Nell'esempio in figura 16.9 il messaggio, contenente un ordine di trasferimento di 1000 Euro, durante la trasmissione in rete viene intercettato, replicato e quindi ritrasmesso più volte allo stesso destinatario. In questo modo, se non ci sono controlli sull'integrità del pacchetto, l'operazione di versamento verrà effettuata un numero di volte pari alla quantità di copie

ricevute. Per tutelarsi da questo tipo di attacco sarebbe bene associare al versamento un codice univoco, in questo modo la somma viene versata solo una volta perché tutti i messaggi replicati verrebbero scartati. Il vero problema però risiede nel fatto che semplici accorgimenti come questo non vengono adottati solo perché si ignora l'esistenza di questo tipo di attacchi.

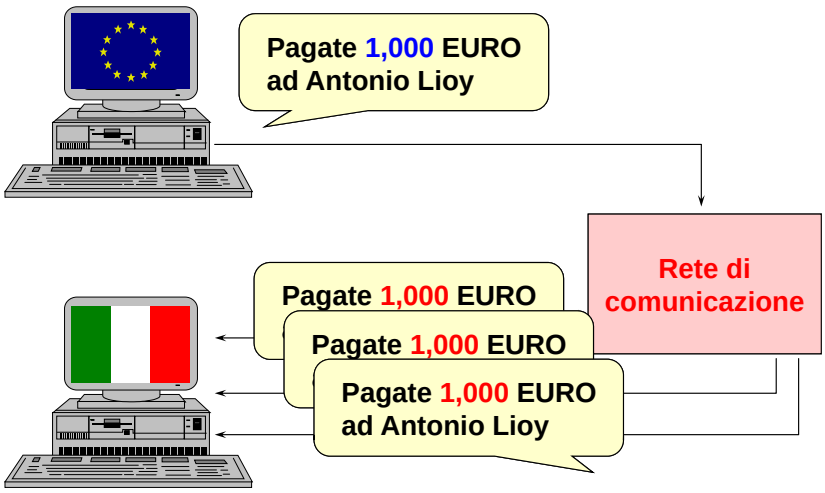

Figura 16.9: Attacco di tipo replay.

16.2.8 Altre proprietà

Esistono altre proprietà oltre a quelle descritte precedentemente. Una di queste è il *non ripudio*: il mittente/ricevente non deve poter negare l'invio/la ricezione delle informazioni e il loro contenuto. Un cliente, ad esempio, ordina via mail 10 prodotti a un artigiano, il quale appena ricevuto l'ordine comincia a produrli. Giunta l'ora della consegna il cliente ripudia la mail con l'ordine, negando di averlo effettuato. Tramite la proprietà di non ripudio della sorgente si è in grado di risalire al mittente dei dati in una transazione, in questo caso si può provare che il cliente ha davvero effettuato l'ordine. Al contrario esiste anche il non ripudio della destinazione attraverso cui si può provare che i dati sono arrivati ad uno specifico destinatario.

Un'altra proprietà importante è la *disponibilità*: non deve esserci nessuna interruzione nell'accesso ai dati. Il sistema deve essere in grado di eseguire un servizio proprio quando l'utente ne ha bisogno, non in un secondo momento. Alcuni tipi di attacco fanno in modo che il sistema richiesto non sia disponibile, così l'utente è bloccato e non può effettuare operazioni per lui rilevanti.

Una terza proprietà utile è la *tracciabilità*: la registrazione di tutte le operazioni effettuate sui dati, compresa l'identità di coloro che le eseguono (lettura, scrittura, invio, ricezione, modifica, cancellazione). In questo modo si è in grado di risalire a operazioni passate e ricostruire le vicende con notevole certezza.

16.3 Attacchi informatici

16.3.1 Atacco Denial-of-Service (DoS)

Il *Denial-of-Service*, o *DoS*, è una tipologia di attacco informatico che mira a interrompere l'erogazione di un servizio tenendo impegnato il nodo che lo fornisce. Può essere ottenuto in vari modi, a seconda del servizio che si vuol rendere indisponibile.

Ad esempio per bloccare la posta elettronica di uno specifico utente si può inviargli una grossa quantità di messaggi di posta (allegando file di notevoli dimensioni) fino a intasare la casella del destinatario; in questo modo, i messaggi importanti che il destinatario attende non verrebbero consegnati sino a quando l'utente non provvede a cancellare i messaggi di attacco.

Un altro attacco DoS molto noto è il *ping flooding* (o *ping bombing*). Come è noto il comando ping misurare il tempo impiegato da uno o più pacchetti ICMP per raggiungere un nodo e tornare indietro. Si invia un pacchetto ICMP echo request e si rimane in attesa di un pacchetto ICMP echo reply in risposta. L'attacco consiste nell'aumentare la dimensione del pacchetto ICMP e inondare un nodo con un numero elevato di richieste inviate senza attendere le corrispondenti risposte. Così facendo la vittima dedicherà tutte le risorse a rispondere, bloccando l'esecuzione delle altre richieste.

Il *SYN attack* è un altro tipo di attacco DoS. E' noto che per aprire un canale TCP è necessario effettuare il *three-way handshake*. L'attaccante, dopo aver mandato il SYN e ricevuto il SYN-ACK dal server, invia un nuovo SYN (invece di rispondere con ACK, come prevede il protocollo) occupando un'altra riga della tabella delle connessioni. Una volta che questa tabella viene saturata, il server è costretto a rifiutare nuove richieste, impedendo quindi ad altri client di stabilire una connessione.

La criticità di questi attacchi consiste nel fatto che non esiste nessuna contromisura definitiva, è possibile solamente attenuare i sintomi con un continuo monitoraggio delle risorse. Se ad esempio la tabella delle connessioni ha spazio per 100 righe e normalmente effettua 50 collegamenti, se viene superata la soglia di 80-90 scatterà l'avviso di un possibile attacco.

16.3.2 Distributed Denial-of-Service (DDoS)

Una modalità più violenta di DoS è il *Distributed Denial-of-Service*, o *DDOS* (figura 16.10). L'attaccante installa all'interno di alcune vittime un programma invisibile detto *daemon*, *zombie* o *BOT*, che può essere controllato da remoto, formando una *BOTNET*: una rete di robot "schiavi". Di conseguenza chi architetta l'attacco non lo sferra direttamente, ma agisce tramite dei nodi intermedi detti *master* a lui subordinati. Ogni master a sua volta controlla un certo numero di zombie che agiscono sotto il suo comando. L'attaccante quindi fornisce ai nodi master l'indirizzo IP della vittima e il metodo di attacco (es. ping flooding, SYN attack), poi si scollega per non essere rintracciato. Ogni master attiverà i suoi daemon, migliaia di calcolatori, che attaccheranno simultaneamente la vittima fino a mandarla fuori uso.

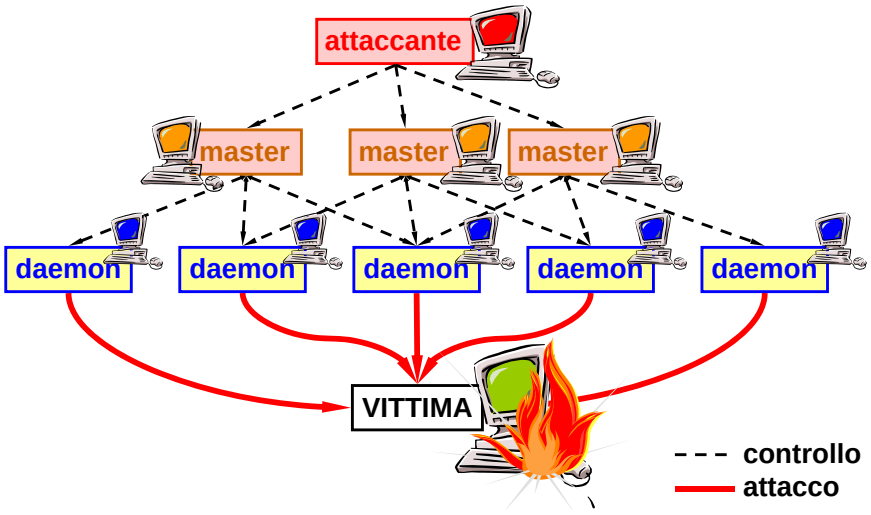

Figura 16.10: Distributed denial-of-service.

16.4 Contromisure difensive

Quali politiche adottare per tutelarsi da questi attacchi? Le seguenti misure si dovrebbero impiegare in modo ciclico per essere sempre all'avanguardia nel campo della sicurezza:

Avoidance – evitare di essere attaccati adottando contromisure difensive (es. firewall, VPN, PKI) ricordanbdo però che ogni difesa può essere raggirata o diventare inefficace;

Detection – tenere sotto controllo le risorse per individuare il prima possibile un attacco proveniente dall'esterno o dall'interno della rete (es. tramite un IDS o un network monitor);

Investigation – studiare ogni attacco subito, sia per denunciare il fatto alle autorità competenti sia per capire come mai l'attacco abbia avuto successo, al fine di migliorare le proprie difese.

In pratica è un'altra versione del classico ciclo di miglioramento continuo (o *ciclo di Deming*), composto da quattro fasi:

Plan – pianificare le attività necessarie a raggiungere un obiettivo desiderato;

Do – realizzare le attività pianificate;

Check – controllare i risultati delle attività svolte;

Act – agire o reagire in base ai risultati ottenuti e soprattutto in base ai problemi verificatisi.

16.4.1 La crittografia

Tra le varie tecniche di salvaguardia delle informazioni c'è la *crittografia*: tecnica per offuscare un messaggio e renderlo incomprensibile a persone non autorizzate a leggerlo. Il mittente utilizza un algoritmo di cifratura combinato a una chiave per crittografare il messaggio in chiaro e lo spedisce. Il destinatario, ricevuto il crittogramma, usa un algoritmo congiunto a una seconda chiave per decrittografare il messaggio e renderlo nuovamente leggibile. La prima chiave è la "ricetta" di alterazione dei dati, la seconda è l'antidoto per ricomporli. In questo modo, chi non possiede entrambe le chiavi non potrà mai leggere i dati. Esistono due tipi di crittografia in base al tipo di chiave.

La crittografia simmetrica

La *crittografia simmetrica* (detta anche *crittografia a chiave segreta*) è la forma di crittografia più semplice e più antica, infatti veniva già usata in passato per scambiarsi messaggi segreti. La sua semplicità è dovuta al fatto che le due chiavi di cui è composta coincidono, quindi esiste un'unica chiave che serve sia per cifrare sia per decifrare. Nasce spontaneo però un dubbio: come fa il mittente a dare la chiave al destinatario senza che cada nelle mani sbagliate? Esistono molti algoritmi a crittografia simmetrica che si diversificano in base alla lunghezza della chiave, più è lunga la chiave più è difficile decifrare il crittogramma. I due algoritmi più significativi sono:

- *DES* (Data Encryption Standard)], il più diffuso in passato e ormai divenuto obsoleto, la chiave lunga 56 bit è considerata troppo corta e si può forzare in poco tempo provando tutte le combinazioni;
- *AES* (Advanced Encryption Standard), il più sicuro, è un algoritmo di cifratura a blocchi che ha sostituito DES (e la sua evoluzione 3-DES).

La crittografia asimmetrica

Inventata verso la metà degli anni '80, la *crittografia asimmetrica* è una tecnica molto complessa in quanto la chiave di cifratura è diversa dalla chiave di decifratura. Entrambe le chiavi sono generate per funzionare solo in coppia: tutto ciò che viene cifrato da una, può essere decifrato solo dall'altra e viceversa. Scendendo nel dettaglio, ogni utente possiede quindi due chiavi. Una chiave pubblica, da diffondere il più possibile, sarà in grado di decifrare solo i messaggi creati con la sua chiave gemella, la chiave privata, che invece deve essere segreta e ben custodita. La chiave pubblica può anche cifrare un messaggio che verrà decifrato esclusvamente dalla chiave privata abbinata. Per fare un esempio, una ragazza scrive un messaggio, lo cifra con la sua chiave privata e lo invia al fratello. Dato che il messaggio è cifrato, se qualcuno lo intercetta nella rete non lo può leggere. Ricevuto il crittogramma, il fratello lo decifra con la chiave pubblica della sorella che, in quanto pubblica, è in suo possesso. Volendo rispondere al messaggio ricevuto, il fratello cifra la risposta con la chiave pubblica della sorella e la invia. L'unico che può decifrare il messaggio di risposta è il possessore della chiave privata, cioè la sorella. In sostanza, se si vuole che il messaggio sia autentico lo si cifra con la chiave privata dell'autore, in questo modo il destinatario lo decifra con la chiave pubblica dell'autore ed è certo dell'autenticità e integrità dei dati (es. firma digitale). Se si vuole che il messaggio rimanga segreto lo si cifra con la chiave pubblica del destinatario, così facendo soltanto lui potrà decifrare e leggere il messaggio tramite la sua chiave privata (es. riservatezza senza segreti condivisi). Gli algoritmi a chiave pubblica si basano sull'intrattabilità di un problema matematico, i principali sono i seguenti:

- *RSA* (Rivest - Shamir - Adleman) è il piu diffuso al mondo, si basa sulla fattorizzazione in numeri primi di numeri molto grandi e permette sia la firma digitale sia la riservatezza senza segreti condivisi;
- *DSA* (Digital Signature Algorithm), brevettato dal NIST (agenzia del governo americano), eleva a potenza due valori (di migliaia di bit) e fa il logaritmo del risultato, viene usato solo per la firma digitale in quanto il governo americano non riconosce la riservatezza dei cittadini.

La distribuzione delle chiavi per la crittografia asimmetrica può rappresentare un problema. La chiave privata, in quanto tale, deve rimanere segreta e ben custodita, quella pubblica deve essere divulgata il più ampiamente possibile, ma come è possibile garantire la corrispondenza tra la chiave pubblica e l'identità della persona? Esistono due soluzioni a questo problema:

1. scambio di chiavi out-of-band, cioè al di fuori della rete;
2. distribuzione delle chiavi tramite un certificato d'identità digitale, un documento elettronico che attesta l'associazione univoca tra una chiave pubblica e l'identità di un soggetto.

La crittofrafia asimmetrica, in particolare la riservatezza senza segreti condivisi, può essere a sua volta uno strumento per scambiare le chiavi segrete usate nella crittografia simmetrica. Ad esempio se X e Y in figura 16.11 vogliono comunicare tramite un algoritmo simmetrico, sorge però l'incertezza su come scambiarsi le chiavi segrete. Tramite un algoritmo asimmetrico X cifra la chiave segreta K con la chiave pubblica di Y, in questo modo la può inviare in sicurezza a Y. Appena ricevuto il crittogramma, mediante la propria chiave privata Y può decifrare la chiave segreta K che gli serve a decrittografare il messaggio. Se l'algoritmo di crittografia è stato progettato correttamente e le chiavi sono tenute segrete, il sistema è impenetrabile.

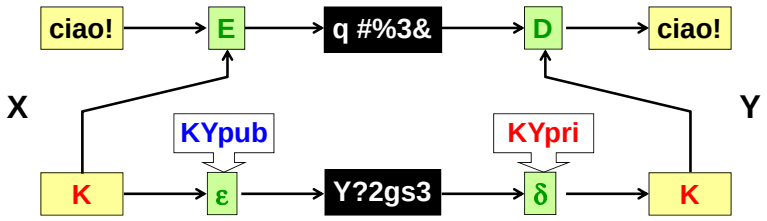

Figura 16.11: Scambio di una chiave segreta mediante algoritmi asimmetrici.

Tuttavia nuovi tipi di attacchi impiegano algoritmi iterativi che sperimentano varie chiavi finché non trovano quella corretta. Il numero di tentativi è pari a 2^{Nbit} dove Nbit è la lunghezza in bit della chiave. Per ottenere maggiore difesa bisogna dunque usare chiavi lunghe e ideare tecniche sempre più complesse se si vuole mantenere un alto livello di sicurezza, in quanto il progresso tecnologico arriverà a far vacillare ogni tipo di contromisura. Al giorno d'oggi con l'attuale potenza di calcolo si può considerare sicura una chiave con almeno 100 bit per chiavi simmetriche e 1024 bit per chiavi asimmetriche.

La crittografia a curve ellittiche

La *crittografia a curve ellittiche* o *ECC* (Elliptic Curve Cryptography) è una nuova tipologia di crittografia a chiave pubblica più complessa e robusta. Invece di lavorare con l'aritmetica modulare, lavora sulla superficie di una curva ellittica 2D, in questo modo si possono usare chiavi più corte (circa 1/10) e avere comunque un grado di sicurezza molto elevato. Questo algoritmo viene usato molto per sistemi embedded, consente la firma digitale (ECDSA) e la cifratura asimmetrica(ECIES).

16.4.2 Il digest

Poiché la crittografia impedisce di leggere i dati ma non garantisce la loro integrità, c'è bisogno di aggiungere un passaggio in più alla normale cifratura asimmetrica: il *digest*. Un

digest (letteralmente riassunto) è una sintesi dei dati ottenuta tramite algoritmi di hash, in pratica è una stringa che identifica univocamente quei dati. Ad esempio in una comunicazione in rete sia il mittente sia il ricevente calcolano il digest dei dati trasmessi, se i due digest coincidono si ha la garanzia che i dati non sono stati violati. L'enorme vantaggio del digest rispetto agli algoritmi crittografici è che l'algoritmo di hash è più veloce, meno dispendioso e irreversibile, cioè non si può risalire al messaggio originale. Gli algoritmi di hash più utilizzati sono:

- MD5: produce un digest di 128 bit partendo da un messaggio di lunghezza arbitraria, superato da algoritmi più efficienti ma ancora ampiamente utilizzato;
- SHA-1: produce un digest di 160 bit partendo da un messaggio di lunghezza variabile, è il più diffuso della famiglia SHA nonostante sia meno sicuro dei suoi successori;
- SHA-2 (SHA-224, SHA-256, SHA 384, SHA-512): formato da 4 sottocategorie ognuna delle quali produce un digest di lunghezza in bit pari al numero indicato nella loro sigla, non è ancora stata provata la sua vulnerabilità.

Il keyed-digest

Il digest da solo non è abbastanza sicuro, quindi si è deciso di affiancarlo a una chiave condivisa tra mittente e ricevente. Nella pratica il mittente crea il documento e ne calcola il *keyed-digest*, cioè il digest combinato alla chiave segreta condivisa, dopodiché li invia al destinatario. Chiunque intercetti il documento per modificarlo non può modificare il keyed-digest perché non è a conoscenza di tale chiave. Il destinatario calcola il keyed-digest sul documento ricevuto con la chiave segreta in suo possesso e lo confronta con il keyed-digest arrivato dal mittente. Se i due coincidono allora il documento è integro e autentico.

La firma digitale

Dato che gli algoritmi crittografici devono eseguire calcoli lunghi e complessi, è nata l'idea di sfruttare le piccole dimensioni del digest cifrando questo anzichè il documento originale. In questo modo si crea uno strumento che soddisfa tutte le proprietà di sicurezza: la *firma digitale*. Al lato pratico il mittente crea il documento, calcola il message digest e lo cifra con la propria chiave privata, al termine di questi passaggi ottiene la firma digitale. A questo punto il documento e la firma vengono inviati. Il ricevente quindi decifra la firma tramite la chiave pubblica del mittente, ottenendo così il message digest, dopodiché applica la funzione di hash al documento e confronta il risultato con il message digest appena decifrato: se i due coincidono l'autenticità e l'integrità del documento sono garantite.

16.4.3 I certificati

Un *certificato a chiave pubblica* è una struttura dati che serve a legare in modo sicuro una chiave pubblica al suo proprietario (persona fisica, società, computer, etc). Il certificato è autenticato dall'emettitore, l'autorità di certificazione (CA), attraverso la firma digitale, cioé cifrato con la chiave privata dell'autorità. Un certificato ha una data di scadenza oltre la quale non è più valido e può essere revocato sia su deciosione dell'utente sia dell'emettitore.

Nell'immagine viene mostrata la struttura di un certificato digitale X.509:

- versione;
- numero seriale;
- algoritmo di firma;
- emettitore: C (Country), O (Organization), OU (Organization Unit);
- validità: data di emissione, data di scadenza;
- soggetto: C (Country), O (Organization), CN (Common name);
- informazioni sulla chiave pubblica del soggetto;
- firma del certificato;

I certificati sono regolati da una infrastruttura tecnica e organizzativa detta PKI (Public-Key Infrastructure), composta dalle autorità di certificazione e dai loro sistemi riservati alla creazione, distribuzione e revoca dei certificati a chiave pubblica. Un certificato può essere revocato prima della scadenza naturale su richiesta del titolare (subject) o autonomamente dall'emettitore (issuer), ad esempio se si scopre che la chiave privata è stata compromessa. Per questa ragione un utente dovrebbe accertarsi che il certificato sia valido, cioé firmato da una CA e non scaduto, e che non sia stato revocato (ad esempio attraverso un Relying Party).

Per tutelarsi quindi l'utente può consultare il CRL (Certificate Revocation List), un elenco di certificati revocati firmato dall'autorità di certificazione o da un delegato. In alternativa l'utente può affidarsi all'OCSP (On-line Certificate Status Protocol), un protocollo che permette di interrogare un server riguardo la validità di un singolo certificato, operazione più veloce rispetto all'utilizzo del CRL. In figura viene mostrata la struttura di una CRL X.509:

- versione;
- algoritmo di firma;
- emettitore: C (Country), O (Organization), OU (Organization Unit);
- data ultimo aggiornamento;
- elenco dei certificati revocati: identificativo del certificato, data della revoca;
- firma del certificato;

Finora si è dato per scontato che l'autorità di certificazione sia inviolabile e quindi una fonte sicura, ma purtroppo non è così. Come si fa a verificare che un certificato a chiave pubblica firmato da una CA sia autentico? Occorre il certificato della sua chiave che è stato firmato dalla CA di livello superiore, innsescando così una catena di sospetti che si fermerà solo interpellando la massima autorità della scala gerarchica. La CA radice è quella che non è stata firmata da nessun'altra CA e la cui chiave privata è usata per firmare i certificati sotto di lei che ereditano così la sua attendibilità.

16.4.4 Metodologie di autenticazione

I fattori attraverso cui un utente può autenticarsi sono principalmente tre: qualcosa che sa (ad esempio una password o un PIN), qualcosa che possiede (ad esempio una carta magnetica) e qualcosa che è (ad esempio un'impronta digitale). Questi diversi metodi possono essere combinati tra loro per passare da una semplice autenticazione a una vera e propria identificazione.

Una *password* è una parola chiave segreta che l'utente custodisce e utilizza per accedere a delle risorse informatiche in modo esclusivo. L'autenticazione tramite password normalmente avviene quando un utente vuole accedere al servizio a lui riservato di un server, il quale chiederà all'utente di fornire l'identificativo e la password in suo possesso. L'autenticazione tramite chiave di accesso è la più semplice per l'utente, però la sua segretezza comporta alcuni svantaggi. La password deve essere conservata in modo opportuno sia dall'utente sia lato server per evitare che venga scoperta, inoltre bisogna assicurarsi che non venga indovinata o letta durante una trasmissione. Una password funziona quindi solo nel caso in cui sia impossibile da indovinare e ben conservata. Per soddisfare la prima condizione si possono seguire alcuni suggerimenti:

1. variare all'interno della parola caratteri maiuscoli, minuscoli, cifre e caratteri speciali;
2. scegliere parole lunghe, cioè con più di 8 caratteri;
3. optare per parole non presenti nel dizionario;
4. cambiare la password frequentemente, ad esempio ogni 12 mesi;
5. non usare password, anche se il suo utilizzo è ormai inevitabile.

Per adempiere al secondo punto, ovvero la buona conservazione, il primo accorgimento è quello di non memorizzare mai la password in chiaro. Si potrebbe optare per cifrare la password, ma questo è sconveniente in quanto il server dovrebbe conoscere la chiave in chiaro. Si potrebbe anche pensare di memorizzare un digest della password, ma anche in questo caso non conviene perché si presta ad attacchi "dizionario" (un ciclo for che applica la funzione di hash a tutte le parole del dizionario finchè non trova la parola che combacia con il digest) velocizzabili tramite rainbow table. La contromisure più efficace è quindi introdurre un *salt*: una sequenza imprevedibile di bit. Per memorizzare una password si genera un salt imprevedibile, lungo e diverso per ogni utente, la cui efficacia aumenta se vengono impiegati caratteri poco usati o di controllo. Si calcola quindi il digest della password con il salt, il risultato verrà memorizzato al posto della sola password insieme all'id dell'utente e al salt utilizzato. In questo modo si evita di avere digest uguali per utenti che hanno password uguale, in più si complicano notevolmente i dictionary attack poichè per ogni bit di salt è necessario il doppio dei tentativi. Ad esempio MySQL, il DBMS di Oracle, dalla sua versione 4.1 ha introdotto un nuovo tipo di memorizzazione delle password che non necessita di salt. Archivia infatti username e password nella tabella degli utenti, poi applica due volte l'algoritmo di hash alla password e memorizza la codifica esadecimale del risultato preceduto dal carattere "*" (per distinguere dalle versioni precedenti alla 4.1). Se si volesse ad esempio salvare nel database la password "SuperPippo!!!" si dovrebbe applicare un doppio hash `sha1(sha1(SuperPippo!!!))` e codificare il risultato in esadecimale ottenendo il campo user.password nel seguente modo `*400BF58DFE90766AF20296B3D89A670FC66BEAEC`. Per verificarne l'esattezza si può applicare il seguente comando:

```
$ echo -n 'SuperPippo!!!'| openssl sha1 -binary | openssl sha1 -hex
  (stdin)= 400bf58dfe90766af20296b3d89a670fc66beaec
```

Un altro ostacolo a cui bisogna fare fronte per conservare al meglio la password è la possibilità che questa venga letta durante la trasmissione in rete. Ciò avviene per mezzo di tre tecniche principali:

il packet sniffing intercettazione passiva da parte di un host all'interno della rete (soprattutto in reti broadcast);

l'attacco MITM acronimo di Man In The Middle, dove l'attaccante (fisico o logico) è in grado di leggere e modificare i messaggi scambiati tra due parti;

il traffic mirroring creazione di una copia dei dati in ingresso/uscita da uno switch o un router che viene inviata ad una applicazione per analizzare il traffico.

Una possibile soluzione a questo problema è la cifratura della password durante la trasmissione, tecnica ampiamente discussa in precedenza, in alternativa si possono usare password non ripetibili cioé valide per una sola transazione. Il vantaggio di usare queste ultime è che la lettura in rete risulta inutile, però essendo usa-e-getta la loro memorizzazione diventa impossibile. Ci sono tre tipologie di *OTP (One-Time-Password)* basate su tre approcci diversi. La prima è una OTP a catena, infatti l'utente possiede un elenco di password numerate e legate tra loro. Al momento di inserire i dati per l'autenticazione, l'utente deve fornire la password associata al numero indicato dal server (in figura il server richiede la password numero 48 dell'elenco). La seconda tipologia è la OTP time-based, cioé valida per un breve periodo di tempo. L'utente possiede un token che periodicamente genera una nuova password, quindi per autenticarsi deve fornire l'ultima password generata (in figura 16.12 la password delle 11:07). L'ultimo tipo di OTP è quello a sfida, dove torna in gioco la crittografia. Per

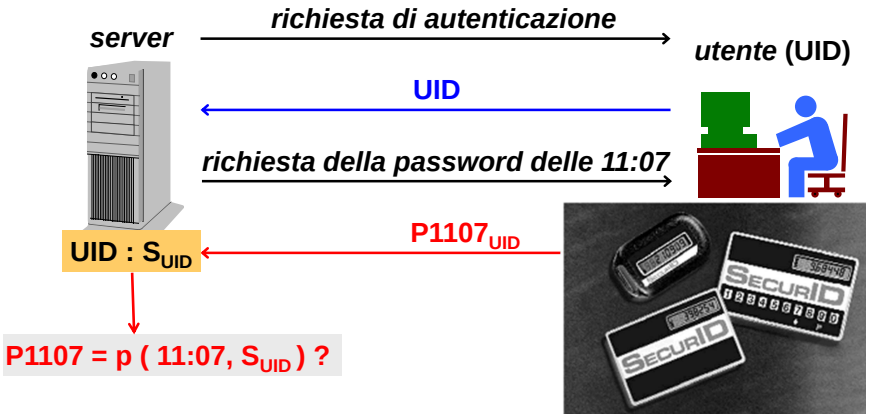

Figura 16.12: OTP time-based.

autenticarsi l'utente infatti fornisce al server un certificato dove sono presenti il suo UID e la sua chiave pubblica, il server quindi sfida l'utente inviandogli un messaggio cifrato con la chiave pubblica. L'utente allora con la propria chiave privata decifra la sfida e manda il messaggio in chiaro al server ottendendo così l'autenticazione. Una ulteriore tipologia ibrida di autenticazione tramite password non ripetibile è quella out-of-band. L'utente per autenticarsi invia al server il proprio UID insieme a un segreto (P), il server genera i dati di autenticazione e li invia all'utente tramite un canale diverso fuori dalla rete (ad esempio SMS tramite rete cellulare). Ricevuti i dati, l'utente li invia sul canale tradizionale al server e verrà autenticato.

16.5 Protezione delle reti e delle applicazioni

16.5.1 VPN

Una *VPN (Virtual Private Network)* è una tecnica (hardware e/o software) che permette di realizzare una rete privata utilizzando canali e apparati di trasmissione condivisi. La finalità di una VPN è quella di estendere una rete locale privata aziendale all'interno della rete pubblica di telecomunicazione in modo tale da far comunicare due parti interne all'azienda ma geograficamente lontane come se fossero sulla stessa LAN. La realizzazione di una rete VPN avviene nei modi seguenti:

- attraverso reti nascoste - si adottano indirizzi "privati", semplice ed economico ma poco protetto;
- tramite routing protetto (tunnel IP) - routing forzato dei pacchetti tra due edge router, più sicuro ma vulnerabile al gestore della rete;
- mediante protezione crittografica dei pacchetti rete (tunnel IP sicuro) - routing forzato combinato alla cifratura, molto sicuro ma spesso molto costoso.

L'*IPsec*, abbreviazione di *IP Security*, è uno standard di IETF (Internet Engineering Task Force) per la sicurezza a livello 3, cioé per fornire connessioni sicure su reti IP (IPv4 e IPv6). Lo scopo di IPsec è sia quello di creare VPN tra reti diverse tramite la modalità tunnel-mode sia di realizzare canali sicuri end-to-end per mezzo del transport-mode. I protocolli di IPsec che forniscono la cifratura del flusso dei dati sono due: l'autentication header (AH) che garantisce integrità e autenticazione e l'Encapsulating Security Payload (ESP) che in più offre la riservatezza dei dati. L'unico protocollo che implementa lo scambio di chiavi invece è l'Internet Key Exchange (IKE). La figura 16.13 mostra come funziona la modalità transport-mode nella creazione di un canale virtuale sicuro end-to-end. Partendo dal presupposto che creare un'architettura di sicurezza end-to-end tra tutti i nodi sarebbe troppo costoso, il canale virtuale sicuro coinvolge quindi solo i due host interessati che dovranno loro stessi essere in grado di implementare IPsec. Nell'immagine 16.14 invece viene illustrata la modalità tunnel-

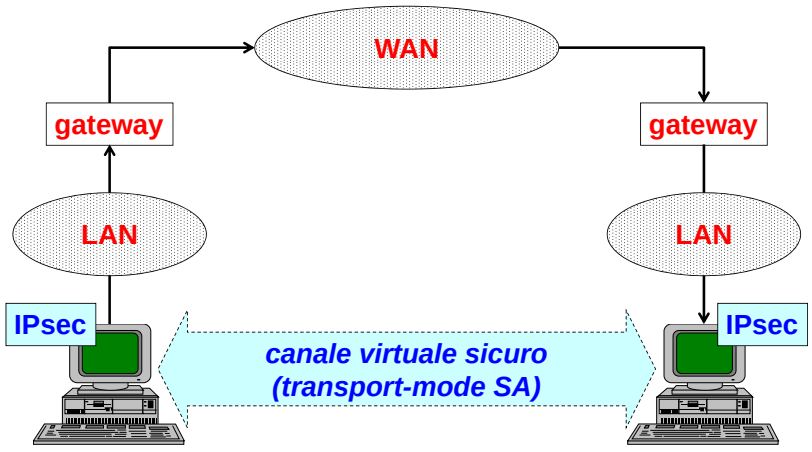

Figura 16.13: Configurazione IPsec "end-to-end security".

mode per realizzare una VPN che collega in modo sicuro due LAN. In questo caso IPsec è configurato nei gateway che, connessi in sicurezza tra loro, diventano il punto di contatto tra la LAN protetta sottostante e la rete di comunicazione (WAN) esterna. Nella figura 16.15

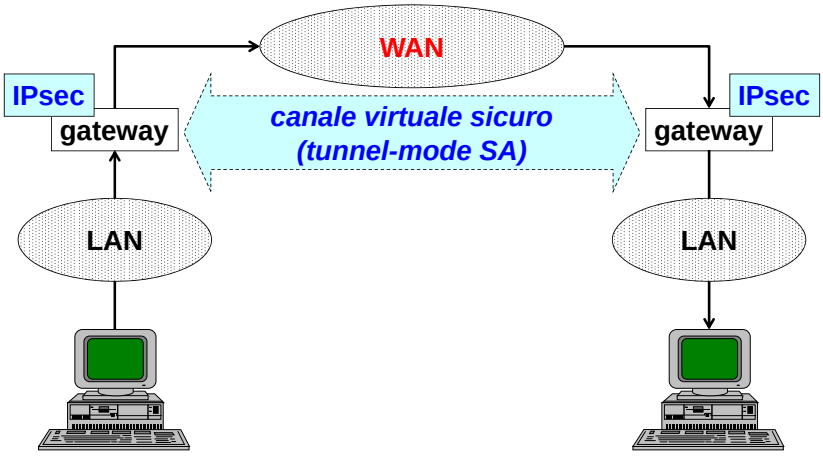

Figura 16.14: Configurazione IPsec "basic VPN".

si può osservare la creazione di una rete sicura che coinvolge un'intera LAN e un singolo host. Nel mondo reale questa situazione potrebbe verificarsi se un impiegato necessita di un collegamento alla rete della propria azienda per accedere ai servizi riservati, IPsec sarà quindi sia sul gateway al confine della LAN aziendale sia sul computer dell'impiegato.

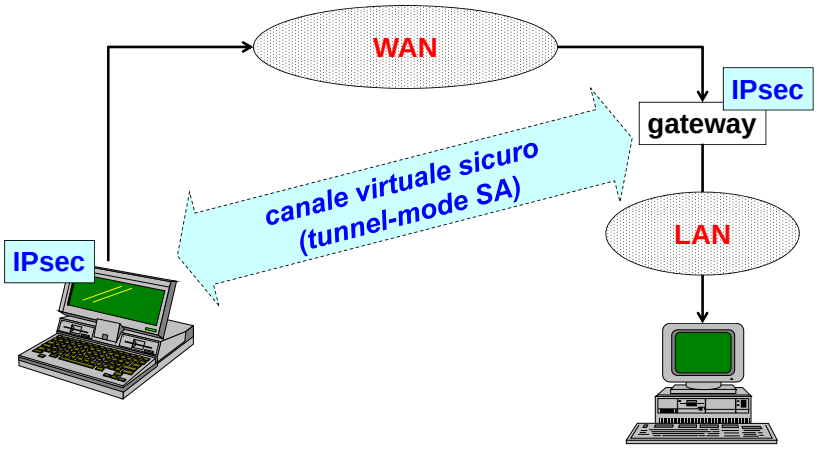

Figura 16.15: Configurazione IPsec "secure gateway".

16.5.2 Firewall

Un altro componente di difesa di una rete privata è il *firewall*. Il termine firewall letteralmente significa "muro tagliafuoco" e come suggerisce il nome impedisce che l'attacco pericoloso si propaghi da una zona con un basso livello di sicurezza ad un'altra che deve rimanere protetta. Nella pratica è un apparato di rete che filtra tutti i pacchetti in transito ritenuti sospetti in base alla politica di sicurezza che gli è imposta L'utilizzo del firewall garantisce maggior sicurezza quando il muro è integro, più si vanno a togliere mattoni per far circolare il traffico desiderato più la sicurezza diminuisce. In alcuni casi nasce l'esigenza di creare una sottorete promiscua dove porre elementi come i server pubblici che devono essere raggiunti agevolmente dall'esterno senza compromettere la sicurezza interna. Questo segmento separato di LAN viene chiamato *DMZ* ovvero zona de-militarizzata. Il firewall quindi agisce come punto di controllo del traffico tra la rete interna, quella esterna e una o più DMZ, come mostrato in figura 16.16. Come funziona questo controllo del traffico di rete? Esistono due livelli di filtraggio che regolano il passaggio dei pacchetti attraverso il firewall.

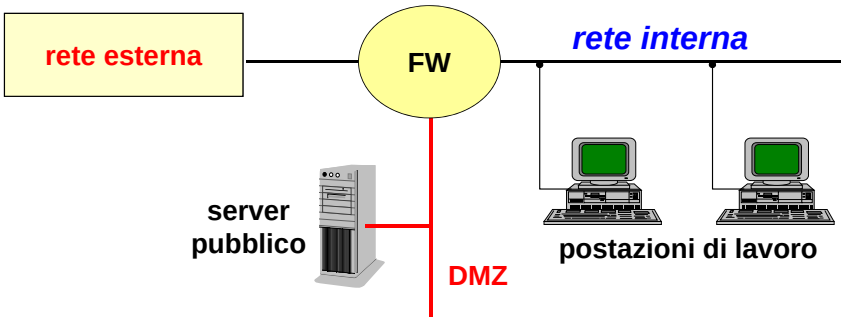

Figura 16.16: Schema di un firewall con DMZ.

Il primo viene detto *packet-filter* e controlla uno ad uno l'header dei pacchetti IP in transito utilizzando lo stesso modulo a prescindere dal servizio che lo richiede. Se da un lato questo meccanismo è molto veloce, dall'altro non garantisce una sicurezza ottimale poiché molti pacchetti pericolosi possono fingersi sicuri e vengono fatti passare. Il secondo livello è l'*application gateway* controlla il flusso applicativo applicando un modulo diverso per ciascuna applicazione (web, posta, etc.). La verifica sui protocolli richiede molto più tempo rispetto al packet-filter, ma la sicurezza che ne deriva è molto alta. Per alleggerire questo processo si è introdotto un *reverse proxy*: un server HTTP che fa solo da front-end e appare al client come un normale server web, ma in realtà passa le richieste al vero server. I benefici dell'utilizzo di un reverse proxy sono molteplici:

- obfuscation, non dichiara il vero tipo di server;
- load balancer, bilancia il carico tra più server di back-end smistando le richieste;
- acceleratore SSL, sveltisce le operazioni di SSL;
- web accelerator, salva nella propria cache i contenuti statici, per inviarli ai client più velocemente e senza coinvolgere i server di back-end;
- compressione, effettua la compressione (dei dati o del canale HTTP) alleggerendo da questo compito i server di back-end;
- spoon feeding, serve poco per volta ai client le pagine ricevute dal server, permettendo a quest'ultimo di dedicarsi ad altri lavori.

Per assicurare questi vantaggi però il reverse proxy deve essere ben posizionato all'interno della rete. La figura 16.17 mostra due possibili configurazioni di rete sfruttando le potenzialità degli apparati. Nello schema a sinistra il reverse proxy è posto nella DMZ collegato a due server web. Se i server devono comunicare con la rete interna invece vengono posti sulla LAN e viene creata una VPN per farli comunicare con il proxy sempre situato nella DMZ.

16.5.3 IDS/IPS

Dal momento che non tutti gli attacchi provengono dalla rete esterna, è necessario individuare anche i possibili attacchi che giungono dall'interno della rete locale. Questo è possibile grazie all'*Intrusion Detection System (IDS)*, un sistema per identificare accessi a computer o reti di utenti non autorizzati oppure autorizzati ma che violano i loro privilegi. Il presupposto è che ogni intruso si comporta in maniera anomala, di conseguenza per scovare le irregolarità sono state ideate due metodologie: la signature-based atta a individuare attacchi già verificatisi

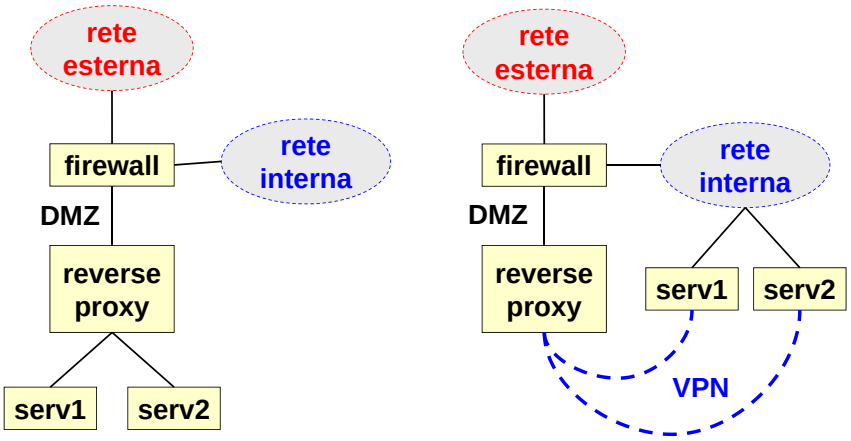

Figura 16.17: Configurazioni di un reverse proxy.

in precedenza memorizzando in un database la loro firma e l'anomalia statistica capace di smascherare nuovi tipi di attacco rilevando comportamenti atipici rispetto alla norma. Per venire in aiuto all'IDS è stato inventato l'*Intrusion Prevention System (IPS)*, una tecnologia che velocizza e automatizza la risposta alle intrusioni, infatti svolge la stessa funzione dell'IDS congiunto a un firewall dinamico distribuito. Il limite dell'IPS però è quello di avere reazioni automatiche agli attacchi, quindi c'è il pericolo che prenda la decisione sbagliata o blocchi traffico innocuo.

16.5.4 SSL/TLS

Salendo di grado nella scala di sicurezza si arriva al livello applicativo, completamente indipendente dalla rete, per cui si rendono necessarie nuove misure di sicureza indispensabili soprattutto per quei canai che attraversano un firewall. Il *Secure Socket Layer (SSL)* e il suo discendente standardizzato da IETF il *Transport Layer Security (TLS)* sono dei protocolli di trasporto sicuro proposti da Netscape con la finalità di proteggere da replay e da filtering le operazioni di autenticazione client-server, l'autenticazione e l'integrità dei messaggi e la riservatezza dei dati trasmessi. Il protocollo è impiegabile facilmente in applicazioni come browser, email (POP su TCP/995), trasferimento di file e soprattutto HTTP (HTTPS su TCP/443). Il funzionamento del protocollo SSL, come mostra la figura 16.18, è suddiviso in varie fasi:

1. richiesta di una pagina web sicura al server da parte del browser;
2. accordo tra le parti sulla configurazione di sicurezza;
3. invio del certificato di sicurezza al client da parte del server;
4. sfida asimmetrica in cui il server si autentica;
5. (opzionale) invio del certificato di sicurezza al server da parte dell'utente (se lo possiede);
6. (opzionale) sfida asimmetrica in cui il client si autentica;
7. creazione del canale sicuro SSL.

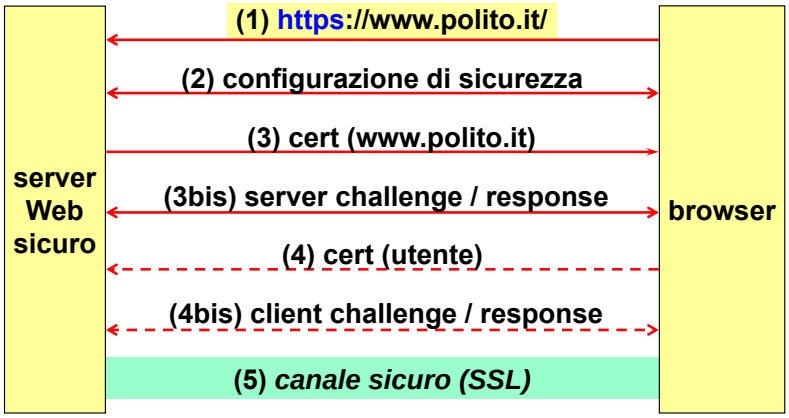

Figura 16.18: Attivazione di un canale SSL/TLS.

Le funzionalità di sicurezza del protocollo SSL sono molteplici. In primis la riservatezza, ottenuta tramite la cifratura simmetrica dei record, dove la chiave di cifratura è decisa dal client e inviata al server cifrata con la sua chiave pubblica. Una seconda funzionalità è la *peer authentication* all'apertura del canale, che consiste nella sfida asimmetrica obbligatoria del server, seguita in alcuni casi da quella del client (opzionale). Un'altra funzionalità è l'integrità, intesa sia come autenticazione dei dati, conseguita per mezzo di un keyed-digest dei record, sia come univocità dei dati, ottenuta grazie al MID (Message IDentifier) inserito in ogni record per evitare attacchi data filtering o replay.

16.5.5 Sicurezza in HTTP

Scendendo più nei dettagli di HTTP, si incontrano i meccanismi di sicurezza definiti dalla versione 1.0 di questo protocollo. Il primo metodo per consentire l'accesso ad un utente è il cosiddetto "address-based", dove il server HTTP permette o nega l'accesso in base all'indirizzo IP del client. Il secondo metodo è invece "password-based" altrimenti detto *Basic Authentication Scheme* in cui l'accesso è controllato dalla coppia username e password trasmessa codificata in Base64.

Entrambi i metodi risultano molto rischiosi poiché in HTTP si presuppone che il canale stesso sia sicuro, per poter usare in sicurezza il Basic Authentication Scheme bisognerebbe adoperare un canale SSL. La figura 16.19 mostra lo schema della tecnica Basic Authetication che avviene nelle seguenti fasi:

1. tramite una richiesta GET, il browser richiede l'accesso a una determinata pagina, protetta da HTTP Basic Authentication;

2. il server risponde con un errore temporaneo 401 (che è uno dei pochi casi in HTTP-1.0 in cui il canale non viene automaticamente chiuso) e lascia il canale aperto perché l'utente non aveva modo di sapere che doveva autenticarsi per accedere alla pagina;

3. il server invia quindi l'header "WWW-Authenticate" per richiedere username e password di un certo "reame" (inteso come dominio di sicurezza per cui l'utente si deve identificare, es. Google, Politecnico di Torino);

4. il browser richiede username e password all'utente e quindi li trasmette al server concatenandoli (usando il carattere ":" come separatore) e codificandoli in Base64 (si noti che se questa trasmissione viene intercettata la privacy viene violata!);

```
C: GET /path/alla/pagina/protetta
S:    HTTP/1.0 401 Unauthorized - authentication failed
S:    WWW-Authenticate: Basic realm="POLITO - portale della didattica"
C: Authorization: Basic Z3BhdXRhc3NvOlNlZ3JldGlzc2ltYQ==
S:    HTTP/1.0 200 OK
S:    Server: NCSA/1.3
S:    MIME-version: 1.0
S:    Content-type: text/html
S:
S:    <html> ... pagina protetta ... </html>
```

Figura 16.19: Accesso ad un risorsa protetta con Basic Authentication.

5. il server riceve le informazioni, le verifica con le password a lui note e – in caso positivo – consente l'accesso alla pagina protetta.

Se intercettiamo la trasmissione in figura 16.19 possiamo facilmente scoprire username e password effettuando la decodifica Base64:

```
B64decode("Z3BhdXRhc3NvOlNlZ3JldGlzc2ltYQ==") = gpautasso:Segretissima
```

Nella versione HTTP/1.1 è stata introdotta una miglioria per garantire maggior sicurezza: la *digest authentication* cioé autenticazione basata su sfida simmetrica. I miglioramenti di sicurezza incentrati sulla digest authentication sono consultabili all'RFC-2617 relativo ad *HTTP authentication: basic and digest access authentication*. La digest authentication è stata specificata originariamente nell'RFC-2069, ora divenuto obsoleto, ma viene ugualmente considerato come caso base in RFC-2617. La risposta R alla richiesta di autenticazione è formata calcolando il keyed-digest nel seguente modo (il simbolo + inidca concatenazione tra stringhe):

1. `HA1 = md5 ( A1 ) = md5 (` *user* `+ ":" +` *realm* `+ ":" +` *pwd* `)`
2. `HA2 = md5 ( A2 ) = md5 (` *method* `+ ":" +` *URI* `)`
3. `R = base64 ( md5 ( HA1 + ":" +` *nonce* `+ ":" + HA2 ) )`

Per evitare il replay viene usato un server nonce, inoltre il server di autenticazione può inserire un campo "opaque" per trasportare informazioni di stato (es. token SAML) relative al server. Nella figura 16.20 viene una richiesta HTTP protetta con digest authentication. Sapendo che la password dell'utente usato per l'autenticazione è `antonio`, è possibile verificare che la risposta inviata dal cliente è corretta coi seguenti calcoli:

1. `HA1 = md5 ("lioy:POLITO:antonio") = 89e81691bc5208d6237d1b850bca4f90`
2. `HA2 = md5 ("GET:/private/index.html") = d224a300ce6574688523c331faec896e`
3. `R = md5 ("89e...f90:dcd98b7102dd2f0e8b11d0f600bfb0c093:d22...96e")`
   `= 32a8177578c39b4a5080607453865edf`

```
C: GET /private/index.html HTTP/1.1
S:    HTTP/1.0 401 Unauthorized - authentication failed
S:    WWW-Authenticate: Digest realm="POLITO",
S:    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
S:    opaque="5ccc069c403ebaf9f0171e9517f40e41"
C: Authorization: Digest username="lioy", realm="POLITO",
C: nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
C: uri="/private/index.html",
C: response="32a8177578c39b4a5080607453865edf",
C: opaque="5ccc069c403ebaf9f0171e9517f40e41"
S:    HTTP/1.1 200 OK
S:    Server: NCSA/1.3
S:    Content-type: text/html
S:
S:    <html> ... pagina protetta ... </html>
```

Figura 16.20: Accesso ad un risorsa protetta con Digest Authentication.

16.5.6 Controllo accessi ai server web

I metodi di controllo degli accessi esposti finora possiedono tutti dei difetti:

VPN - la comunicazione è impossibile a livello IP;

SSL client authentication - se l'autenticazione non va a buon fine il canale TCP viene chiuso;

HTTP Basic/Digest Authentication - se l'autenticazione fallisce non si esegue il metodo HTTP richiesto;

autenticazione applicativa (form) - rischio elevato di errori di implementazione.

Nell'ottica di minimizzare l'*attack surface* (superficie di attacco), bisogna effettuare il controllo di accesso il prima possibile, poiché più tardi viene effettuata l'autenticazione più si espongono i livelli inferiori ad attacchi che sfruttano bachi o errori.

16.5.7 Sicurezza nelle applicazioni

Sicurezza in un form

Con il termine *form* si indica un'interfaccia che permette al client di compilare e inviare dei dati al server, nella maggioranza dei casi i dati sono sensibili e l'utente è restio a inserire tali dati in una pagina che non ritiene attendibile (ad es. non è HTTPS). Tecnicamente, non importa la sicurezza della pagina in cui si introducono i dati perché la sicurezza effettiva dipende dalla URI del metodo usato per inviare username e password al server. Ad esempio:

```
<form ... action=‘‘https://www.etc..’’>
```

. Psicologicamente però diventa importante la sicurezza della pagina in cui si introducono i dati perché pochi utenti hanno le conoscenze tecniche necessarie a verificare la URI del

metodo usato per l'invio. Il programmatore è quindi incentivato a porre molta cura nel maneggiare i dati degli utenti a partire dalla loro validazione, infatti dati non validati o non ripuliti sono sorgente di numerosi attacchi. Bisognerebbe pertanto optare per la politica "check that it looks good", ovvero controllare che sembri giusto, anzichè accertarsi che non sia sbagliato ("don?t match that it looks bad") poichè questa seconda opzione non riuscirebbe a vagliare ogni possibilità. In ogni tipo di applicazione non bigogna mai fidarsi del codice eseguito sul client, ma assumere sempre che i dati scambiati possano essere manipolati in modo improprio o inatteso, per questo motivo la sicurezza deve essere server-based (come mostrato in figura).

16.5.8 Attacco SQL injection

Un famoso tipo di attacco volto alle applicazioni web che utilizzano un DBMS di tipo SQL è il cosiddetto *SQL injection*. Questa tecnica sfrutta l'inefficienza dei controlli sui dati fornendo un input artefatto per alterare il codice SQL generato dinamicamente da un server, in questo modo si manipolano le query per rubare dati sensibili fuori dalla tabella che si sta usando (es. inserendo una UNION con la vista DBA_USERS). Un primo esempio di SQL injection in PHP è riportato di seguito. L'immagine 16.21 raffigura il codice PHP usato per controllare l'accesso ad un sito tramite username e password: una query SQL dove i dati (username e password) vengono passati tramite la request di PHP. Un utente normale

```
$sql = "SELECT * FROM WebUsers WHERE Username='"
  . $_REQUEST["username"]
  . "' AND Password='"
  . $_REQUEST["password"] + "'";

$rset = msqli_query ($con, $sql);

if (msqli_num_rows($rset) != 0)
  login OK ...
```

Figura 16.21: Estratto della pagina login.php passibile di attacco SQL injection.

quindi, come mostra la figura 16.22, compila il form e può effettuare il login solo se la coppia username/password è corretta. Un utente maligno invece, come mostrato in figura 16.23,

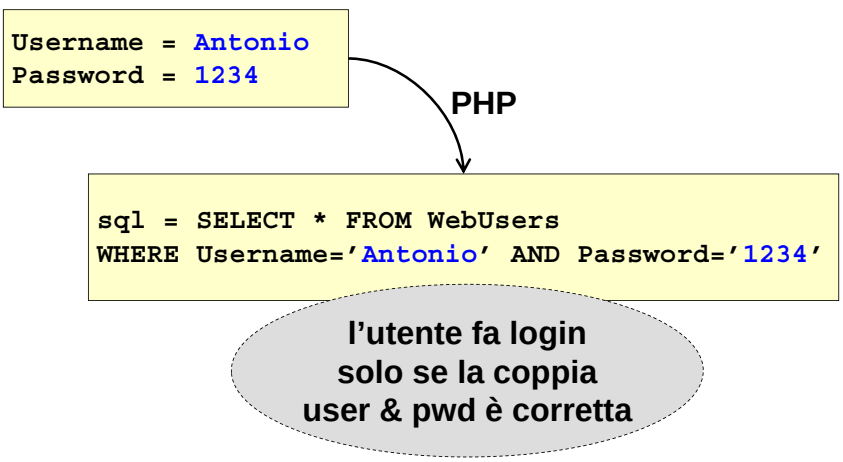

Figura 16.22: Uso della pagina login.php da parte di un utente normale.

compila normalmente il campo username, mentre nel campo password aggiunge in coda al dato una porzione di query. Quest'ultima, unita alla query prefabbricata, permette all'attaccante di effettuare il login senza conoscere username e password. In un altro esempio molto

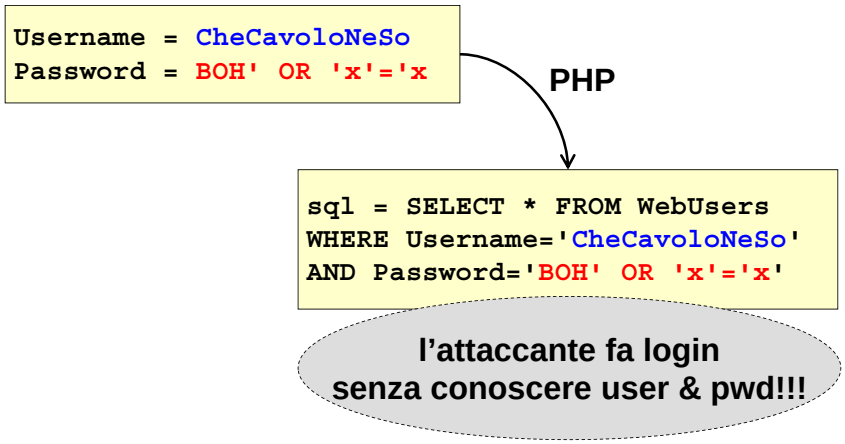

Figura 16.23: Uso della pagina login.php da parte di un utente malevolo.

più dannoso il dato che l'utente deve inserire è il nome del prodotto desiderato (figura 16.24). L'utente normale inserisce il nome del prodotto e ottiene i dati relativi all'articolo selezionato

```
$sql = "SELECT * FROM product WHERE ProductName='"
  . $_REQUEST["product_name"]
  . "'";

$rset = msqli_query ($con, $sql);

while ($row = mysqli_fetch_assoc($result))
{
   le righe vengono inviate al browser ...
}
```

Figura 16.24: Estratto della pagina prodotto.php passibile di attacco SQL injection.

(figura 16.25). Invece un utente maligno può inserire un'intera query SQL, concatenandola a quella preesistente, per ottenere tutte le coppie username/password presenti nel database (figura 16.26). Essendo l'SQL injection un attacco così dannoso, è opportuno prendere delle precauzioni per evitarlo. E' necessario quindi revisionare tutti gli script e le pagine dinamiche generate con ogni linguaggio (CGI,ASP,PHP,JSP, etc.). Inoltre bisogna sempre validare l'input dell'utente, ad esempio trasformando gli apici singoli in sequenze di due apici, e applicare tutte le contromisure presenti nelle librerie di sanitizzazione esistenti per diversi linguaggi. Un altro suggerimento che gli sviluppatori dovrebbero seguire è quello di usare query parametrizzate per introdurre i valori delle variabili fornite dall'utente, piuttosto che generare codice SQL tramite concatenazione di stringhe. Infine è incentivato l'utilizzo di sowtware di testing per verificare la propria vulnerabilità a questo tipo di attacco.

16.5.9 Attacco cross-site scripting

Un altro tipo di attacco che approfitta dell'insufficienza dei controlli nei form è il *cross-site scripting*, anche noto come XSS (o talvolta CSS). Questa mancanza di controlli da parte dei web server dinamici viene sfruttata per vari scopi, ma soprattutto per rubare le

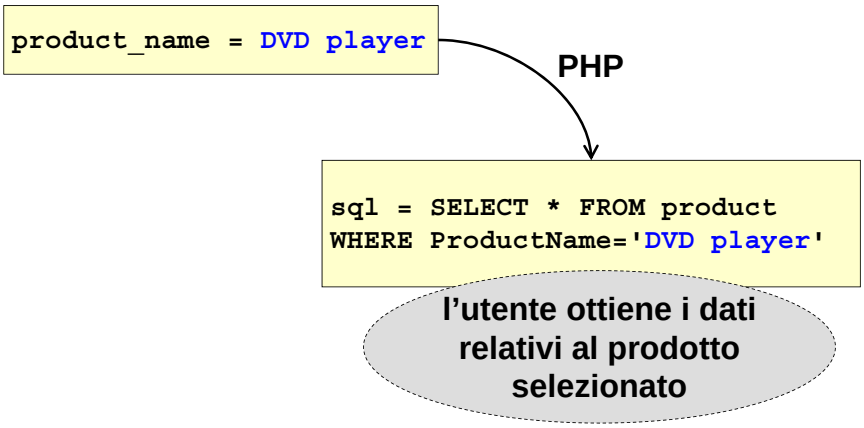

Figura 16.25: Uso della pagina prodotto.php da parte di un utente normale.

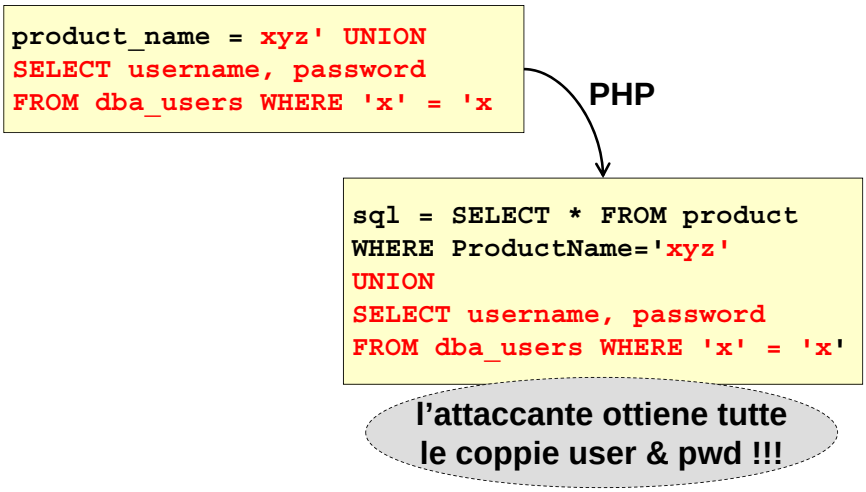

Figura 16.26: Uso della pagina prodotto.php da parte di un utente malevolo.

credenziali di autenticazione di un utente. Questo attacco, molto più comune di quanto si pensi, comprende una grande varietà di meccanismi e richiede una qualche forma di social engeneering; per questi motivi e per colpa della complessità delle attuali applicazioni web è ancora poco compreso dagli sviluppatori applicativi. Un esempio di XSS è riportato in figura 16.27. L'attaccante identifica un sito web che non filtra i tag HTML quando accetta input da un utente, per cui può inserire codice HTML e/o script arbitrari in un link o una pagina. Ad esempio l'attaccante potrebbe incorporare lo script nella casella della descrizione mentre registra un oggetto su Ebay, oppure potrebbe mandare lo script in una e-mail in formato HTML. Il meccanismo di XSS è studiato per raggirare la *Same-Origin Policy* (SOP), una regola di sicurezza che impedisce agli script l'accesso a pagine diverse da quella di provenienza. I browser permettono infatti agli script provenienti dal sito X di accedere solo ai cookie ed ai dati relativi a X. Per aggirare la SOP ed ottenere informazioni sensibili si fa eseguire alla vittima uno script per inviare i suoi cookie (relativi al sito dello script) al sito web dell'attaccante. Nel caso di Ebay quindi si inserisce lo script maligno nell'annuncio, la vittima aprendo l'annuncio sul proprio browser esegue lo script e inconsapevolmente invia i propri cookie all'attaccante. Il XSS pone in un'immagine la URL del dominio del cookie che si vuole rubare e il sito dell'attaccante. Se l'utente ha abilitato il riempimento automatico dei dati, i cookie vengono automaticamente inviati al sito maligno.

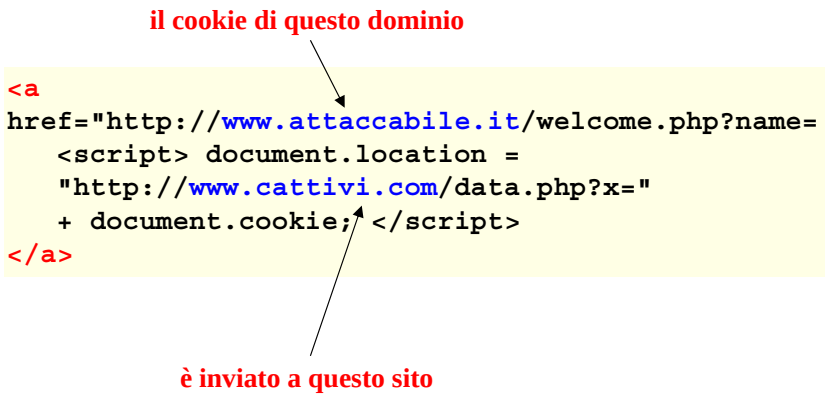

Figura 16.27: Cosa succede con questo script?

Protezione dei database

La sicurezza dei dati non riguarda solamente il loro utilizzo ma anche la conservazione, per questo motivo è molto importante proteggere i database adeguatamente. In primis il design della banca dati deve essere chiaro, bisogna separare le tabelle e i database, definire i ruoli dei vari utenti e limitarne i permessi. Inoltre la connessione al DBMS deve essere sicura, si possono utilizzare ad esempio TLS o IPsec, in modo da proteggere i dati in transito tra front-end e DBMS. Lo storage dei dati sensibili come password o dati personali deve essere custodito e cifrato, in questo modo si prevengono anche i furti di backup. Infine per sfuggire ad attacchi di SQL injection è bene validare sempre l'input dell'utente. Approfondendo la parte di design relativa ai ruoli, è opportuno definire sempre utenti logici (ruoli) oppure utenti fisici con associati i possibili ruoli. In aggiunta bisogna associare permessi diversi, ad esempio "read" o "write", alle diverse tabelle e alle operazioni sul database come "create table" o "drop table". Ad esempio tutti i direttori di un'azienda devono leggere i dati del personale, quindi avranno tutti il permesso di leggere, però solo il direttore di un'area può variare lo stipendio di uno degli impiegati, sarà quindi l'unico autorizzato a scrivere. Bisogna prestare molta attenzione a non collegarsi mai al DBMS come amministratore, ma utilizzare sempre un username specifico, in questo modo si può tenere traccia degli utenti che apportano modifiche al database. Per un'agevole gestione della banca dati è necessario seguire la politica del "least privilege", ovvero attribuire agli utenti i permessi minimi necessari a svolgere le loro attività lavorative. Ciò può essere fatto su base individuale (permessi assegnati esplicitamente ai singoli utenti) ma è più comodo e frequente assegnare i permessi in base al ruolo, come mostrato nell'esempio in figura 16.28.

16.5.10 OWASP

Per aumentare la sicurezza delle applicazioni è nato un progetto opensource chiamato *Open Web Application Security Project (OWASP)*, sul sito del progetto è possibile trovare una tabella degli attacchi maggiormente rischiosi. Nel 2013 la classifica dei 10 maggiori rischi sul web era così composta:

1. Injection;
2. Broken authentication and session management;
3. Cross-Site Scripting (XSS);
4. Insecure direct object references;

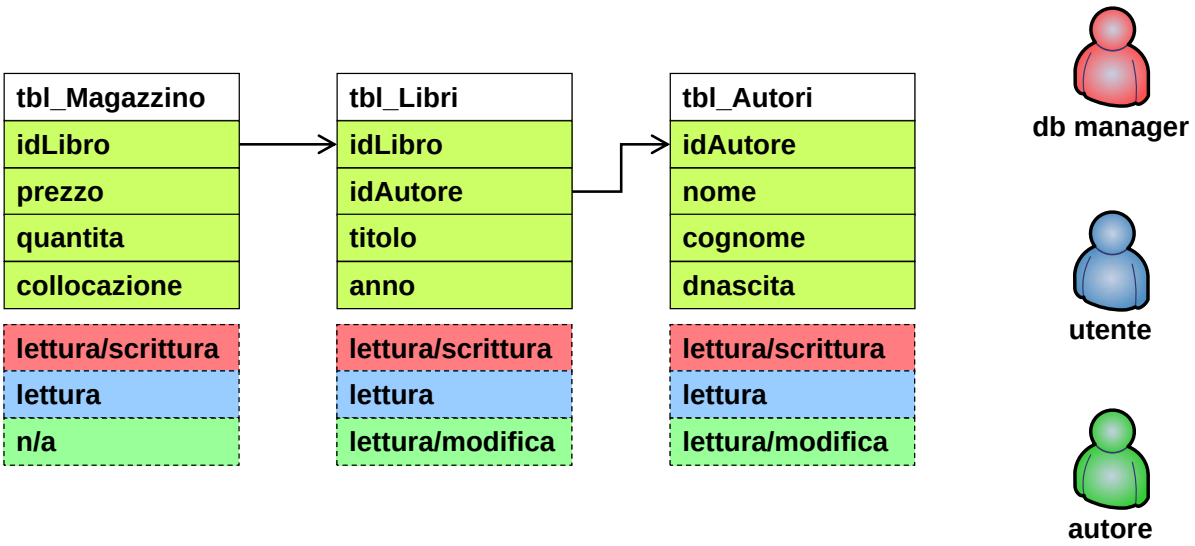

Figura 16.28: assegnazione di permessi differenti alle diverse classi di utenti.

5. Security misconfiguration;
6. Sensitive data exposure;
7. Missing function-level access control;
8. Cross-Site Request Forgery (CSRF);
9. Using components with known vulnerabilities;
10. Unvalidated redirects and forwards.

Injection

E' al primo posto, infatto l'SQL injection è il problema più famoso, ma sono anche possibili attacchi con LDAP, XPath, XSLT, HTML, XML e OS command. Il problema scaturisce dall'input inatteso eseguito da un interprete. Per questo motivo ci sono due soluzioni: la prima è evitare gli interpreti, la seconda invece è quella di adoperare comandi a struttura fissa, usando l'input dell'utente solo come un parametro.

Broken authentication and session management

La scorretta gestione dell'autenticazione e delle sessioni implica che schemi non standard di autenticazione contengano spesso errori poiché realizzarli in maniera corretta è molto difficile. Soprattutto ci sono difetti (come l'esposizione di password, account, identificativi della sessione) che sorgono in aree quali il logout, gestione password, timeout, "ricordami su questo computer" o l'aggiornamento dell'account. Queste debolezze permettono di ottenere i privilegi della vittima e spesso di controllare molte altre sessioni. Essendo tanti metodi diversi, sono attacchi molto difficili da individuare e di conseguenza da risolvere.

Cross-Site Scripting (XSS)

Il Cross-Site Scripting è stato già ampiamente trattato in precedenza.

Insecure direct object references

I riferimenti insicuri a oggetti diretti avvengono quando l'applicazione usa direttamente come parametro un oggetto reale (es. chiave di DB, file), infatti il client può cambiare tale riferimento per cercare di accedere a un diverso oggetto. Ad esempio nel 2000 il server web dell'ufficio Australiano delle tasse usava nella URL il codice fiscale dell'utente, in questo modo un malintenzionato ha raccolto i dati fiscali di altri 17.000 utenti. La soluzione a questo problema è quella di non esporre mai nell'interfaccia direttamente i riferimenti agli oggetti applicativi. Lo scenario di questo attacco è costituito da una query SQL composta con input fornito dall'utente:

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt =
connection.prepareStatement(query , ...);
pstmt.setString (1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Cambiando il parametro "acct" dal proprio browser l'attaccante può accedere a qualsiasi altro account (`http://example.com/app/accountInfo?acct= notMyAcct`).

Security misconfiguration

Un'errata configurazione di sicurezza permette di accedere al sistema o raccogliere informazioni riservate usando credenziali di default, pagine in disuso, vulnerabilita? non risolte, file e cartelle non protetti, ecc. Gli errori di configurazione possono presentarsi a qualsiasi livello dello stack applicativo (piattaforma, web server, application server, framework e codice personalizzato) per cui si rende necessaria la collaborazione con gli amministratori per assicurarsi che l'intero stack sia correttamente configurato anche usando scanner automatici. Gli scenari tipici della cattiva configurazione sono:

- framework vulnerabile con XSS quando l'aggiornamento è stato rilasciato, ma le librerie non sono state aggiornate;
- account amministratore creato automaticamente con username e password standard, ma non viene disabilitato;
- Directory Listing non disabilitato per cui l'attaccante scarica il codice sorgente ed identifica le vulnerabilità più facilmente;
- server web visualizza lo stack trace e l'attaccante ricava informazioni dai messaggi d'errore.

Sensitive data exposure

I dati sensibili (es. id personale, carte di pagamento, credenziali di autenticazione) vanno protetti in modo particolare sia quando memorizzati sia in trasmissione (rispettivamente noti come *data at rest* e *data in transit*).

I tipici problemi dei dati memorizzati sono molteplici: dati sensibili non cifrati o cifrati con algoritmi "casalinghi", uso errato di algoritmi forti o notoriamente deboli (es. RC2-40), chiavi memorizzate direttamente nell'applicazione o in file non protetti, memorizzare dati non necessari (ossia violando il principio *need-to-know*).

Per quanto riguarda i dati in transito bisogna esclusivamente usare canali protetti per le sessioni autenticate verso i client e per i collegamenti verso il back-end. Spesso non viene usato un canale protetto a causa di un presunto peggioramento delle prestazioni o per conflitti con gli IDS/IPS e quando viene usato ciò avviene solo durante la fase di autenticazione oppure con certificati scaduti o non configurati correttamente.

I dati memorizzati sono in pericolo anche quando vengono cifrati all'interno di un database, come ad esempio le carte di credito, se esistono query che li possono decifrare perché tramite SQL injection si ottengono i dati in chiaro, quindi bisogna rendere il database robusto verso questo tipo di query. Un'altra situazione rischiosa è quella di avere il backup su CD di alcuni dati, come quelli sanitari, ma la chiave è inserita nello stesso backup. Infine un'ulteriore circostanza dannosa è quella di archiviare le password senza salt in un database, in questo frangente un attacco brute force terminerebbe in 4 settimane anzichè 3000 anni.

I dati in transito invece sono a rischio quando un sito non protegge bene la pagina che richiede autenticazione (SSL/TLS), infatti l'attaccante può intercettare il traffico di rete, catturare cookie di sessione e impersonare la vittima. Allo stesso modo se il sito utilizza un certificato scaduto, abituato all'avvertimento l'utente non si accorge quando viene reindirizzato su un sito diverso e gli comunica le sue credenziali (phishing). Infine il caso peggiore avviene quando la connessione al database avviene tramite oggetti ODBC/JDBC perché tutte le comunicazioni avvengono in chiaro.

Missing function-level access control

Letteralmente significa mancanza di controllo di accesso a livello di funzione, avviene quando per proteggere una URL (o un'azione richiesta, es. `?action=payment`) la si nasconde o la si toglie dall'interfaccia (security through obscurity) senza eseguire alcun controllo di autenticazione, ruolo, client-side o semplicemente sui dati forniti dal client. Il problema sorge perché la protezione è facilmente superabile, ad esempio se la URL era presente in una prima versione e poi rimossa. Supponendo di avere le seguenti pagine:

`http://example.com/app/getappInfo`

`http://example.com/app/admin_getappInfo`

getappInfo è disponibile a tutti, il link che porta a admin_getappInfo richiama una procedura di autenticazione che però non viene attivata se si scrive direttamente la URL nel browser, per cui basta solo indovinare il nome del file.

Cross-Site Request Forgery (CSRF)

Questo attacco avviene tramite l'invio al browser (es. tramite sito web "cattivo" o mail HTML) di una URL attiva per eseguire un'azione sul server bersaglio. Ad esempio:

```
<img src="http://miabanca.it/do?azione=bonifico">
<img src="http://example.com/app/transferFunds?
   amount=1500&destinationAccount=attackerAcct#" width="0" height="0" />
```

E' molto efficace verso quei server web che usano autenticazione automatica (es. basata su cookie dopo login, ticket di Kerberos, user+pwd inviati automaticamente).

Using components with known vulnerabilities

Letteralmente significa utilizzo di componenti con vulnerabilità note e si verifica quando le applicazioni web sono complesse e usano molte componenti (es. librerie lato server o client, framework, servizi). Le vulnerabilità di queste componenti sono spesso note e risolte, ma raramente gli sviluppatori aggiornano le applicazioni (re-build) anche perché spesso non sanno quali librerie stanno usando (dipendenze profonde e nascoste). Ciò permette l'uso di strumenti di attacco automatici ed estende la platea degli attaccanti anche a persone senza le necessarie conoscenze, per cui occorre implementare un adeguato Component Lifecycle Management (CLM).

Unvalidated redirects and forwards

L'attacco consiste nel forzare la vittima ad usare un link con un redirect non validato, poiché se link appartiene a un sito valido la vittima si fida. Specificare pagina bersaglio in un parametro non validato permette di scegliere arbitrariamente la pagina di destinazione. E' un attacco facile da rilevare perché si possono controllare i redirect, non è altrettanto semplice se si usano i forward (uso interno allo stesso sito). Gli scenari di reindisizzamenti e inolti non validati sono normalmente siti con funzione "redirect.jsp" che ricevono un singolo parametro "url" dove l'attaccante crea un URL malevolo per installare malware o per fare phishing (`http://www.x.com/redirect.jsp?url=evil.com`). Altri casi sono i forward per girare richieste tra diverse parti del sito tramite un parametro (es. se una transazione è avvenuta correttamente) dove l'attaccante crea un URL per accedere a una pagina alla quale non è possibile accedere direttamente (`http://www.x.com/boring.jsp?fwd=admin.jsp`).

16.6 Sistemi di pagamento elettronico

A causa del fallimento della moneta elettronica per problemi tecnici e normativi (es. bancarotta di DigiCash) e del fallimento di SET (Secure Electronic Transaction) per problemi tecnici ed economici, attualmente il metodo di pagamento elettronico più usato è la trasmissione del numero di carta di credito su canale SSL. Questo metodo però non garantisce contro le frodi: la maggior parte dei tentativi di frode nascono da transazioni Internet, che però sono solo una piccola parte del volume d'affari. La figura 16.29 mostra l'architettura di pagamento via web, che si svolge nelle seguenti 8 fasi:

1. il merchant, ovvero il negoziante online, pubblica un'offerta della merce che vuole vendere;
2. l'offerta attira l'attenzione di un cardholder, un possessore di carta di credito, che decide di effettuare un ordine;
3. il cardholder viene reindirizzato al payment gateway dal sito del merchant;
4. tramite SSL il payment gateway chiede i dati della carta di credito al cardholder;
5. il cardholder risponde fornendo i dati, sempre tramite SSL;
6. il POS virtuale si interfaccia tra il mondo di Internet e il mondo finanziario, chiede alla payment network se i dati ricevuti dal cardholder sono corretti;
7. dalla payment network arriva il responso sui dati;

8. se i dati sono corretti si procede con il pagamento, il merchant viene informato del pagamento avvenuto e può quindi spedire la merce.

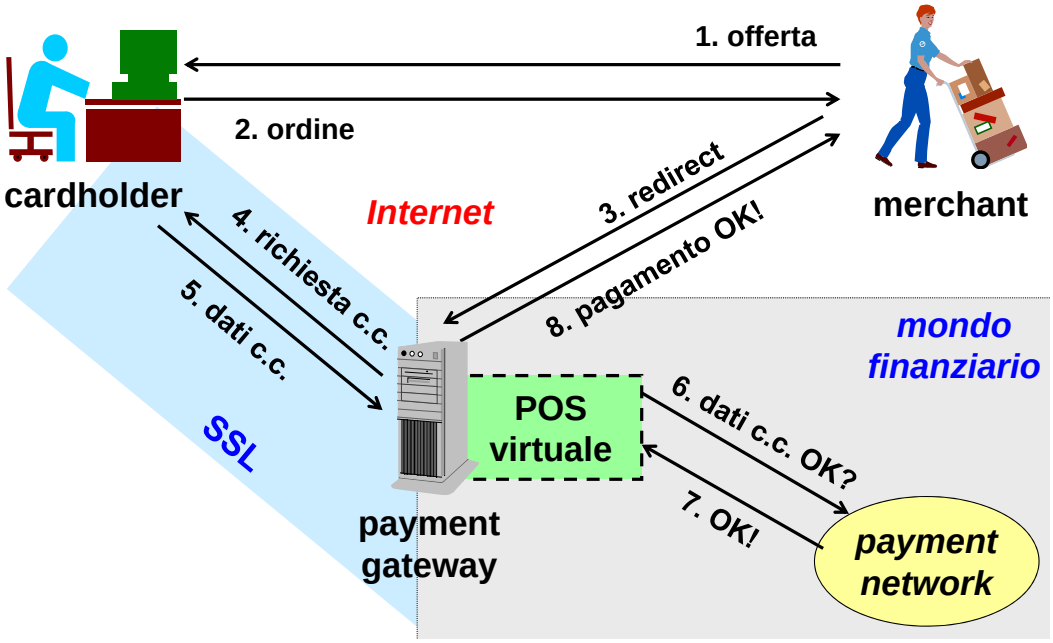

Figura 16.29: Architettura di pagamento via web

L'ipotesi base affiché un pagamento elettronico vada a buon fine è che l'acquirente possieda una carta di credito e che abbia un browser con SSL. La sicurezza effettiva però dipende dalla configurazione sia del server sia del client, inoltre il payment gateway ha tutte le informazioni (pagamento + merce) mentre il merchant ha solo le informazioni sulla merce. L'ente *Payment Card Industry Data Security Standard (PCI DSS)* ha definito una serie di regole con l'obiettivo di proteggere i dati dei possessori di carte di credito. Ad oggi questo standard viene richiesto da tutte le carte di credito per le transazioni su Internet, infatti è molto più prescrittivo rispetto ad altre norme di sicurezza come la HIPAA (Health Insurance Portability and Accountability Act). Le regole principali verranno esposte di seguito, maggiori dettagli sugli standard si possono trovare al sito: `https://www.pcisecuritystandards.org`. Se si vuole costruire e mantenere una rete protetta bisogna installare e mantenere una configurazione con firewall e non usare password di sistema predefinite o altri parametri di sicurezza impostati dai fornitori. Per proteggere invece i dati dei titolari delle carte è sempre opportuno memorizzarli con attenzione e cifrarli quando vengono trasmessi attraverso reti pubbliche aperte. Inoltre bisogna sempre rispettare un programma per la gestione delle vulnerabilità, di conseguenza bisogna usare e aggiornare con regolarità l'antivirus, ma anche sviluppare e mantenere protetti applicazioni e sistemi. In particolar modo è necessario implementare misure forti per il controllo degli accessi, quindi si deve consentire l'accesso ai dati solo se questo è effettivamente indispensabile per lo svolgimento dell'attività commerciale (politica del least privilege) e di conseguenza limitare la possibilità di accesso fisico ai dati, magari chiudendo i server a chiave e introducendo l'uso di un ID univoco per ogni utente del SI. Infine bisogna monitorare e testare le reti con regolarità, tenendo traccia di tutti gli accessi effettuati alle risorse della rete e ai dati dei titolari delle carte, ma anche eseguendo test periodici dei processi e dei sistemi di protezione. Quindi per mantenere un sistema sicuro bisogna stilare l'elenco delle regole che si vuole adottare e soprattutto osservare la politica di sicurezza che si viene così a creare, senza quest'ultima importante regola tutte le altre vengono meno.

Indice analitico