Ministero dell'Istruzione, dell'Università e della Ricerca Servizio Automazione Informatica e Innovazione Tecnologica

Modulo 13

Realizzazione di pagine Web

ForTIC

Piano Nazionale di Formazione degli Insegnanti sulle Tecnologie dell'Informazione e della Comunicazione

Percorso Formativo C

Materiali didattici a supporto delle attività formative

2002-2004

Promosso da:
- Ministero dell'Istruzione, dell'Università e della Ricerca, Servizio Automazione Informatica e Innovazione Tecnologica
- Ministero dell'Istruzione, dell'Università e della Ricerca, Ufficio Scolastico Regionale della Basilicata

Materiale a cura di:
- Università degli Studi di Bologna, Dipartimento di Scienze dell'Informazione
- Università degli Studi di Bologna, Dipartimento di Elettronica Informatica e Sistemistica

Editing:
- CRIAD - Centro di Ricerche e studi per l'Informatica Applicata alla Didattica

Progetto grafico:
- Campagna Pubblicitaria - Comunicazione creativa

In questa sezione verrà data una breve descrizione del modulo.

Gli scopi del modulo consistono nel mettere in grado di:

- Conoscere i principi di progettazione di una pagina *Web* e gli strumenti per produrla.
- Sviluppare pagine *Web* con l'uso di opportuni strumenti *software* di *authoring* e programmazione che permettano l'inserimento di link, *frame*, tabelle, opzioni di accessibilità per disabili.

Il modulo è strutturato nei seguenti argomenti:

- **Progetto delle pagine**
  - Descrivere i fattori di interazione uomo-macchina che influenzano il progetto di pagine *Web* e di un sito.
  - Descrivere e usare il processo di organizzazione (*storyboarding*) di un sito *Web*.
  - Descrivere i principi di progettazione, strutturazione e costruzione di un sito *Web*.
  - Valutare un sito *Web* usando principi di buona progettazione, strutturazione e formattazione.
- **Strumenti di produzione**
  - Elencare strumenti di produzione in ordine di complessità di uso.
  - Valutare *software* per la realizzazione di pagine *Web*.
  - Installare e configurare strumenti per la produzione di pagine *Web*.
- **Sviluppo di pagine e siti**
  - Creare pagine *Web* usando strumenti di *authoring*.
  - Usare linguaggi di programmazione *Web* per creare e aggiornare pagine *Web*.
  - Inserire un *e-mail* link in una pagina *Web*.
  - Inserire link interni ed esterni in una pagina *Web*.
  - Inserire *frame* in una pagina *Web*.
  - Inserire tabelle in una pagina *Web*.
  - Inserire opzioni di accessibilità per disabili in una pagina *Web*.
  - Inserire possibilità di trasferimento *file* in una pagina *Web*.
  - Progettare e creare un sito *Web*.
  - Installare e configurare un motore di ricerca per un sito *Web*.

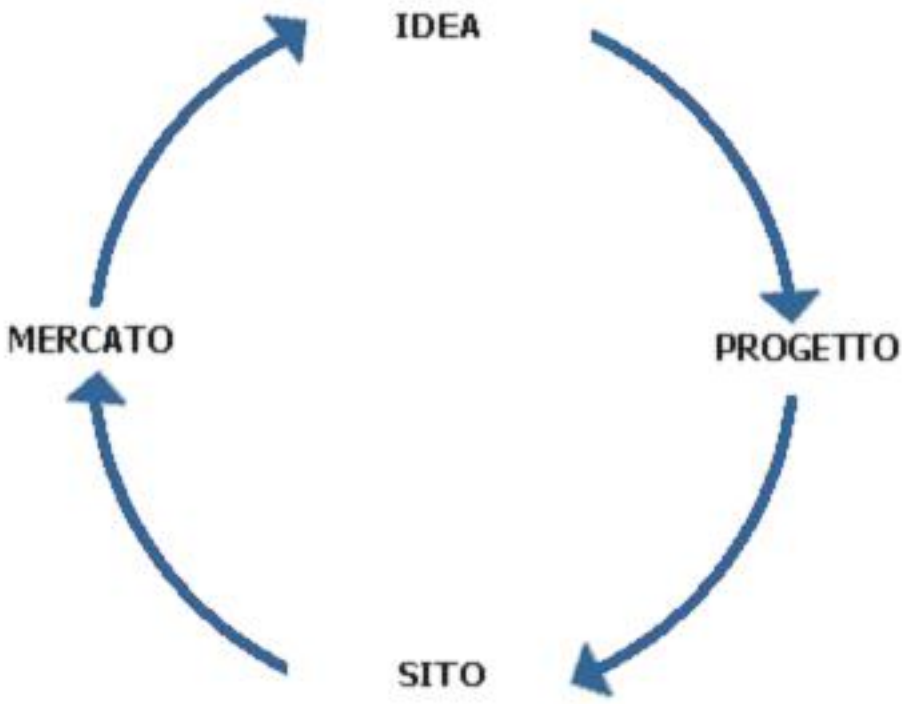
Introduzione

Progetto delle pagine

Cosimo Laneve

Introduzione

Il progetto di una pagina *Web* va inquadrato all'interno di quello più ampio, ovvero di **sito *Web***. Va detto che un sito *Web* non è una necessità: ci sono importanti attività nell'Internet che si possono svolgere senza avere un sito. Soprattutto, una presenza *Web* è un punto di arrivo e non un punto di partenza. Questo è ovvio se si imposta il problema secondo le normali logiche di impresa, e cioè partendo da una esigenza del mercato, secondo il flusso descritto di seguito.



flusso che illustra la logica imprenditoriale

Ma stranamente nel caso delle nuove tecnologie accade spesso che si segua il percorso inverso, con risultati quasi sempre deludenti. Tuttavia è abbastanza probabile che un coerente ed efficace progetto di attività *on-line* porti, presto o tardi, ad avere anche un sito *Web* o magari più di uno. Questo capitolo riassume alcuni dei criteri che è opportuno seguire nella progettazione, gestione e manutenzione di un sito.

Non è necessario che:

- **un sito abbia una frequentazione molto numerosa**. Nella maggior parte dei casi di grande afflusso indifferenziato non è un vantaggio, è un

problema. La qualità conta molto più della quantità. Dieci visitatori **specificamente interessati** valgono molto di più di mille vagabondi di passaggio che sottraggono risorse al sistema senza utile. E' meglio avere pochi visitatori, con cui verificare la funzionalità del sito e dei servizi offerti, per poter correggere gli inevitabili errori e affinare le relazioni, prima di sviluppare l'attività su scala più ampia

- **Ci sia un eccessivo investimento iniziale**. Ci sono imprese italiane che hanno speso (e sprecato) miliardi per mettere un sito online, con conseguenze catastrofiche. Per cominciare bene occorre un investimento adeguato (soprattutto in **definizione delle strategie**, organizzazione del progetto e sviluppo delle risorse umane) ma è importante che si predispongano fin dall'inizio investimenti continuativi e graduali più che tentare di fare tutto e subito.

Fattori di interazione uomo-macchina che influenzano il progetto di pagine Web e di un sito

Definire gli obiettivi del sito

Quanto più ristretto e specifico è l'obiettivo, tanto più efficace è la funzione di un sito e più gestibile è la sua attività. Solo in casi particolari (e in base a specifiche esigenze della strategia d'impresa) può avere senso assumere il ruolo di **portale**, cioè cercare di essere un punto di riferimento più esteso rispetto alla missione specifica dell'impresa. In questa ipotesi l'**impegno** - soprattutto per la generazione e la gestione dei contenuti - è molto maggiore che nel caso di un sito aziendale con obiettivi più mirati e gestibili.

Tra gli obiettivi, non bisogna assolutamente trascurare i problemi specifici di accesso agli elaboratori da parte delle varie categorie di persone disabili. Questi aspetti sono trattati diffusamente nel modulo 14.

Privilegiare i contenuti, poi l'estetica

A differenza di una vetrina, un manifesto, un annuncio su un giornale o qualsiasi forma di ***display***, un sito *Web* non ha il problema di attirare l'attenzione in competizione con altri, anzi la grafica dovrebbe essere finalizzata anch'essa all'idea, ma soprattutto alla **usabilità** delle informazioni predisposte. Quando una persona arriva su un sito lo fa intenzionalmente e vede solo quello. È bene che un sito abbia anche un aspetto gradevole e attraente - ma la grafica e l'estetica non hanno il compito di attrarre l'attenzione. Devono essere funzionali e favorire un rapido e **agevole** accesso ai contenuti e alle informazioni.

Negli altri mezzi di comunicazione si ha un controllo diretto su come si vedrà ciò che proponiamo. Questo non è possibile con la tecnologia *Web*, come vedremo più avanti. C'è un contrasto fra due modi di affrontare il problema: **(1) accettare e prevedere la variabilità della ricezione**, o **(2) cercare di forzare una ricezione meno imprecisa**. Possono convivere, secondo le esigenze specifiche, le due soluzioni - ma in generale è meglio che prevalga la **(1)**, e comunque non è mai possibile un controllo totale e un'assoluta omogeneità.

La struttura **ipertestuale** (a differenza di qualsiasi altro veicolo di comunicazione) permette una quantità potenzialmente infinita di informazione. La conseguenza è che

quando una persona si collega a un sito *Web* (o a qualsiasi sistema di informazione disponibile in rete) si aspetta di trovare **tutto** ciò che cerca e se non lo trova rimane delusa. Questo non accade con un annuncio, con un film e neppure con un libro: contenitori potenzialmente finiti da cui non ci si può aspettare tutto. Ma la completezza e ricchezza di informazioni (con un ben articolato sistema di accesso e orientamento) diventa un'esigenza inderogabile nel caso di un sito *Web*.

Un sistema interattivo

In *Internet* ci si aspetta una possibilità di dialogo interattivo e quindi è necessario prevederla nella progettazione delle pagine, ed offrirla. L'offerta deve essere efficace e gradita a chi la chiede, ma anche praticamente gestibile senza eccessive difficoltà.

Organizzazione di un sito Web

La fase più importante della progettazione di un sito *Web* è l'**organizzazione delle informazioni**. Al fine di considerare accuratamente ciò che si desidera dire e come si desidera dirlo è fondamentale conoscere profondamente il contenuto del proprio sito. Allora, prima di passare alla realizzazione del sito e delle pagine, bisogna creare prospetti e suddividere le informazioni in sezioni e sottosezioni, e pensare alle interrelazioni tra le sezioni create. Bisogna inoltre, avere le idee chiare su come una sezione del sito si collega ad un'altra, ed avere un chiaro senso dell'organizzazione. E' possibile utilizzare quattro strutture essenziali per creare siti *Web*:

Sequenze - Questo è il modo più semplice per organizzare le informazioni e consiste nell'inserirle in sequenza. L'ordinamento sequenziale può essere cronologico oppure alfabetico come nelle enciclopedie. Questo tipo di struttura è adatta per i siti nei quali il lettore deve percorrere una sequenza fissa di informazioni ed i collegamenti costituiscono un percorso lineare. Anche i siti *Web* più complessi possono essere organizzati come sequenza logica, ma ogni pagina della sequenza principale può avere collegamenti ad altre pagine non facenti parte della sequenza o ad altri siti *Web*.

Griglie - Le griglie rappresentano un buon metodo per correlare variabili, come informazioni cronologiche i storiche, in varie categorie *standard*, per esempio tecnologia, cultura, storia. Le singole unità di una griglia devono condividere una struttura uniforme di argomenti e sottoargomenti ed il pubblico deve essere in grado si comprendere la natura della struttura generale. Quindi le griglie possono essere difficili da seguire se gli utenti non riconoscono le relazioni tra le categorie e le informazioni; per questo motivo sono più adatte ad un'utenza esperta con una certa comprensione dell'argomento trattato e della sua organizzazione logica.

Gerarchie - Per organizzare unità complesse di informazioni le gerarchie sono il modo migliore. Generalmente i siti *Web* sono organizzati su un'unica *home page*, e gli schemi gerarchici sono particolarmente adatti a tale organizzazione. Inoltre i diagrammi gerarchici sono molto utilizzati nella vita aziendale ed istituzionale, quindi per la gran parte degli utenti questa struttura è di facile comprensione. Attenzione però, perché le gerarchie sono pratiche solo con materiali ben organizzati, e questo richiede un'approfondita organizzazione analitica delle informazioni da proporre.

Reticolati - Nelle strutture a reticolato l'obiettivo è imitare il pensiero associativo ed il flusso libero delle idee, consentendo agli utenti di seguire i loro interessi senza alcuna sequenza preimpostata. Con questa struttura si sfrutta al massimo la potenza di

collegamento e associazione *Web*, ma talvolta la struttura a reticolato può indurre in confusione. Questo perché gli schemi organizzativi associativi sono molto difficili da comprendere e prevedere per l'utente. Allora le strutture a reticolato sono adatte a siti di piccole dimensioni basati su elenchi di collegamenti e non sono adatte alla comprensione di un argomento.

Il progettista *Web*, che conosce a fondo le informazioni che vuole proporre, potrà scegliere di usare uno dei quattro modelli informativi descritti, ma potrà anche creare un modello che condivida alcuni aspetti di tutti e quattro i tipi di struttura.

Principi di progettazione, strutturazione e costruzione di un sito Web

Lo sviluppo di un sito *Web* non è sostanzialmente diverso da qualsiasi altro progetto di comunicazione. Parte da una strategia e si conclude con una realizzazione, poi sottoposta a verifica. Ma il processo si svolge in un modo che è specifico a questo sistema e quindi può essere opportuno definirlo nelle sue fasi, che sono principalmente **sei**.

- **Strategia e obiettivi.**Occorre individuare gli obiettivi specifici del sito; come si collocano nel sistema complessivo di comunicazione dell'impresa; in che relazione si collocano con gli altri sistemi di relazione, interni ed esterni. Occorre anche comprendere come ci si assicurerà che tutto il sistema-impresa sia cosciente di ciò che il sito fa e promette; come l'attività *on-line* sarà coordinata e integrata con il resto.
- **Contenuti e aspettative.** Chi dovrà realizzare il sito deve avere indicazioni chiare e precise sulle informazioni che devono essere raccolte e gestite, sulla strategia dell'impresa, sugli obiettivi del sito, sui risultati attesi, sull'investimento assegnato (non solo nella fase iniziale ma anche per le evoluzioni successive), su quali saranno le esigenze di gestione e verifica nel tempo.
- **Servizio e gerarchia dei valori**. Occorre definire con chiarezza a quale pubblico ci si rivolge, quali specifici servizi si vogliono offrire, quali sono le gerarchie di importanza dei servizi e dei contenuti e perciò in che modo dovranno essere resi accessibili e fruibili, compatibilmente con le risorse a disposizione. Di conseguenza si cominciano a definire le tecnologie necessarie.
- **Progettazione.** Si esplorano gli aspetti grafici, colori e stili, si identifica l'interfaccia di base, si decide se ci sarà bisogno di particolari come audio, filmati o altro. Si identificano le possibilità di interazioni. Questa è la parte più appariscente del lavoro, può essere la più divertente, ma non è necessariamente la più importante. Soprattutto non deve distrarre o deviare dagli obiettivi.
- **Sviluppo.** Si scrivono i testi che ancora mancano, si elaborano immagini e altri elementi tecnici, si elaborano le pagine statiche e si preparano i prototipi per le pagine dinamiche, che vengono create al volo sulla base delle richieste dell'utente. Si verificano le varie soluzioni su diversi tipi di *browser* e di piattaforma. Si verifica, si corregge, si verifica, si corregge, e infine si pubblica.
- **Verifica ed evoluzione.** Finalmente il sito è *on-line*, e le verifiche sono appena cominciate. Un sito ben funzionante è sempre in continua evoluzione e sperimentazione, in quotidiana manutenzione, in costante aggiornamento e in perenne verifica e miglioramento. Non c'è nulla di peggio che connettersi ad un sito che contenga le notizie che ci servono che sono vecchie di anni.

I punti chiave da tener conto, fase per fase, sono i seguenti

- **Strategie e obiettivi.** Come si inquadra l'obiettivo del sito nel sistema di comunicazione e di relazioni dell'impresa. Come si collega con i sistemi informativi interni ed esterni. Quali risultati si intendono ottenere, con quali previsioni di tempo e con quali criteri di verifica. Quali sono le strutture interne dell'impresa che devono essere coinvolte e come è organizzato il metodo per il loro coinvolgimento. Quali sono i vincoli determinati dalla cultura e missione dell'impresa, dai suoi programmi operativi, dalle relazioni esistenti con fornitori, intermediari, clienti e altri interlocutori del sistema-impresa. Quali sono le aree di possibile rischio e quali le cautele per prevenirle. Quali sono i referenti all'interno dell'impresa per la soluzione di problemi imprevisti e quali i metodi per poter intervenire velocemente.
- **Contenuti e aspettative.** Identificare i concorrenti e verificare come si muovono. Definire i contenuti, in termini di informazione, di servizio e (se previsto) di vendita. Verificare la reperibilità dei contenuti. Identificare le funzionalità necessarie. Previsione dei sistemi, risorse umane ed eventuali risorse tecnologiche per rispondere a specifiche esigenze di aggiornamento, di controllo o di servizio. Definire quali sono i pubblici del progetto e come devono essere soddisfatte le esigenze di ciascuno. Stabilire la tipologia di fruizione. Individuazione delle particolarità di accesso alla rete, livello di esperienza nell'uso di *Internet*, uso di *browser* e piattaforme, richieste di servizio da parte del *core target* del sito. Decidere i tempi, come le scadenze per le varie fasi. Definire il *budget* per il progetto iniziale (con adeguate riserve per eventuali cambiamenti o imprevisti e per la continuità).
- **Servizio e gerarchia dei valori.** Definire il messaggio principale del progetto (perché qualcuno dovrebbe visitare il sito?). Definire la tipologia di informazione e funzionalità. Identificare le connessioni logiche e le principali aree del sito. Stabilire con precisione tutte le necessità in termini di interrogazione dei *database* e altre funzionalità di servizio o interattive. Identificare eventuali risorse specializzate per funzionalità particolari o altre necessità del sito. Identificare il posizionamento del progetto, il nome, la filosofia di comunicazione. Definire l'architettura.
- **Progettazione.** Decidere lo stile visivo da dare al progetto (ambiente, colori, metafore). Definire l'interfaccia sulla base dall'architettura. Realizzare (o reperire) il corredo iconografico: illustrazioni, fotografie, icone, pulsanti. Iniziare il lavoro di *Web* copy delle pagine principali e dell'interfaccia. Realizzare una prima demo per la verifica della funzionalità dell'interfaccia. Definire tutte le tipologie di pagina necessarie. Realizzare i prototipi per tutte le necessità di interazione (ricerche, interrogazioni, acquisti, richiesta di informazioni, o altro ancora previsto dalle funzionalità del sito). Realizzare una seconda demo che simuli tutte le situazioni possibili sul sito. Verificarla anche con le varie funzioni all'interno dell'impresa (e per quanto possibile anche all'esterno) che dovranno interagire. Individuare test di usabilità e mettere a punto il progetto sulla base dei test.
- **Sviluppo.** Identificare e risolvere tutti i problemi di compatibilità e funzionalità rilevati nelle fasi precedenti . alizzare tutti i testi. Reperire o realizzare il materiale iconico eventualmente mancante. Implementare e testare le parti dinamiche e di interattività, verificando che tutto sia funzionante (interrogazioni, carrello d'acquisto, e altro che preveda risposte create dal *server*). Completare la realizzazione di tutte le pagine. Verificare correggere le pagine prima della pubblicazione. Pubblicare tutto il sito online in una *directory* non accessibile al

pubblico. Controllare *on line* tutte le pagine, tutti i *link* e tutte le funzionalità previste. Invitare a visitare il sito in anteprima persone (interne ed esterne all'impresa) che non hanno seguito le fasi precedenti del progetto - e tener conto delle loro esperienze, osservazioni ed eventuali difficoltà. Pubblicazione.

- **Verifica ed evoluzione.** Individuare persone esterne all'impresa che visitano il sito e tener conto delle loro esperienze, osservazioni ed eventuali difficoltà. Periodicamente, individuare possibili siti *competitors* e confrontarli col proprio sito, possibilmente attraverso una griglia di parametri. Intervenire in tutte le fasi precedenti secondo necessità.

Valutare un sito Web

I fattori che possono generare disaffezione e l'allontanamento dal sito *Web*, imputabili alla struttura del sito stesso, sono di varia natura. I principali sono legati alla impossibilità di fruire delle risorse del sito e alla lentezza nell'accedervi. Il famigerato HTTP *Error* 404 *Not Found* è abbastanza disarmante: l'utente non riesce a trovare l'informazione perché il *link* non porta a nessuna risorsa e quindi cerca altre strade che gli facciano trovare ciò che ricerca. Questo errore nasce spesso in seguito a una modifica di un **URL** non correttamente riportata a tutti i *link* che si richiamano a essa: capita spesso con risorse esterne, mentre con risorse interne non dovrebbe mai accadere (se accade è visto in modo negativo e rimanda al pressappochismo progettuale).

diagramma con i fattori che contribuiscono all'aggiornamento di un sito

Quando il tempo di scaricamento di una pagina è troppo elevato, ad esempio raggiunge i 30 secondi su una linea con modem a 28.8 Kbps, l'utente abbandona, o se non può si arrabbia. Ad esempio l'uso indiscriminato di applet, bottoni grafici che rispondono al passaggio del *mouse* e immagini non ottimizzate è da evitare il più possibile, proprio per non appesantire la consultazione. Diminuendo e ottimizzando questi contenuti il sito non perde assolutamente in forza d'impatto, anzi diventa più navigabile e l'utente può meglio usufruire delle informazioni contenute.

Il costante aggiornamento del sito *Web*, pratica che dovrebbe essere fatta da tutti quelli che ne hanno uno, porta alla generazione di pagine obsolete, il cui contenuto ha ormai perso di validità. Le strade che un amministratore può seguire sono generalmente due:

archiviarle, indicando chiaramente che si tratta di materiale vecchio, o eliminarle, se il loro contenuto è realmente inutile. Il controllo sulla presenza di pagine vecchie è necessario perché l'utente che cerca informazioni non deve venire a contatto con dati superati, a meno che non vengano ricercati volontariamente.

Il costante controllo dei fattori critici è importante per avere un sito efficiente e di facile consultazione. Chi si occupa del sito può far affidamento su vari strumenti per evitare la presenza dei problemi più comuni, ad esempio molti *editor* **HTML** hanno specifiche funzioni per "pesare" le pagine. ***compuware***, forte della sua esperienza nel campo del *testing* delle applicazioni, ha creato **webcheck.it**, un servizio a disposizione delle aziende per analizzare il proprio sito *Web*. La peculiarità di questo servizio è che agisce da remoto, cioè non serve fermare il sistema per permettere l'analisi del sito. Ma molto del lavoro deve essere fatto prima dal *Web master*. Una analisi accurata ed una attenta organizzazione dei contenuti, la separazione dei contenuti statici e delle componenti attive, contribuisce a creare ordine in un *server* ed evitare problemi di inefficienza o mancanza di scalabilità.

Strumenti di produzione

Cosimo Laneve

Strumenti

Gli strumenti più utilizzati per produrre pagine *Web* sono essenzialmente due:

- *Netscape Composer*, gratuito e scaricabile da netscape.org assieme al *browser Netscape*;
- *Microsoft FrontPage*, a pagamento, fornito da *Microsoft* nel pacchetto *Office*.

Questi strumenti sono molto semplici perchè visualizzano la pagina che si sta editando nello stesso modo come sarà visualizzata dal *browser*, una tenica detta **WYSIWYG** (*what you see is what you get*). Uno strumento simile a *Microsoft FrontPage* è Adobe GoLive, un programma che nasce su piattaforma *Macintosh*, ma disponibile anche per *Windows*. Infine si segnala Macromedia Dreamweaver un programma che non è dotato di funzionalità potenti per quanto riguarda la gestione dei siti, ma che è forse il migliore per quanto riguarda la grafica e le animazioni.

Accanto a questi strumenti, ci sono i vari editori (ad esempio **Notepad** o **Wordpad** che sono forniti gratuitamente con il sistema operativo *Windows*, oppure *emacs* o *vim* per i sistemi *Unix* o *Linux*) che consentono di editare in modo diretto *file* in formato **HTML**. Perciò tali sistemi presuppongono la conoscenza del linguaggio **HTML** (che sarà discusso in seguito) e sono quindi di non facile utilizzo per un principiante. Un sistema intermedio tra gli editori *standard* e quelli per pagine *Web* è **Word**, il noto editore a pagamento fornito dalla *Microsoft* nel pacchetto *Office*. *Word* consente il salvataggio dei documenti in formato **HTML**. A tal fine, è sufficiente utilizzare l'opzione salva come e scegliere pagina *Web*. La trasformazione da documento *Word* a documento **HTML** comunque fa perdere alcune caratteristiche della pagina editata (poichè non sono visualizzabili dai *browser*). Ciò si manifesta con un opportuno messaggio di allarme quando si tenta di salvare.

Netscape Composer

*Netscape Composer*è un editore di pagine *Web* facile da usare (quanto un comune *word processor*) che è incluso in *Netscape Communicator*. Il *Composer* utilizza fonti, stili, paragrafi, liste, tabelle e un analizzatore lessicale.

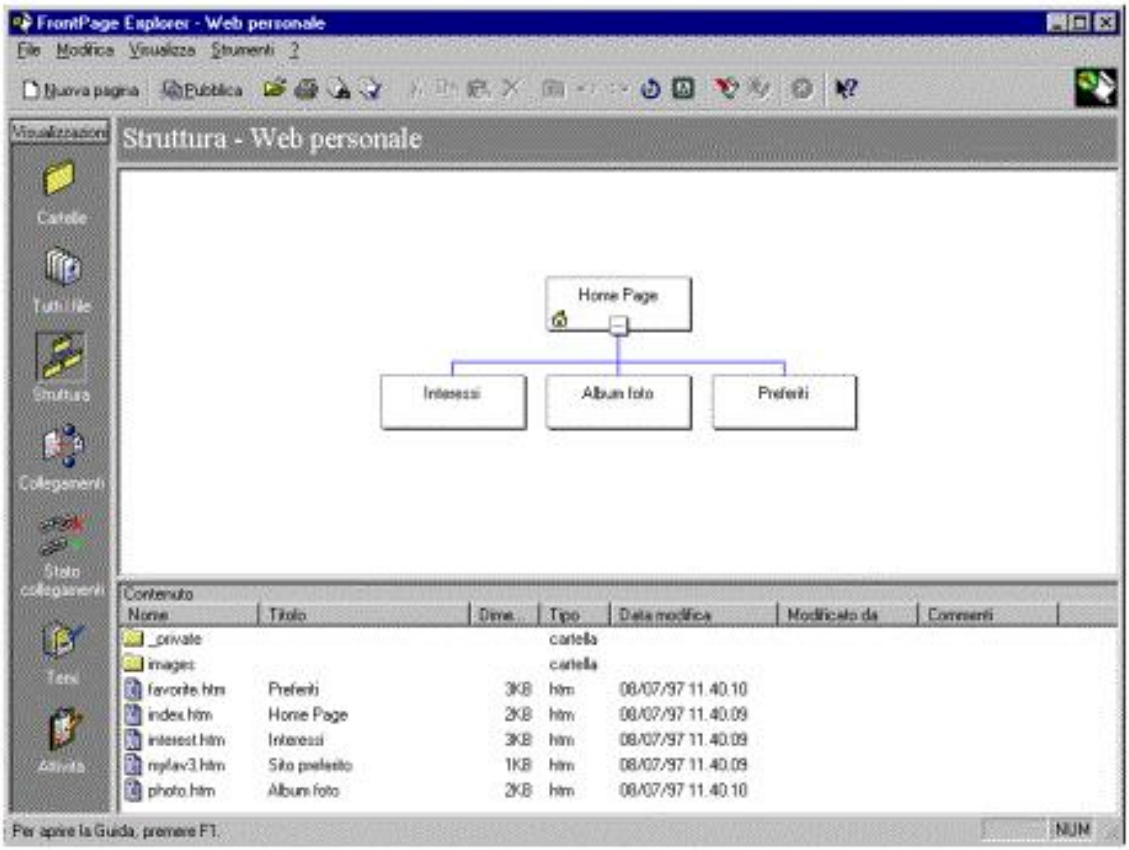
*Netscape Composer*può essere esteso con ulteriori funzionalità, e quindi personalizzabile, mediante *plug-ins* di *Netscape Communicator*. Infine esso rende semplice la pubblicazione dei documenti in una intranet o su *Internet*.

Pagina pubblicitaria di Netscape Compose

Microsoft FrontPage

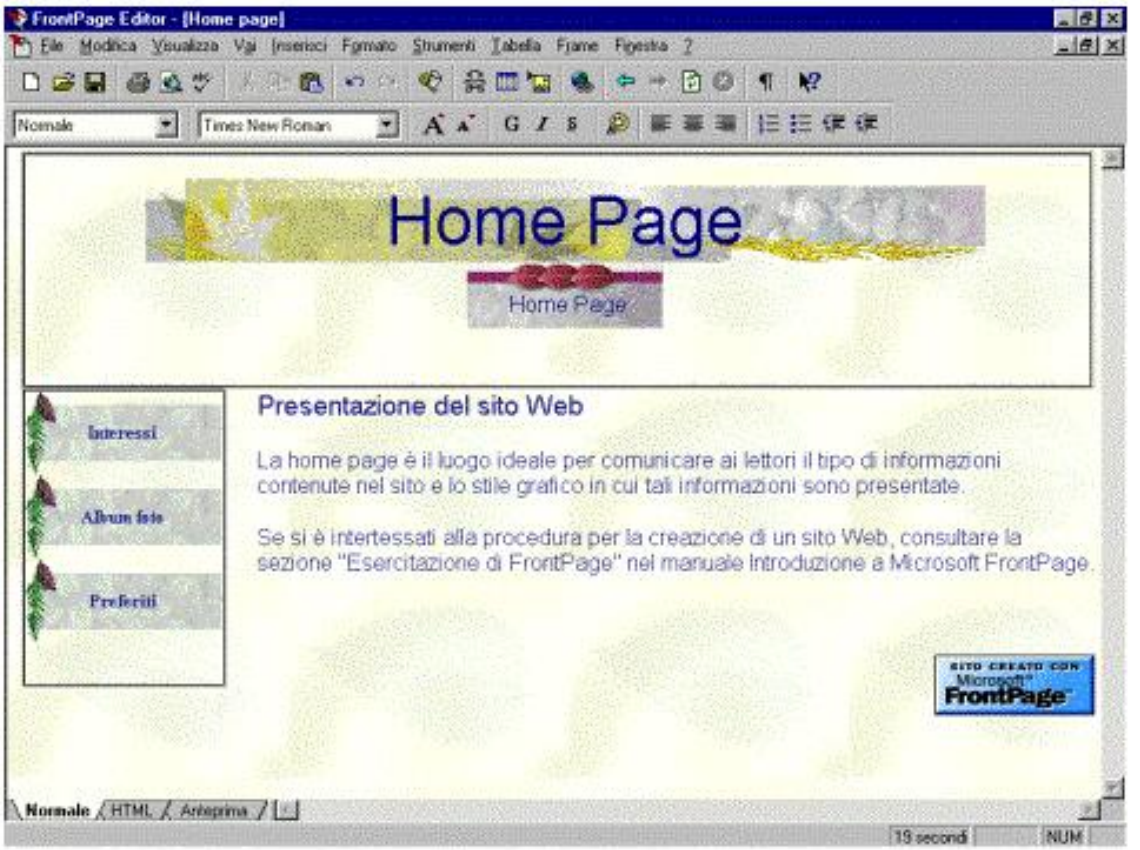
Microsoft FrontPage consente di creare e gestire le informazioni da pubblicare sul *World Wide Web* organizzandole in modo strutturato. Le strutture create da *FrontPage*, dette ***Web* di *FrontPage***, contengono tutte le pagine, le immagini e i *file* che costituiscono un sito *Web*. Utilizzando *FrontPage Explorer* e *FrontPage* Editor è possibile creare pagine e siti *Web* direttamente su un *computer* con sistema operativo *Windows* o *Macintosh*.

In *FrontPage Explorer* sono disponibili più modalità di visualizzazione che consentono di osservare e modificare in modi diversi il contenuto dei *Web* di *FrontPage*.



struttura di un sito con Microsoft FrontPage

FrontPage Editor è il *word processor* di pagine *Web* che è completamente integrato con *FrontPage Explorer*. Il testo, le tabelle, le immagini, i moduli, i controlli e gli altri elementi delle pagine vengono visualizzati in modalità **WYSIWYG**, ovvero esattamente come appariranno nel *browser Web*.



la visualizzazione di una pagina con FrontPage

È utile segnalare che esiste una versione gratuita di *FrontPage*, sebbene con diverse funzionalità in meno, detta **FrontPage Express**, che viene distribuita insieme al *browser Explorer*.

Valutare software per la realizzazione di pagine Web

I *software* per realizzare pagine *Web* dovrebbero fornire le seguenti funzionalità:

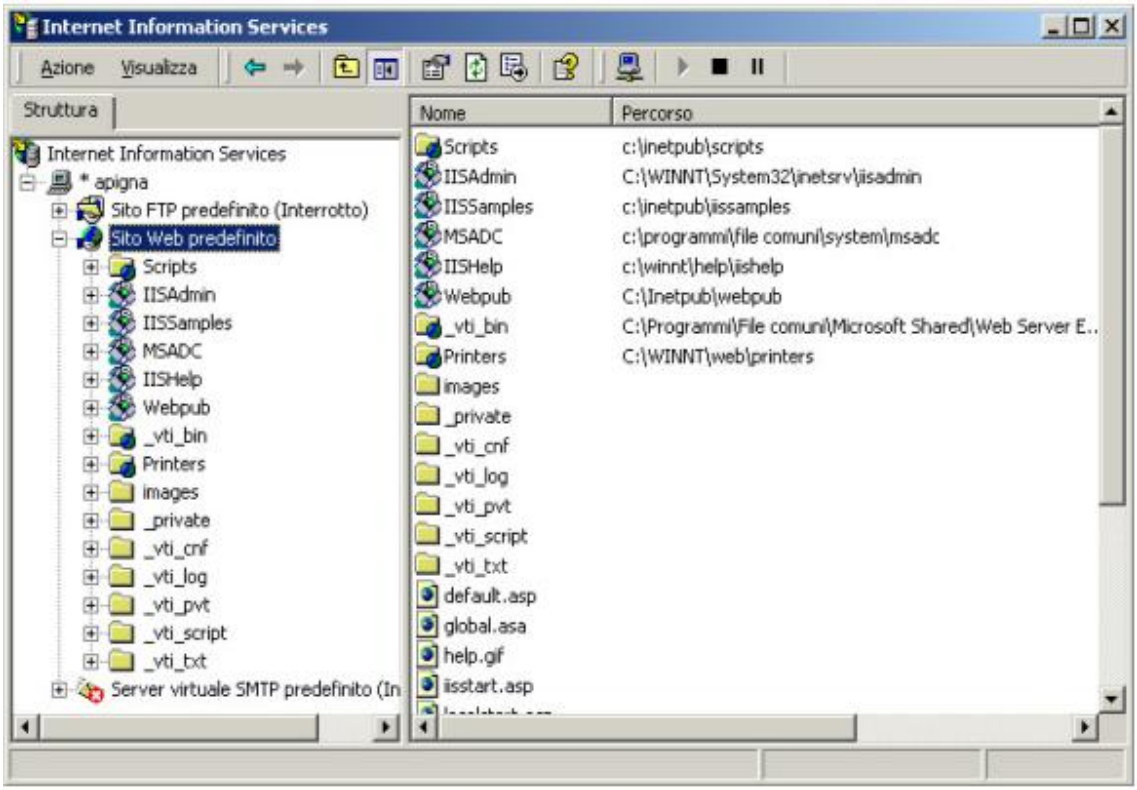
- un buon **editore**, che consenta inclusioni di tabelle, immagini, suoni e video. L'editore dovrebbe anche garantire l'accesso contemporaneo sia alla visualizzazione con un *browser* che al codice html.
- **Creazione e gestione di mappe** complete dei collegamenti ipertestuali contenuti nei *file* di un sito. Se un sito viene copiato da un *server* a un altro, la corrispondente mappa dei collegamenti ipertestuali verrà rielaborata. Se si sposta una pagina da una cartella a un'altra dello stesso sito, bisognerebbe aggiornare automaticamente tutti i collegamenti ipertestuali alla pagina spostata eventualmente contenuti in altre pagine o documenti dello stesso sito.
- **Amministrazione**. Possibilità di specificare i nomi degli utenti autorizzati ad amministrare, modificare o esplorare il *Web*. Creazione e gestione di elenchi attività in cui vengono specificate le operazioni da svolgere per completare un *Web*.

Installazione e configurazione di software per la

produzione di pagine Web: Microsoft IIS

I sistemi *Microsoft IIS* (4 e 5) sopportano due possibili interfacce grafiche: un *browser* e la *Microsoft Management Console*. Quest'ultimo è più flessibile ed lo strumento che andiamo ad analizzare.

La *Microsoft Management Console* (MMC) permette di lavorare, come se si stesse usando Esplora Risorse. Infatti ritroviamo la stessa disposizione grafica degli oggetti, cioè a sinistra un albero di possibili risorse da esplorare, a destra ritroviamo il contenuto della risorsa che abbiamo selezionato. Per aprire questa console è necessario premere Avvio ed andare in Programmi/*Option* Pack4/ *Internet Information Server* 4.0/*Internet Information Manager*, la prima volta che si compirà questa operazione sulla sinistra apparirà una sola cartella chiamata *Console Root*. Cliccando sopra questa si espanderà un albero che a seconda dei componenti istallati conterrà un numero variabile di cartelle, tali cartelle prendono il nome di *Snap-in* e sono i vari strumenti che abbiamo appunto installato. Sotto *Windows* 2000 è possibile accedere a IIS5 attraverso il pannello di controllo e nei servizi di amministrazione scegliere la voce Gestione servizio **Internet** *Microsoft*.



Internet Information Service di FrontPage

Normalmente dovrebbero apparire almeno due *Snap-in*: Internet *Information Server* e *Microsoft Transaction Server*; se apriamo il sottoalbero che parte da IIS4 avremo l'icona di un PC (quello su cui è istallato OP4) e una serie di cartelle come il *Default FTP*, *Default Web site* e *Administration Web Site* (ovviamente la presenza o meno di queste dipende da cosa si è scelto di istallare). Se andiamo col *mouse* a selezionare una di queste cartelle e poi, secondo la filosofia *Microsoft*, premiamo il tasto destro,

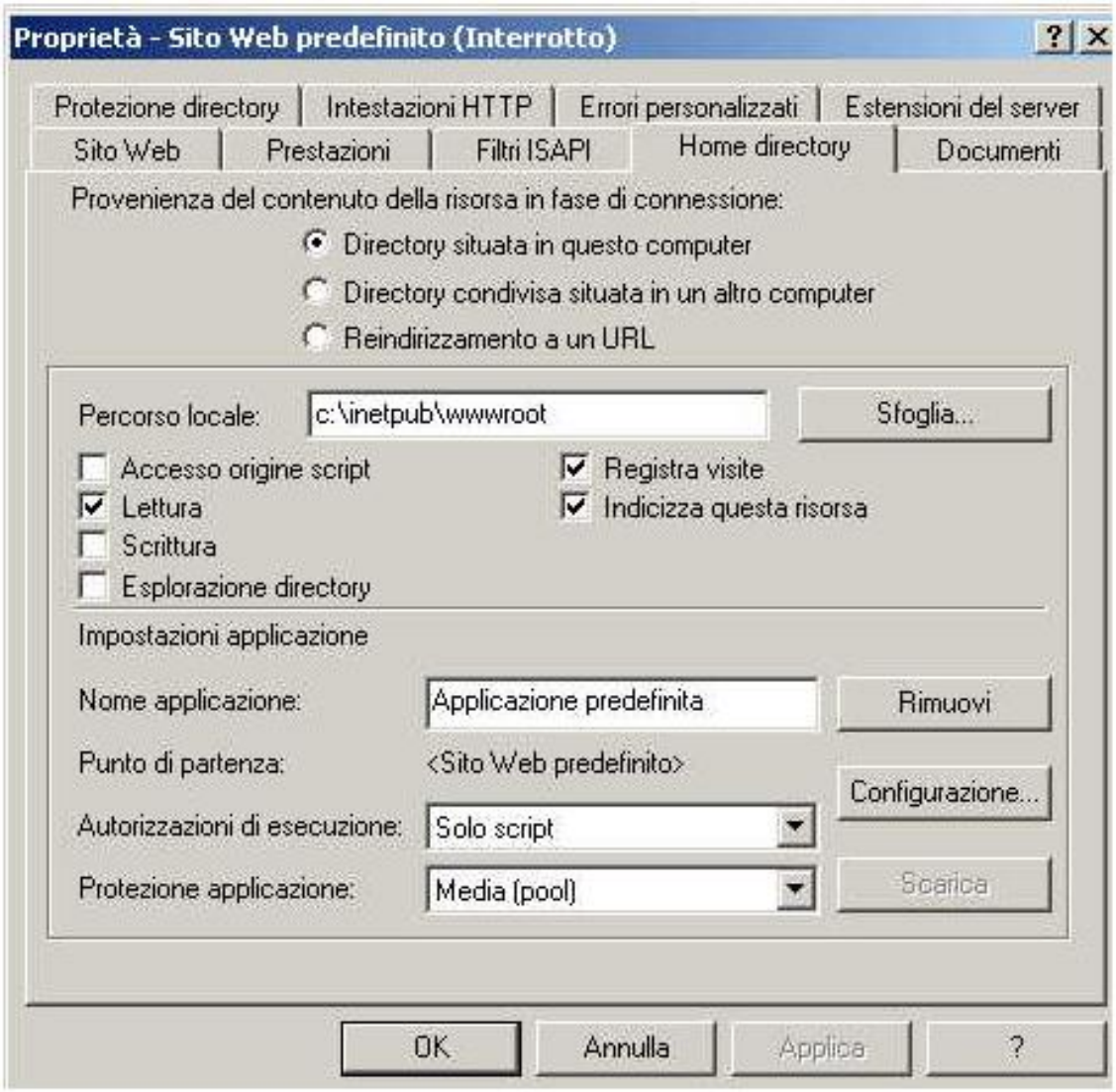
otterremo l'apertura di un menu a tendina che ci elenca le varie possibilità di interagire con l'oggetto selezionato.

Se si è scelto il *Default Web Site* potremo, tra le altre cose avviare, fermare o mettere in pausa questo servizio altrimenti potremmo visualizzarne le Proprietà. Questo ci sarà molto utile in seguito poichè è proprio dal menù che si ottiene da Proprietà che riusciremo a configurare il nostro sito in maniera opportuna.

Creazione di un sito *Web*

Prima di vedere i semplici passi che porteranno alla costruzione di un sito *Web*, è bene commentare le modifiche effettuate dall'istallazione di IIS. Infatti viene creata, nella *root* di istallazione (il disco C:\ nel nostro caso), una *directory* INETPUB in cui sono incluse le FTPROOT, WWWROOT, SCRIPTS e IISSAMPLES: le prime due sono le *home directory* dei due siti di *default* creati automaticamente, la terza è la *directory* di *default* per gli scripts e la quarta quella che contiene le pagine del sito di esempio. Veniamo ora alla procedura per la creazione di un sito *Web*:

- **apertura di IIM**. Sul lato sinistro deve essere completamente visibile il sottoalbero che parte dallo *snap-in* IIS; in questa maniera si potrà visualizzare l'icona con il nome della macchina su cui si sta lavorando (se si clicca due volte si potranno vedere tutti i siti che in questo momento stanno girando o meglio che sono presenti).
- **Creazione di un sito**. Selezionando il nome della macchina e premendo il tasto destro si apre un menù a tendina la cui quarta voce è *New*: selezionandola si apre una seconda finestra a tendina; da questa si scelga *Web site*.
- ***Wizard* di creazione del sito *Web***. Viene attivato automaticamente dalla procedura sopra elencata. Nella prima finestra di dialogo dovremo immettere il nome del sito che si desidera creare, nella seconda si deve immettere l'indirizzo *IP* a cui si vuole far corrispondere il sito e la porta a cui il *server* fa riferimento (per un sito www di *default* è la porta 80), nella terza bisogna specificare dove fisicamente è collocata la *directory* del sito(di *default* C:\INETPUB\WWWROOT) e se si permette l'accesso anonimo, infine, dovremo configurare i permessi di accesso(ad es. sola lettura sola scrittura esecuzione o permettere gli *script* ed infine se permettere il *browsing* della *directory*).



Configurazione dei permessi di accesso in IIS

Dando una occhiata al IIM, si può ora notare il nuovo sito che si trova in stato di *Stop*. Per avviarlo, cioè per poter permettere la navigazione da parte di un utente esterno, bisogna premere l'icona *Start* che si trova sulla barra dei comandi di IIM, oppure, premere il tasto destro (una volta selezionato il sito che si vuole attivare) e poi *Start*.

Creazione di una *Virtual Directory*

Una volta creato il sito abbiamo bisogno di una *home page* da far visualizzare. Per fare questo dobbiamo creare una pagina **HTML** e salvarla nella *Home directory* che abbiamo inserito nella terza finestra del passo 3. Sarebbe comodo, tuttavia, avere diverse *directory* in cui poter mettere le immagini visualizzate nella *Home* page o dei documenti che vogliamo mettere a disposizione di quanti facciano visita al nostro sito. Con l'uso delle *virtual directory* è possibile mappare nell'albero delle *directory* del sito delle cartelle che fisicamente stanno da qualche parte nella macchina o addirittura su di un altro *server*. Attraverso l'uso di questa tecnica e di IIM possiamo vedere le cartelle che stanno in luoghi diversi come se fossero tutte sottocartelle della *home directory* del

sito creato, rendendo la gestione del sito *Web* più semplice. Per creare una VrtDir dobbiamo eseguire la seguente procedura:

- aprire IIM, andare sul nome del sito testè creato, premere il tasto destro ed andare su Nuovo.
- Scegliere *Virtual Directory*, si aprirà così un *wizard* di creazione.
- Il primo passo consiste nello scegliere un alias per la VrtDir; questo sarà il nome con cui la troveremo all'interno dell'albero delle cartelle.
- Il secondo passo è quello di specificare dove fisicamente è collocata la dir (cioè il percorso assoluto di questa).
- In questo terzo ed ultimo passo si specificano i permessi di accesso alla *directory* (cioè lettura e scripts per *default*) premendo fine si completa il processo.

Guardando, al solito la parte sinistra di IIM si può notare la nuova *directory* come sottocartella del nostro sito, in maniera analoga si può procedere per la creazione di altre cartelle.

Sviluppo di pagine e siti

Cosimo Laneve

Creare pagine Web

HTML è l'acronimo di *HyperText Markup Language*; si tratta di un linguaggio utilizzato per la marcatura ipertestuale. A differenza dei comuni linguaggi di programmazione, **HTML** non serve a implementare programmi, funzioni o *subroutine* in grado di tradurre un algoritmo di calcolo, ma viene impiegato per definire le caratteristiche di pagine ipertestuali. Le istruzioni di **HTML** impostano gli attributi relativi agli oggetti utilizzati nella pagina, ovvero ne determinano la posizione, il formato, la visibilità e le altre proprietà che risulta possibilie impostare.

la gerarchia dei linguaggi di markup

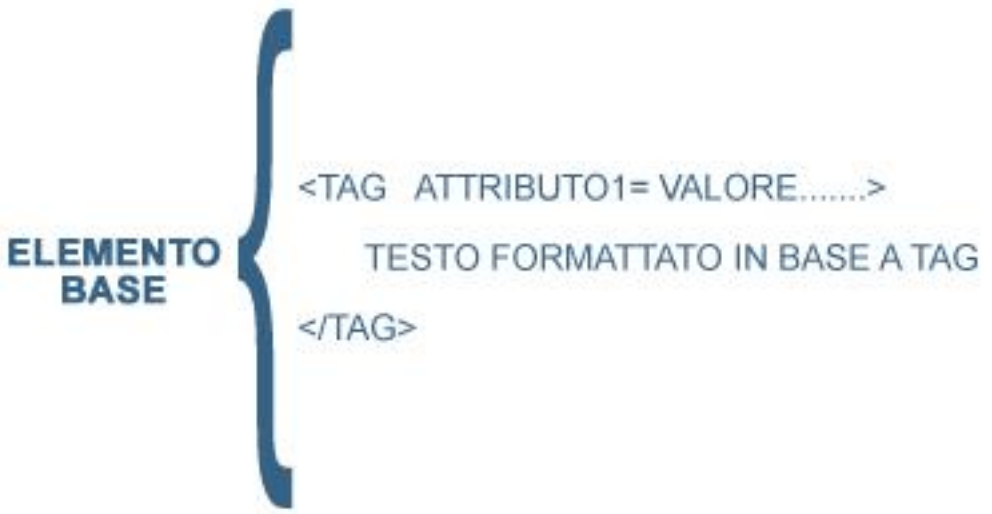
HTML si basa sulla sintassi dello *Standard Generalized Markup Language* (**SGML**), un metalinguaggio per la definizione di sistemi di marcatura. Attraverso il linguaggio **HTML** è possibile inserire gli iperlink, ovvero dei collegamenti ancorati a oggetti posizionati all'interno della pagina (testuali e non), che permettono di saltare da una pagina all'altra del medesimo sito o di siti remoti. Risulta allora evidente che **HTML** permette di realizzare un ipertesto, ovvero un testo la cui lettura tipicamente può procedere secondo diversi percorsi determinati dal programmatore, ma scelti di volta in volta dal fruitore delle pagine.

In questa parte saranno discussi concetti introduttivi di **HTML**. Per I concetti più avanzati si rimanda all'appendice.

Usare HTML per creare e aggionare pagine Web

Durante la scrittura di una pagina occorre seguire le regole sintattiche del linguaggio. Tuttavia, **HTML** è presente in **Internet** in diversi dialetti, ciascuno dei quali risulta ottimale per un determinato *browser*. In tal modo la stessa pagina potrà essere visualizzata in modo diverso Da *MS Internet Explorer* o da *Netscape Navigator*. Può accader inoltre che una determinata struttura sintattica (si parlerà di marcatori, o tag)

non sia riconosciuta affatto da alcuni *browser*. Anche in questo caso la pagina sarà visualizzata senza alcuna generazione di errori, ma semplicemente ignorando l'elemento sconosciuto. Si usa dire a proposito del linguaggio **HTML**, che occorre essere rigorosi nella scrittura del codice letto da *browser* che risultano essere tolleranti.



struttura dei tag HTML

Così come avviene per linguaggi di programmazione di alto livello, anche la scrittura di una pagina **HTML** può essere realizzata attraverso l'uso di indentazione o altre modalità che facilitino la lettura del codice. Tuttavia il *browser* non interpreterà invii a capo o inserimento di spazi multipli (a meno che si faccia uso di un particolare marcatore di preformattazione o si inseriscano 'fisicamenté degli spazi), leggendo il testo scritto sequenzialmente senza soluzione di continuità. Come detto se vengono inseriti più spazi, essi vengono visualizzati in un solo elemento. Si parla cioè di spazi collassati.

Struttura della pagina

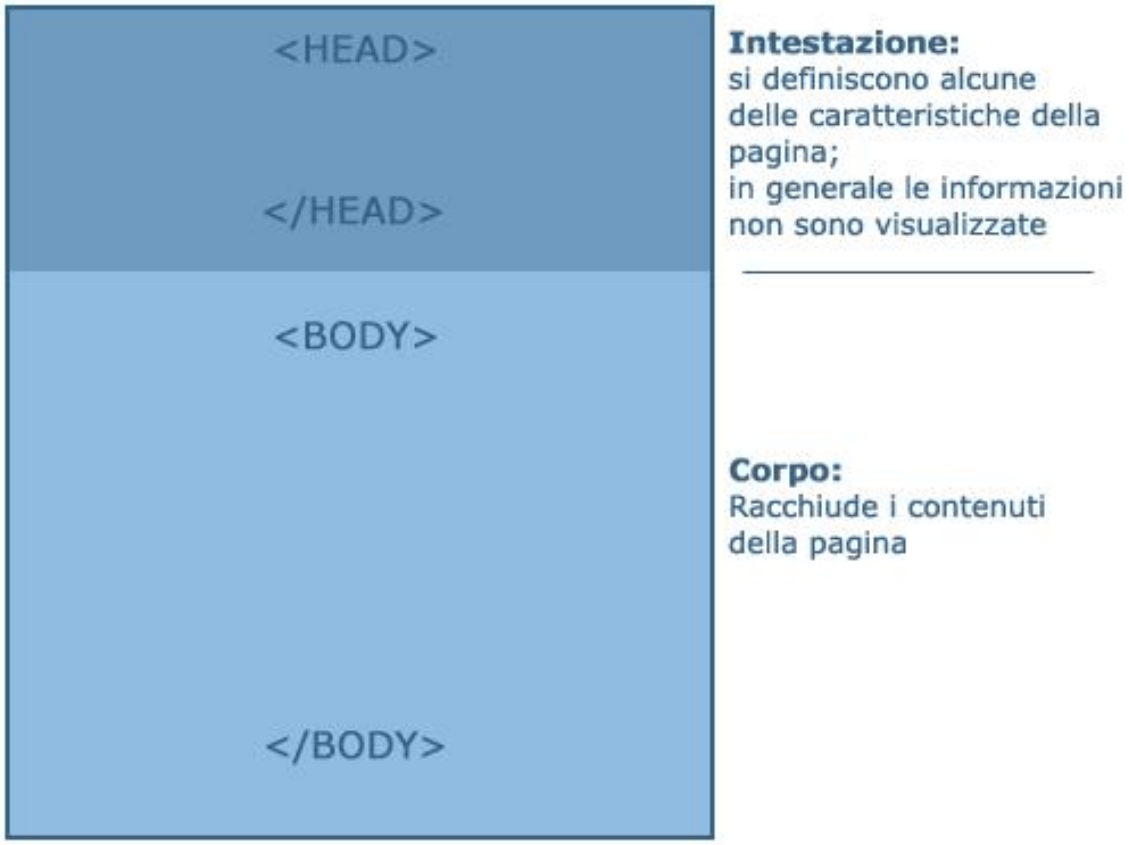
Una pagina ipertestuale scritta in **HTML** si può considerare composta da due sezioni: una parte di intestazione, all'interno della quale vengono definite alcune caratteristiche generali della pagina, e il corpo della stessa pagina, ove si troveranno tra l'altro i contenuti da visualizzare. Le due sezioni devono essere scritte in modo sequenziale e racchiuse da marcatori di inizio e fine sezione; entrambe le sezioni saranno quindi delimitate da marcatori di inizio e fine pagina. Un marcatore è inserito attraverso la propria sintassi, che prevede la digitazione del proprio nome tra parentesi angolari; per l'elemento di chiusura bisognerà far precedere il nome dal simbolo /.

E' possibile inoltre inserire dei commenti, ovvero delle descrizioni che non hanno effetto sulla visualizzazione della pagina. Per inserire un commento occorrerà utilizzare la seguente sintassi.

<!-- stringa di commento -- >

In sostanza la struttura di una pagina **HTML** è la seguente.

<HTML>



struttura di un documento html

Come per alcuni linguaggi di programmazione tradizionali, il codice **HTML** va scritto preferibilmente usando una indentazione che tuttavia non ha alcun effetto sulla presentazione della pagina, ma può risultare uno strumento che facilita la lettura del listato (per esempio aumentando il grado di indentazione quando si passa a un livello inferiore di marcatura, ovvero si inserisce un tag che deve essere incluso nel precedente).

I marcatori principali

- **Il marcatore di pagina**. I marcatori di inizio e fine pagina sono rispettivamente i seguenti:

  <HTML> </HTML>

  tale marcatore ha la funzione di delimitare gli elementi costitutivi della pagina stessa. Tutti gli altri marcatori (compresi quelli di intestazione e di corpo) dovranno essere compresi tra tali tag.

- **Il marcatore di intestazione.** La sezione di intestazione è delimitata dai marcatori:

  <HEAD> </HEAD>

Si è detto che l'intestazione serve a definire alcune caratteristiche generali, che per la gran parte non risultano visibili all'interno della pagina o non risultano comunque indispensabili. Tra i marcatori di intestazione possono comunque essere inserite le seguenti informazioni: Titolo, Meta **Tag**, Elementi di stile, *Script*. Gli stili permettono di impostare le caratteristiche che riguardano il formato degli oggetti contenuti in più pagine. Gli *script* sono frammenti di codice che implementano funzioni diverse, come per esempio quelle che forniscono dinamicità alle pagine. All'interno dell'intestazione possono essere inseriti i marcatori che servono a porre informazioni non visibili all'interno del corpo della pagina; rientrano in questa categoria il titolo, i marcatori che definiscono caratteristiche di pagina (META per la definizione delle parole chiave, eccetera), elementi di stile o *script* che devono essere posti in esecuzione.

- **Il marcatore di titolo**. I marcatori di inizio e fine titolo sono rispettivamente i seguenti.

  <TITLE> </TITLE>

  Questo marcatore dovrà essere posto internamente al marcatore di intestazione. Con questo si definirà la stringa che sarà visualizzata nella barra del titolo del *browser*. Alcuni motori di ricerca utilizzano il contenuto del marcatore di titolo per ricevere informazioni (es. *hot words*) sulla pagina corrente.

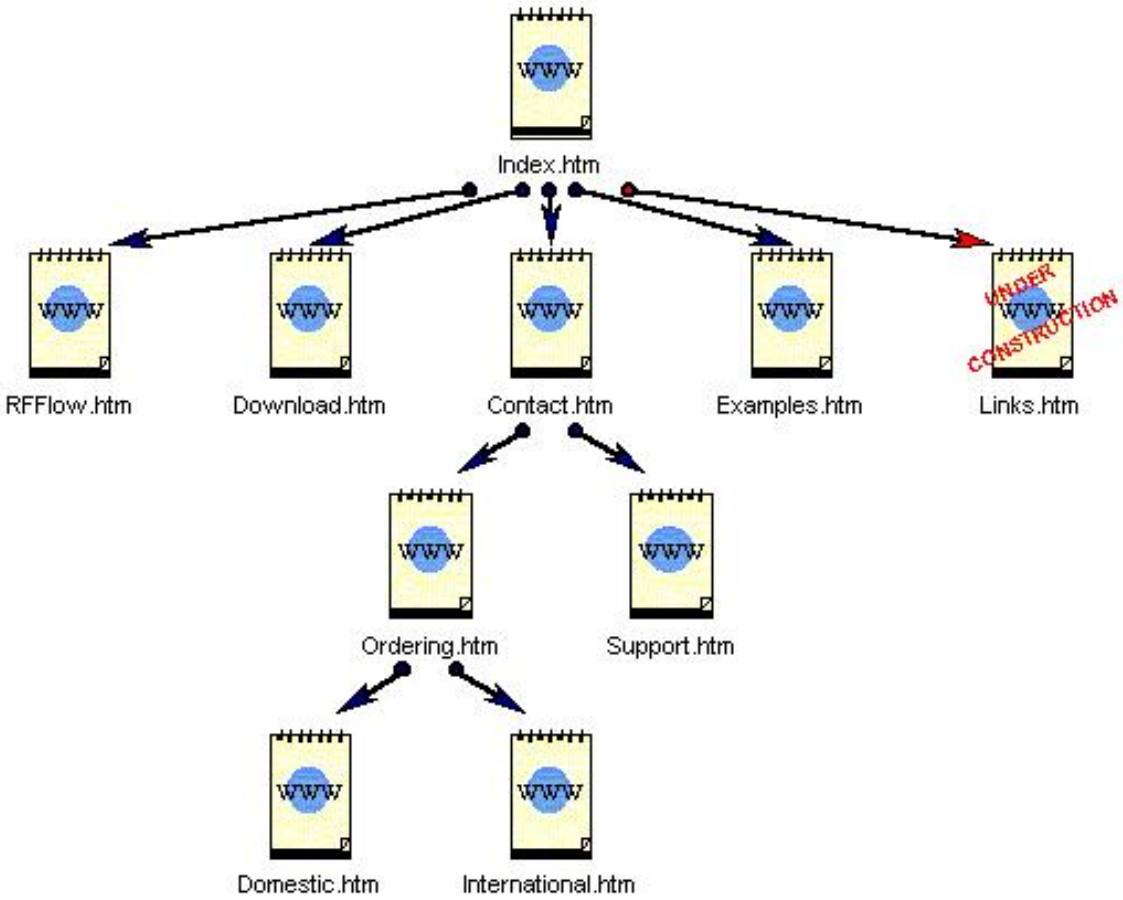
Link

L'uso dei *link* permette di effettuare riferimenti a *file* remoti o ad altre parti del documento stesso (ad esempio per spostarsi velocemente da un indice alla voce interessata). Questo tag serve perciò a creare *link* tra il documento e un immagine o animazione o *file* sonoro. La sintassi del **Tag** è:

<A HREF ="file_name" (in formato URL)> testo che fa da collegamento </A>

Esempio: <A href="prova.html"> Questo è il *link* a prova.html </A>

Questo esempio crea un *link* con il *file* ipertestuale prova.html. Nel documento il *link* appare come la frase Questo è il *link* a prova.html sottolineata o evidenziata (a seconda del ***Browser***) cliccando sulla frase verrà caricato il nuovo documento prova.html. Si consiglia di usare sempre *path* relativi in caso di spostamento dei *file* si avranno meno problemi.



una gerarchia di documenti HTML collegati da link

Formato dell'URL

Per effettuare il richiamo di qualsiasi *file* in Rete e non, si usa il formato **URL** (*Uniform Resource Locator*). Il formato é cosi' composto:

Risorsa://SERVER[:PORT]/Path/Filename.

dove:

- Risorsa é una delle seguenti:
  - **file** = specifica che l'oggetto in esame é un *file* e verrà prelevato dal *filesystem* locale del *browser*.
  - **http** = *file retrieval* da un WWW *server*.
  - **gopher** = *file retrieval* tramite un *Gopher server*.
  - **WAIS** = *file retrieval* tramite *WAIS server*.
  - **news** = *news server*.
  - **telnet** = *file retrieval* via telnet.
  - **mailto** = lancia l'applicazione per la posta elettronica con destinatario l'argomento della risorsa.
  - **ftp** = ftp *server*, consente di scaricare *file* residenti su siti remoti.
- **SERVER** é l'indirizzo *IP* (o nome DNS) dell'*host* dove il *file* è localizzato.

- **Port è** la porta da usare nella connessione. Opzionale.

Riassumendo: la prima parte dell'**URL** (prima delle due sbarre (//)) specifica il metodo di accesso. La seconda é tipicamente l'indirizzo dove sono locati il *computer*, i dati o i servizi richiesti. Le parti successive specificano gli eventuali nomi di *file*, di porte di connessione o il testo da cercare in un *database*. Ecco alcuni esempi di **URL**:

- file://www.iol.it/suono.au
- file://www.iol.it/picture.gif
- file://www.iol.it/directory
- http://www.iol.it/directory/book.html
- ftp://www.iol.it/pub/file.txt
- gopher://www.iol.it
- telnet://www.iol.it:1234
- news:alt.hypertext
- mailto:mariorossi@libero.it

La maggior parte dei *browser Web* concedono all'utente di specificare un **URL** e di connettersi a quel documento o servizio. Quando si seleziona dell'ipertesto in un documento **HTML** l'utente sta in effetti spedendo la richiesta di aprire un **URL**. In questo modo gli *hyperlink* possono essere fatti non solo ad altri testi e *media* ma anche ad altri servizi di rete. I *browser Web* non sono dei semplici *client* del *Web*, ma sono anche dei *client* totalmente operativi per i sistemi FTP, *Gopher* e telnet.

Ancore e *Links* a parti specifiche di un documento

I *link* o le ancore possono essere anche utilizzate per spostarsi all'interno dello stesso documento facendo riferimenti a vari punti di questo. Supponiamo si voglia creare un *link* da un documento A ad una specifica sezione del documento B. Prima bisognerà marcare la sezione a cui poi si fara' riferimento nel modo seguente:

<A NAME="qui_Kirk">il testo che verra' visualizzato</A>

Quindi si creerà il *link* cosi':

<A HREF="...filename#qui_Kirk">...testo_visualizzato...</A>

Ove ...filename é l'**URL** che localizza il documento B e ..testo_visualizzato.. é il testo (mostrato nel documento A) che farà da *link*.

Approfondimento

Concetti avanzati di HTML

Cosimo Laneve

13.3.2 (Usare linguaggi di programmazione Web per creare e aggiornare pagine Web)

Liste

Il formato **HTML** mette a disposizione 5 **Tag** per definire delle liste, supportando liste numerate, non numerate, di descrizione, menu e *directory*.

Liste Numerate e Ordinate

Una lista numerata si crea usando i seguenti **Tag**:

- usare il **Tag** di *open list*<OL>
- usare il **Tag**<LI> seguito dall'oggetto della lista;
- iterare questo punto per ogni elemento da listare;
- chiudere la lista con il **Tag** </OL>.

Sia le liste ordinate che quelle non ordinate vengono visualizzate indentate di qualche spazio rispetto al testo normale ed un carattere particolare viene messo prima degli elementi della lista (per le liste ordinate é un num. progressivo).

Liste non numerate

Una lista non numerata si crea usando i seguenti **Tag**:

- usare il **Tag** di *open list* <UL>
- usare il **Tag** <LI> seguito dall'oggetto della lista;
- iterare questo punto per ogni elemento da listare;
- chiudere la lista con il **Tag** </UL>.

Liste di menu

Questo tipo di lista dovrebbe essere fatta di piccoli paragrafi (solitamente 1 linea per elemento). Essa viene visualizzata con uno stile piu' compatto rispetto alle liste non ordinate. Una lista di menù si crea usando i seguenti **Tag**:

- usare il **Tag** di *open list* <MENU>;
- usare il **Tag** <LI> seguito dall'oggetto della lista;
- iterare questo punto per ogni elemento da listare;
- chiudere la lista con il **Tag** </MENU>.

Questo tipo di lista dovrebbe essere composta di piccoli elementi, tipicamente di 20 caratteri. Questi possono essere incolonnati lungo la pagina. Una lista di *directory* si crea usando i seguenti **Tag**:

- usare il **Tag** di *open list* <DIR>;
- usare il **Tag** <LI> seguito dall'oggetto della lista;
- iterare questo punto per ogni elemento da listare;
- chiudere la lista con il **Tag** </DIR>.

Liste di descrizione

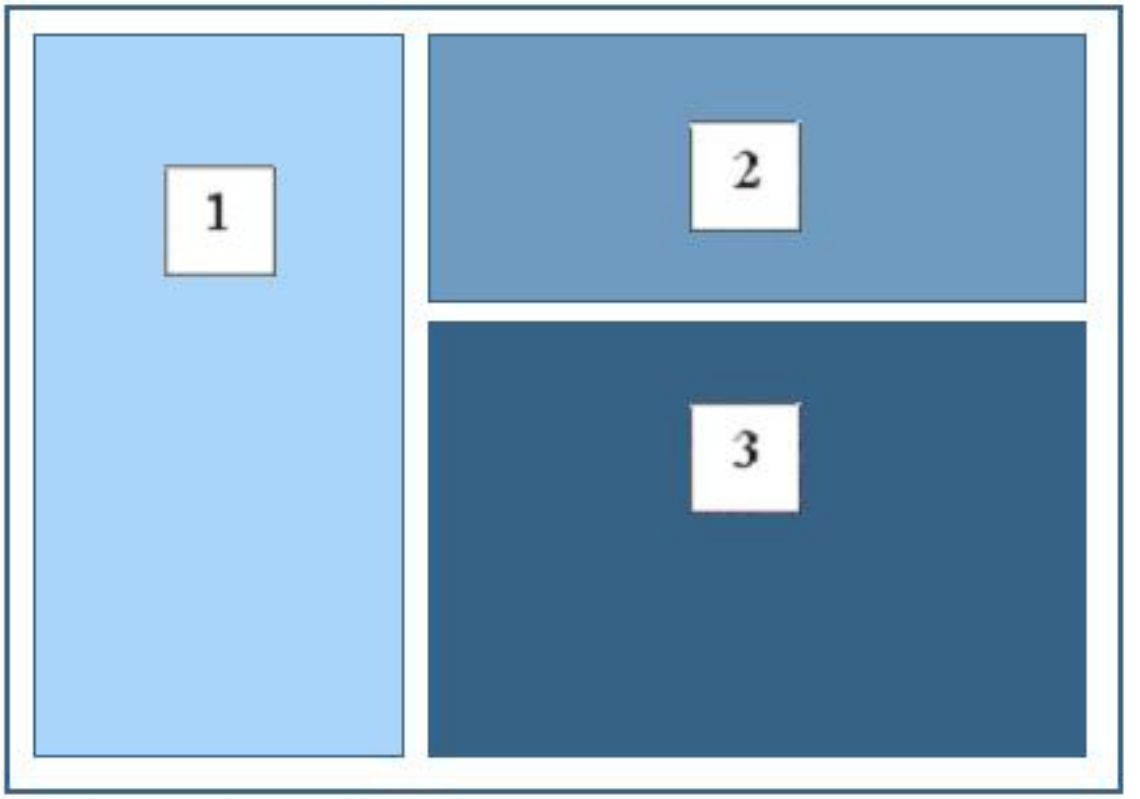
Una lista di descrizione é solitamente un alternanza di titoli (DT) e descrizioni (DD). Si crea usando i seguenti **Tag**:

- usare il **Tag** di *open list* <DL>
- inserire il titolo del primo oggetto della lista dopo il **Tag** <DT>
- inserire la descrizione del corrispondente oggetto dopo il **Tag** <DD>
- iterare questo punto per ogni elemento da listare;
- chiudere la lista con il **Tag** </DL>.

Attenzione: sono ammesse liste nidificate di qualsiasi livello ma l'*output* risultante varia a seconda del *browser* usato, e alcuni non le supportano.

I frame

E' utile impostare la pagina **HTML** in modo da suddividerla in sezioni, ciascuna delle quali è a sua volta un documento **HTML**. In tal modo, è possibile agire sulla struttura in fase di visualizzazione, per esempio, modificando il contenuto di un riquadro della pagina principale.

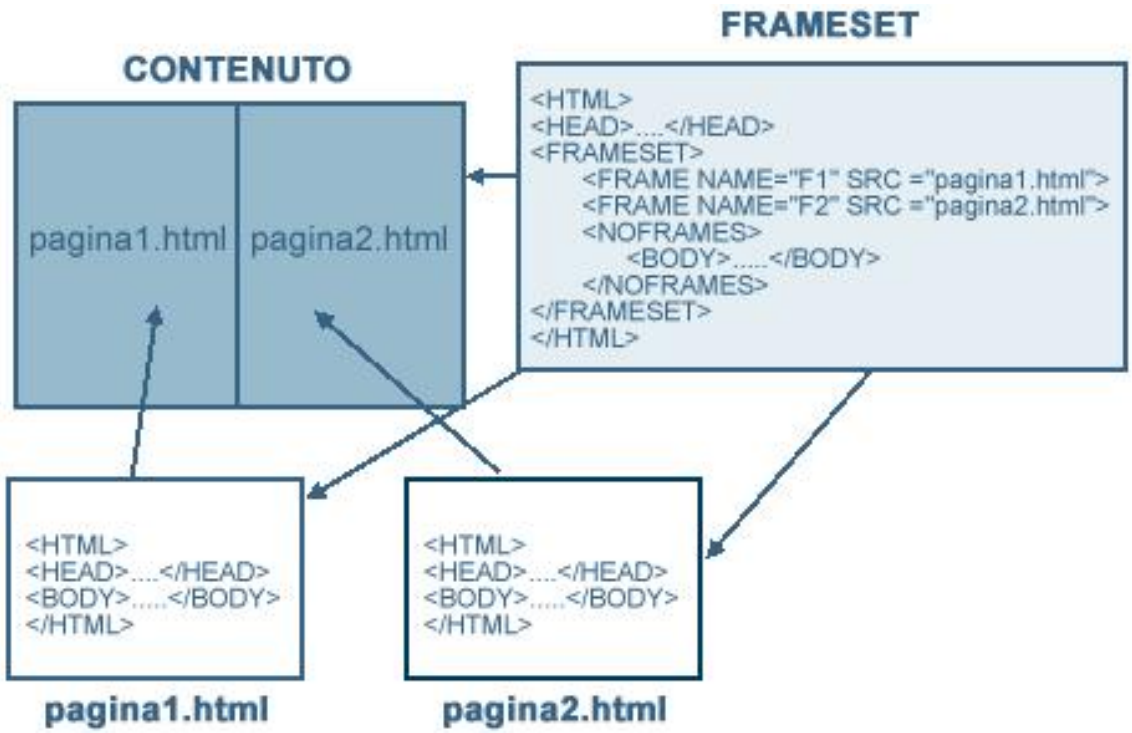


una pagina strutturata con i frame

Una situazione di questo tipo è piuttosto frequente nel caso in cui si voglia mantenere sempre disponibile un menu (per esempio nel riquadro 1) ed altre informazioni (per esempio una intestazione nel riquadro 2), mentre si vuol modificare dei contenuti visualizzati (per esempio nel riquadro 3). A ogni riquadro viene dunque assegnato inizialmente una pagina da visualizzare, pagina che può essere diversa nel tempo.

I marcatori per i Frame

Quando si vuole definire la struttura a riquadri di una pagina, i marcatori sostituiscono il corpo del contenitore. Ovvero non comparirà il marcatore <BODY> della pagina principale. Il marcatore che definisce le righe o colonne di una struttura è <FRAMESET>, in cui saranno riportati dei parametri che indicano la dimensione delle righe o colonne che compongono la griglia. La dimensione può essere definita in *pixel*, in percentuale, o inserendo il carattere * che indica come quella riga (o colonna) comprende lo spazio rimasto dopo aver impostato le righe o colonne precedentemente definite. Ogni riquadro dovrà definire almeno la pagina da visualizzare attraverso il parametro SRC (*source*).



dettagli di pagine strutturate con i frame

L' esempio seguente definisce due colonne (la prima a dimensioni fisse da 100 *pixel*, la seconda per lo spazio rimanente). La prima colonna contiene inizialmente la pagina index.html e il riquadro è stato chiamato *menu* che rappresenta il nome per intervenire sul riquadro, per esempio per modificare la pagina visualizzata. La seconda colonna è a sua volta suddivisa in due righe (di dimensione pari a 110 la prima, a completamento della pagina la seconda).

Di seguito viene riportato il marcatore <NOFRAMES> utile per impostare dei contenuti che saranno visualizzati dai vecchi *browser* che non riconoscono i *frame*.

```
<FRAMESET COLS = "100,*">
  <FRAME SRC="index.html" NAME="menu">
    <FRAMESET ROWS = "110,*" >
    <FRAME SRC="Logobar.html" NAME="Logobar">
    <FRAME SRC="hp.html" NAME="Main">
  </FRAMESET>
```

```
</FRAMESET>
<NOFRAMES>
<BODY>
.....
```

Indirizzamento di un riquadro

La struttura iniziale di una pagina prevede la visualizzazione di determinate pagine all'interno di ogni riquadro. Tuttavia, l'utente può avere necessità di modificare il contenuto di un settore. Sarà necessario utilizzare un marcatore di collegamento (<A>) indicando la pagina da visualizzare (attraverso il parametro HREF) e lo spazio di destinazione (il parametro è TARGET=nome_del_frame_indirizzato), indicandone il nome definito con il parametro NAME del marcatore FRAME. Per esempio se è stata definita la struttura:

```
<FRAMESET COLS = "85,*">
  <FRAME SRC="menu.html" NAME="menu">
  <FRAMESET ROWS = "110,*" >
    <FRAME SRC="Logobar.html" NAME="Logobar">
    <FRAME SRC="hp.html" NAME="Main">
  </FRAMESET>
</FRAMESET>
```

Attraverso il parametro *TARGET* inserito nei *link* è possibile definire il riquadro di destinazione della pagina da visualizzare.

```
TARGET = nome | _blank | _self | _parent | _top
<A HREF="nuova.html" TARGET="Main">Nuovo caricamento</A>
```

- **blank**: apre un'altra sessione del *browser*, visualizzando la pagina a schermo intero;
- **self**: la pagina da visualizzare appare nello stesso riquadro da cui è partita la richiesta;
- **parent**: serve a indirizzare il documento nel *frameset* da cui ha avuto origine quel documento.Se il documento non ha un *frameset* di origine, viene caricato nella stessa area da cui parte la richiesta;
- **top**: il documento viene ricaricato a pieno schermo, che diventa un'unica area.

Da notare che l'uso dei *frame* può comportare anche dei svantaggi:

- alcuni vecchi *browser* non consentono la visualizzazione dei *frame*; risulta necessario prevedere la compilazione di siti diversi (o pagine diverse) in funzione della compatibilità dei *frame* ai *browser*.
- Il caricamento di una intera pagina strutturata in *frame* è più lenta, dovendo caricare pagine distinte per ogni riquadro.
- Lo spazio disponibile per il contenuto informativo è ridotto, non essendo presentato a pieno schermo. Le barre di scorrimento possono ulteriormente ridurre tale spazio.
- E' poco agevole la gestione dei riquadri a livello utente (visualizzazione del codice sorgente, aggiunta dei segnalibri).

Tabelle

Le tabelle (*tables*) vengono adoperate nei siti per due ragioni principali:

- la prima, più ovvia, è quella di sistemare le informazioni in una tabella;
- la seconda - meno ovvia - è quella di creare il *layout* della pagina servendosi di tabelle nascoste.

L'uso delle tabelle per dividere la pagina in diverse sezioni è uno strumento estremamente efficace. Quasi tutti i siti più grandi sulla rete si servono di tabelle nascoste per strutturare le pagine. Gli aspetti più importanti della progettazione di una pagina che si possono trattare con l'uso di tabelle sono:

- **la divisione della pagina in sezioni separate**. Una tabella invisibile è una risorsa eccellente per questo scopo.
- **La creazione di menù**. In genere con un colore per la voce principale e un altro per i *link* che seguono nelle righe successive.
- **L'inserimento di form *field* interattivi**. Generalmente un'area grigia contenente un'opzione di ricerca.
- **La creazione di titoli di pagina rapidamente scaricabili**. Una tabella colorata con un testo viene scaricata in un batter d'occhio rispetto ad un *banner* anche semplicissimo.
- **L'allineamento di immagini che sono state tagliate in parti più piccole**.
- **La disposizione del testo in due o più colonne sistemate una accanto all'altra**.

L'importanza delle tabelle nella progettazione non deve essere tuttavia sopravvalutata. Ci sono alcune cose da tenere presenti quando si decide di farne uso. La più importante è che il contenuto della tabella viene mostrato solo quando l'intera tabella è scaricata. Se hai pagine molto lunghe, ti consigliamo di dividerle in più tabelle - in modo che l'utente possa leggerne l'inizio, mentre il resto della pagina viene scaricato.

Le tabelle sono definite con il tag <table>.

Per inserire una tabella sulla tua pagina si devono semplicemente aggiungere questi tag nel punto in cui vuoi metterla.

```
<table>
</table>
```

La tabella qui sopra non può funzionare, perché non ha né righe né colonne.

Formattazione delle tabelle

I tag per inserire righe in una tabella sono <tr> e </tr>, e per le colonne <td> e </td>:

```
<table>
  <tr>
    <td>Prima riga, sinistra.</td>
    <td>Prima riga, destra.</td>
  </tr>
  <tr>
    <td>Seconda riga, sinistra.</td>
```

```
   <td>Seconda riga, destra.</td>
  </tr>
</table>
```

Prima riga, sinistra. Seconda riga, sinistra.	Prima riga, destra. Seconda riga, destra.

Esistono diverse opzioni per personalizzare le tabelle. In particolare è possibile formattare le tre zone in cui è divisa una tabella, oltre alle celle:

- spazio tra le celle;
- spazio (spessore) del bordo;
- spazio interno tra celle e testo;
- l'allineamento, dimensioni e colori per ognuna delle *zone* citate.

Tali attributi vanno utilizzati all'interno delle righe (colonne) a cui si vuole attribuire l'aspetto desiderato. La tabella sottostante riporta all'interno di se stessa gli attributi che la descrivono. Ci sono attributi per il bordo (border="0"), per la larghezza (width ="72%"), per lo spazio tra le celle (cellspacing="1"), per lo spazio tra celle e testo (cellpadding="3") e per le celle (width="50%" bgcolor="#C7E1F7" height="17")

attributo	nome e valore
bordo	border="0"
larghezza	width="72%"
spazio tra le celle	cellspacing="1"
spazio tra testo e cella	cellpadding="3"
in ogni cella	width="50%" bgcolor="#C7E1F7" height="17"

Alcuni attributi (il colore per esempio) vanno definiti all'interno dello spazio di ciascuna cella. Altri invece (bordo) sono globali per la tabella.

Formato dei caratteri

Singole parole o intere frasi possono essere messe in risalto tramite l'uso di speciali Stili. L'**HTML** supporta due tipi di Stili: quello logico e quello fisico. Il formato logico dà uno stile al testo a cui si riferisce in accordo con il suo significato (quello del tag) e può variare da *browser* a *browser*. I due tipi di stile sortiscono lo stesso effetto, ma quello logico sarebbe da preferire.

Stile fisico

stile	tag
Italico	<I>testo</I>
Grassetto (Bold)	<B>testo</B>
Sottolineato	<U>testo</U>

Stile logico

stile	tag
Associato all'italico	<DEN>
Per dare enfasi	<EM>
Per titoli di libri, film, ecc	<CITE>
Per scrivere codice	<CODE>
Per gli *input* da tastiera	<KBD>
Per comunicazioni dal *computer* all'utente	<SAMP>
associato al grassetto	<STRONG>
Solitamente in Italico.	<VAR>

Sequenze di *Escape* e altri Tag

Qui di seguito vi sono alcune **escape sequence** per inserire caratteri particolari (quali < &, eccetera) che altrimenti verrebbero considerati come facenti parte di **Tag**: **&**seguito, senza spazi, da:

- lt; per <.
- gt; per >.
- amp; per &.
- quot; per ".
- egrave; per è.
- Egrave; per È.

Attenzione: le sequenze di *escape* sono case sensitive cioè vi è differenza nell'usare un carattere in minuscolo dall'usarlo in maiuscolo.

Installare e configurare un motore di ricerca per un sito Web

Quando un sito *Web* contiene diverse pagine diventa necessaria la presenza di un motore di ricerca. Individuare il giusto motore di ricerca non è cosa banale. Nel passato le ricerche erano fatte mediante indici non strutturati che erano generati automaticamente dalla strutturazione del *file system*. Negli ultimi anni i motori di ricerca hanno generato indici basandosi sui collegamenti ipertestuali nelle pagine *Web*. Questo approccio ha introdotto il problema di controllare le pagine a cui accedono i motori di ricerca.

Le principali caratteristiche che deve possedere un motore di ricerca sono:

- indiciare velocemente un grande numero di documenti in formato differente, tra cui testo, **HTML** e **XML**; per altri tipi di *file*, come PDF, gzip, o *postscript*, usare filtri;
- utilizzre *Web spider* per indiciare documeti remoti su HTTP;
- consentire di memorizzare nel *file* di indice anche le proprietà dei documenti (di solito definiti come META o elementi **XML**) e ritornare queste proprietà come risultato della ricerca;
- possibilità di ritornare sunti dei documenti trovati;
- usare espressioni regolari per selezionare i documenti;
- possibilità di limitare in maniera semplice le ricerche a parti dei siti *Web*;
- possibilità di memorizzare i risultati per rilevanza o secondo un numero di

proprietà in ordine crescente o decrescente;
- limitare le ricerche a parti di documenti, come certi tag **HTML** (META, TITLE, commenti, etc.) o a elementi **XML**.

Tra i motori di ricerca più importanti si ricordano, google, Yahoo, eXcite, Swish-e, altavista, MSN serach.

Google

In questa sezione analizziamo come installare e configurare *google*. Le corrispondenti operazioni per gli altri motori sono simili.

Google aggiunge nuovi siti all'indice ed aggiorna quelli esistenti dopo ogni scansione della rete. In alternativa, è possibile inviare direttamente l'**URL** attraverso la pagina (http://www.google.it/intl/it/add_url.html). Per segnalare un sito, è necessario inserire l'**URL** completo, compreso il prefisso http://; ad esempio : http://www.google.com/. È possibile anche aggiungere commenti o parole chiave che descrivano il contenuto della pagina. Per evitare rimozioni ingiustificate di pagine dall'indice, *Google* può escludere le pagine di un sito dai propri indici **solo su richiesta del *Webmaster* responsabile del sito**.

Rimozione di un sito

È possibile escludere un sito *Web* o una parte di *server* (*directory*) dall'indicizzazione di *Google*. A tal fine è necessario installare nella *directory* principale del *server* un *file* denominato **robots.txt**. Ad esempio, le seguenti istruzioni in robots.txt

User-Agent:*
Disallow: /

evitano che *Google* e altri motori di ricerca possano effettuare operazioni di scansione sul sito. Per ulteriori informazioni su robots.txt, consultare il sito *Web* http://www.robotstxt.org/wc/norobots.html.

Rimozione di singole pagine

Per impedire a tutti i *file* robots di indicizzare singole pagine del sito, inserire il seguente metatag nel codice della pagina **HTML**:

<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">

Per escludere le pagine solo dall'indicizzazione di *Google* e permetterne l'indicizzazione da parte di altri motori di ricerca, utilizzare il seguente tag:

<META NAME="GOOGLEBOT" CONTENT="NOINDEX, NOFOLLOW">

Rimozione di una sintesi

Per sintesi si intende il testo relativo a un risultato della ricerca in cui tutti i termini ricercati sono evidenziati in grassetto. Le sintesi permettono all'utente di esaminare il contesto in cui sono utilizzati i termini prima di selezionare la pagina. Se per una pagina è disponibile una sintesi, l'utente sarà più propenso a selezionarla.

Per evitare che *Google* visualizzi una sintesi per ciascuna delle pagine, utilizzare il seguente tag:

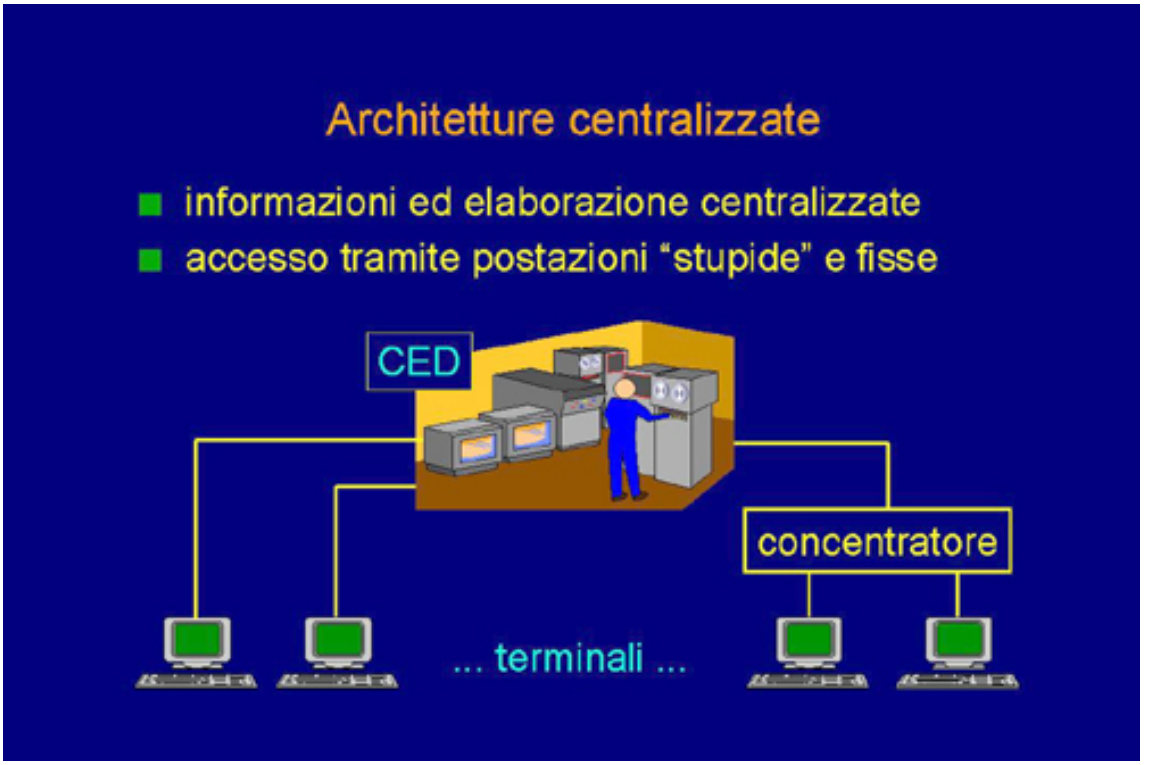
<META NAME="GOOGLEBOT" CONTENT="NOSNIPPET">

Architetture dei Sistemi Informativi Distribuiti

Cosimo Laneve

13.3.9. (Progettare e creare un sito Web)

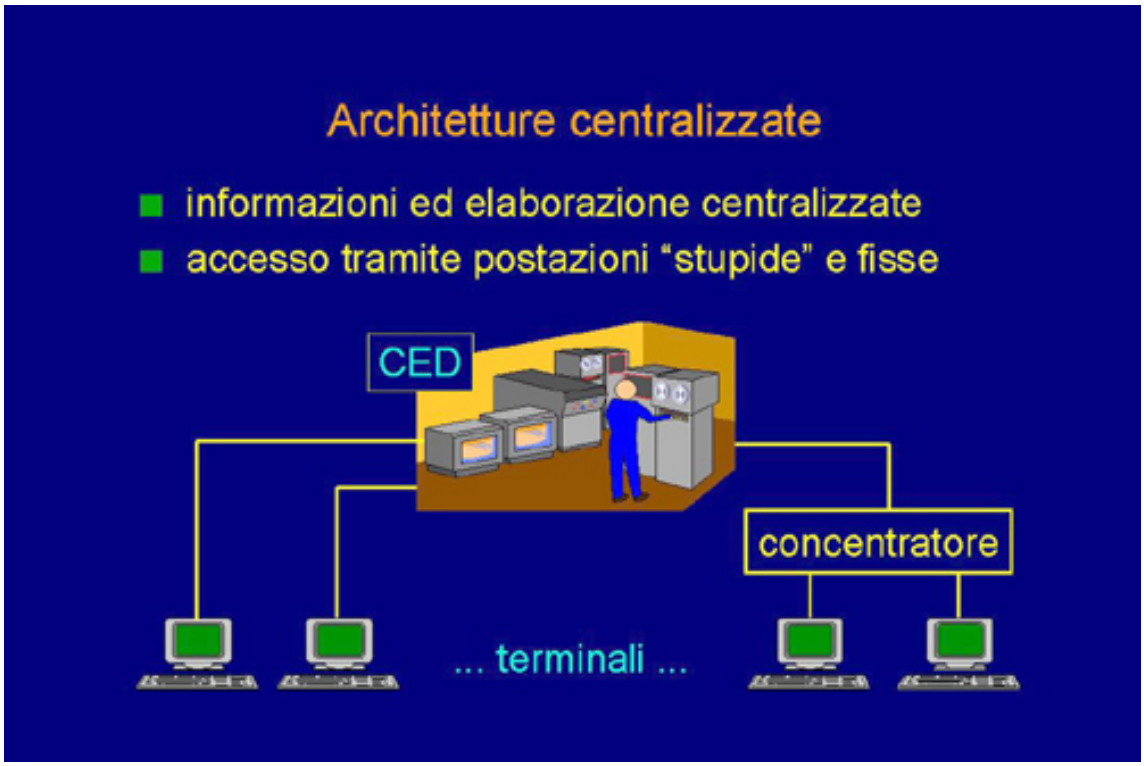
Architetture centralizzate (1/2)



Architetture centralizzate (1/2)

Con questa lezione iniziamo la trattazione relativa alle architetture dei sistemi informativi di tipo distribuito: questi sono i sistemi informativi che oggi vengono realizzati nella maggior parte dei casi, ma per capirne appieno le potenzialità, le funzionalità e l'architettura, conviene prima fare un passo indietro ed esaminare quelli che erano i sistemi informativi di tipo centralizzato che regnavano sovrani fino a circa dieci anni fa e che ancora oggi in molti casi esistono e vengono utilizzati. In un'architettura centralizzata ritroviamo dei calcolatori e delle unità di memoria che sono tipicamente concentrati in un'unica stanza, normalmente denominata CED (Centro Elaborazione Dati) oppure SED (Servizio Elaborazione Dati); quindi in un sistema centralizzato tutta la potenza di calcolo e tutta la potenza di memoria sono memorizzate fisicamente in un unico luogo, chiuso, accessibile soltanto agli operatori. Gli utenti possono interagire con il sistema solo ed esclusivamente attraverso i terminali, questi ultimi sono a tutti gli effetti delle postazioni di lavoro fisse e, per così dire, stupide, nel senso che permettono solo ed esclusivamente di fare delle operazioni di *input-output*, dato che un classico terminale non è dotato altro che di una tastiera ed un video che permettono di fornire dati al sistema di elaborazione collocato all'interno del CED e di prendere le risposte e visualizzarle sul video della postazione di lavoro.

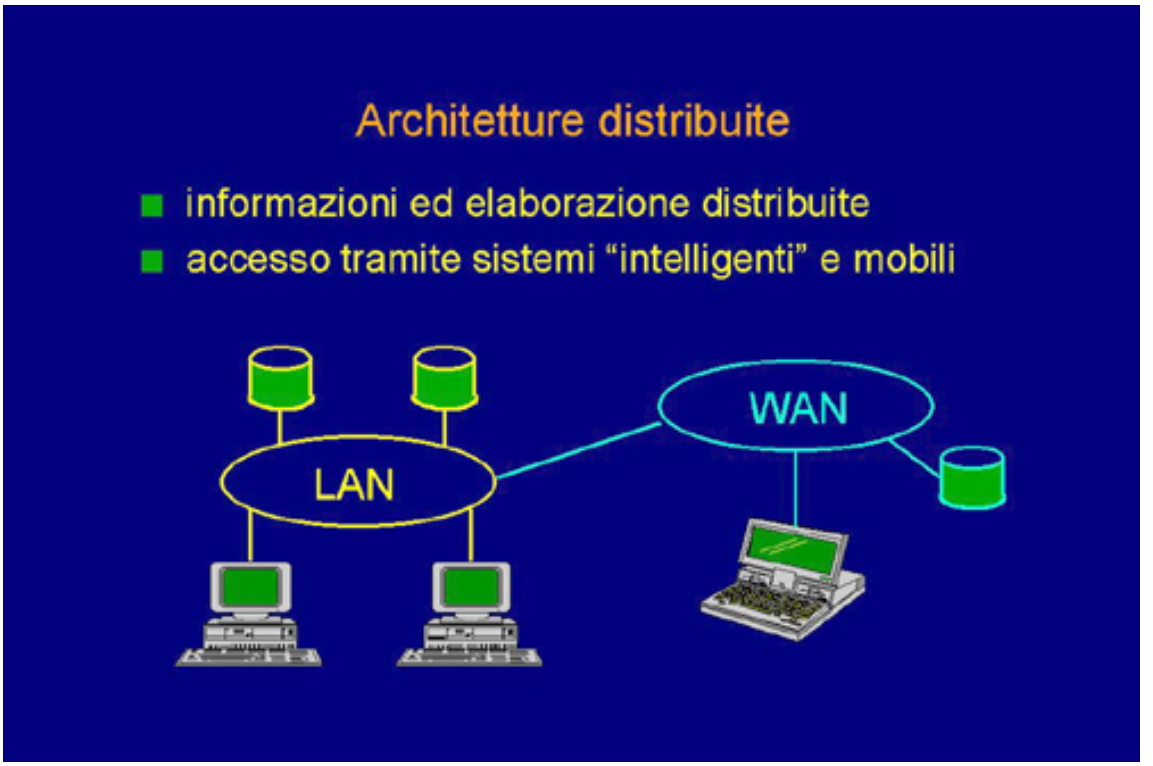
Architetture centralizzate (2/2)



Architetture centralizzate (2/2)

Un terminale quindi non possiede né una sua CPU né una sua capacità autonoma di memorizzare dei dati. I terminali vengono normalmente collegati ai sistemi di elaborazione tramite delle linee dedicate, nel senso di linee seriali, cavi coassiali, che corrono dalla postazione di lavoro dell'utente fino al tipico scantinato, in cui erano posizionati i grossi sistemi di calcolo. Nel caso che la distanza fra il terminale e il nodo di elaborazione sia eccessiva, tipicamente i terminali non venivano collegati direttamente al nodo di elaborazione, ma, come mostrato qui in figura, venivano collegati prima ad un concentratore, cioè un'apparecchiatura di telecomunicazione che si faceva carico di concentrare più linee e di convogliare il flusso di *input-output* su un'unica linea dedicata, ad esempio un collegamento via modem fra una sede remota e il nodo di elaborazione. Notate quindi che queste postazioni sono stupide e fisse e notate altrettanto bene che non è la stessa cosa dire che al posto di un vero e proprio terminale si usa un *personal computer* in modalità emulazione di terminale, perché il *personal computer* è invece dotato di una sua autonoma capacità di elaborazione e di una sua autonoma capacità di memorizzazione dei dati. L'avvento dei *personal computer* e delle reti sono i due elementi base che hanno portato alla morte dei vecchi sistemi informativi centralizzati e alla nascita e all'avvento dei sistemi informativi di tipo distribuito.

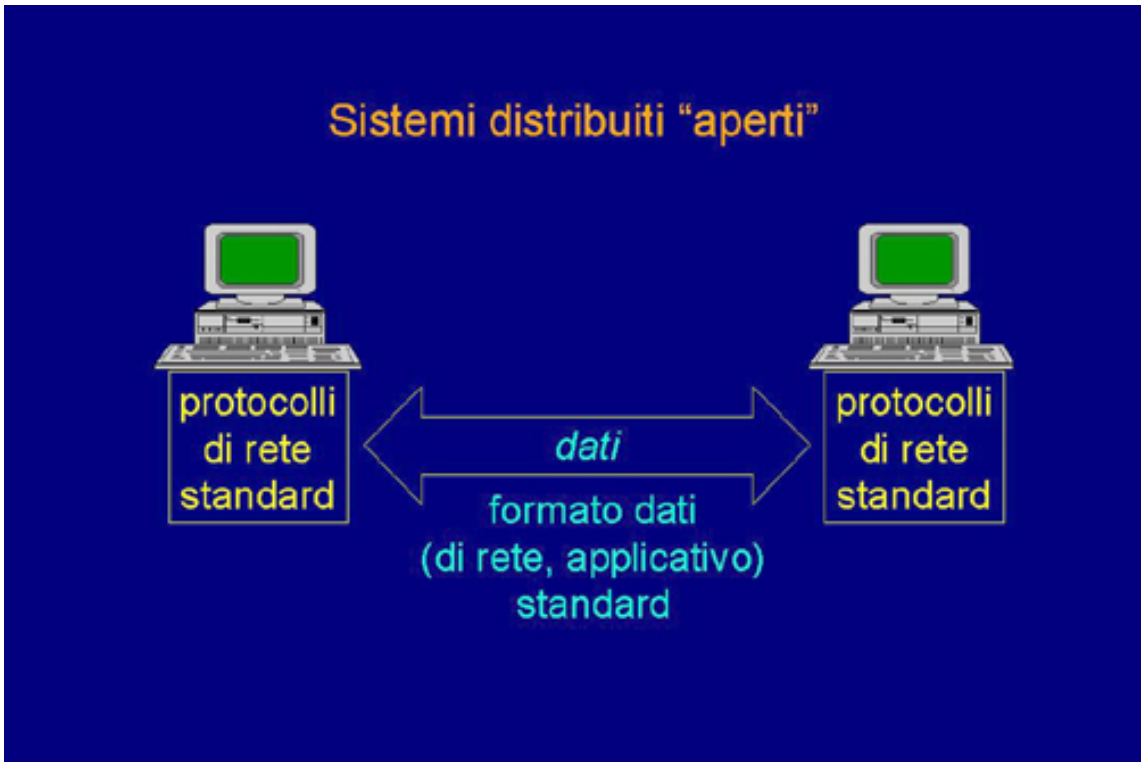
Architetture distribuite



Architetture distribuite

Un sistema informativo di tipo distribuito può essere rappresentato secondo lo schema che vedete qui indicato: le postazioni di lavoro sono diventate dei nodi autonomi di elaborazione: sono i *personal computer*, sia di tipo fisso, cioè quelli che troviamo sulle scrivanie dei nostri tavoli di lavoro, ma anche di tipo mobile, cioè quelli usati dalle persone che per vari motivi devono viaggiare molto e quindi hanno necessità di portarsi appresso la propria capacità di elaborazione. Quindi, pur continuando ad esistere dei *server*, che sono attaccati alla rete locale o alla rete geografica, non è più vero che i *server* sono gli unici depositari della potenza di calcolo e della capacità di memorizzazione. È ancora vero che c'è della capacità di memorizzazione, ma questa è tipicamente distribuita. In sostanza, il grosso cambiamento sta nel fatto che sia la capacità di elaborazione che la capacità di memoria non sono più concentrate in un unico luogo, ma distribuite; inoltre possono essere mobili e possono essere distribuite tra i responsabili del servizio, cioè quelli che gestiscono il successore del CED, il *server* di elaborazione, ma distribuite in parte anche tra gli utenti stessi che possono autonomamente svolgere delle operazioni e memorizzare dei dati. Cerchiamo quindi di capire quali sono gli elementi base da un punto di vista *software* che caratterizzano un'architettura di tipo distribuito.

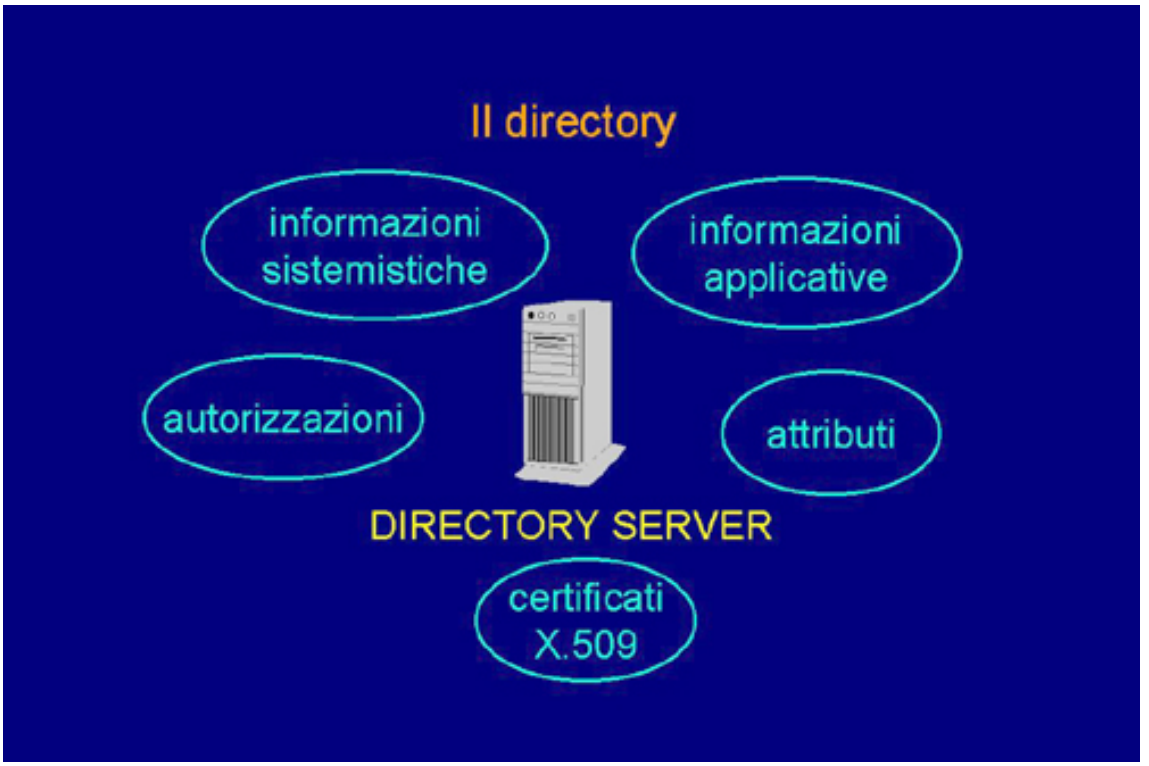
Sistemi distribuiti aperti



Sistemi distribuiti aperti

Innanzitutto affinché si possa parlare di un sistema distribuito di tipo aperto occorre che i nodi di elaborazione che desiderano comunicare, debbano essere dotati di protocolli di rete standard; notate che per comunicare devono essere dotati di protocolli di rete, in particolare poi per comunicare in modo aperto, ossia senza restrizione su quali tipi di nodi possono comunicare, bisogna implementare degli standard aperti, quali ad esempio gli standard della rete *TCP/IP*. Se i due nodi di elaborazione implementano questi protocolli di rete standard, ecco allora che possono scambiare dei dati. È altrettanto importante che anche il formato di questi dati, sia a livello di pacchetto di rete che di tipo applicativo, sia altrettanto standard, perché questa è l'unica garanzia che abbiamo di realizzare delle applicazioni che non siano vincolate ad un unico fornitore, ad un'unica particolare architettura *hardware* o *software*. Quindi la parola standard è da intendersi sia a livello di protocolli di telecomunicazione, sia a livello di formato dei dati grezzi o del formato dei dati applicativi.

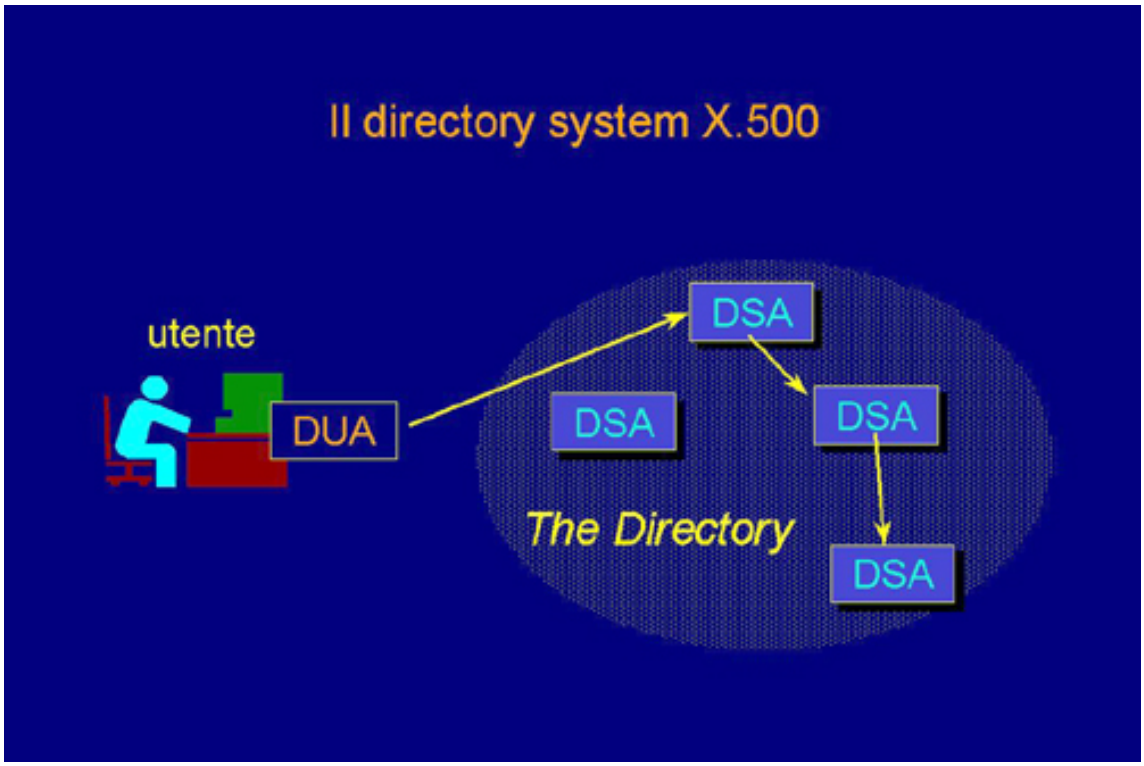
Il directory



Il directory

In particolare, oggi come oggi, uno degli elementi che si stanno imponendo all'attenzione di tutti i progettisti di sistemi distribuiti aperti per la sua capacità di creare, di memorizzare informazioni e distribuirle all'interno della rete è il cosiddetto *directory*. Il *directory* viene normalmente implementato tramite un *server* ad esso dedicato che mantiene una serie di informazioni e le fornisce a tutti i nodi della rete. Questo ci permette di avere in un unico posto le informazioni che servono e far sì che queste vengano distribuite a tutti i nodi; in particolare il *directory* è emerso come un elemento essenziale quando si è cominciato a pensare di voler fare sicurezza di un sistema distribuito, perché tramite il *directory* noi possiamo fornire i cosiddetti certificati a chiave pubblica X.509 che sono uno degli elementi fondamentali, portanti, di tutte le architetture di sicurezza per sistemi distribuiti. Ma questo è solo uno dei tipi di dati che si possono memorizzare nel *directory*; ad esempio il *directory* può essere utilizzato per memorizzarci informazioni sistemistiche, quindi i nomi degli utenti, le *password*, i privilegi di ciascun utente. Oltre a questo tipo di informazioni sistemistiche, una volta che viene messo in piedi un *directory server*, viene ovviamente voglia di convogliare su di esso più informazioni possibili per semplificarne la gestione e per evitare di avere copie di dati non allineate sui vari sistemi. Quindi oltre alle informazioni sistemistiche si possono inserire anche informazioni di tipo applicativo, ossia che dipendono dalla particolare applicazione che un utente desidera usare. Si possono inserire anche degli attributi non necessariamente correlati al sistema operativo, ma correlati alle funzioni applicative che un utente svolge. Si possono mettere anche le autorizzazioni, che sono tipicamente il tramite tra gli utenti e le applicazioni, ossia un'autorizzazione non è niente altro che la concessione o la negazione del permesso per un certo utente di accedere a determinate risorse di tipo informatico.

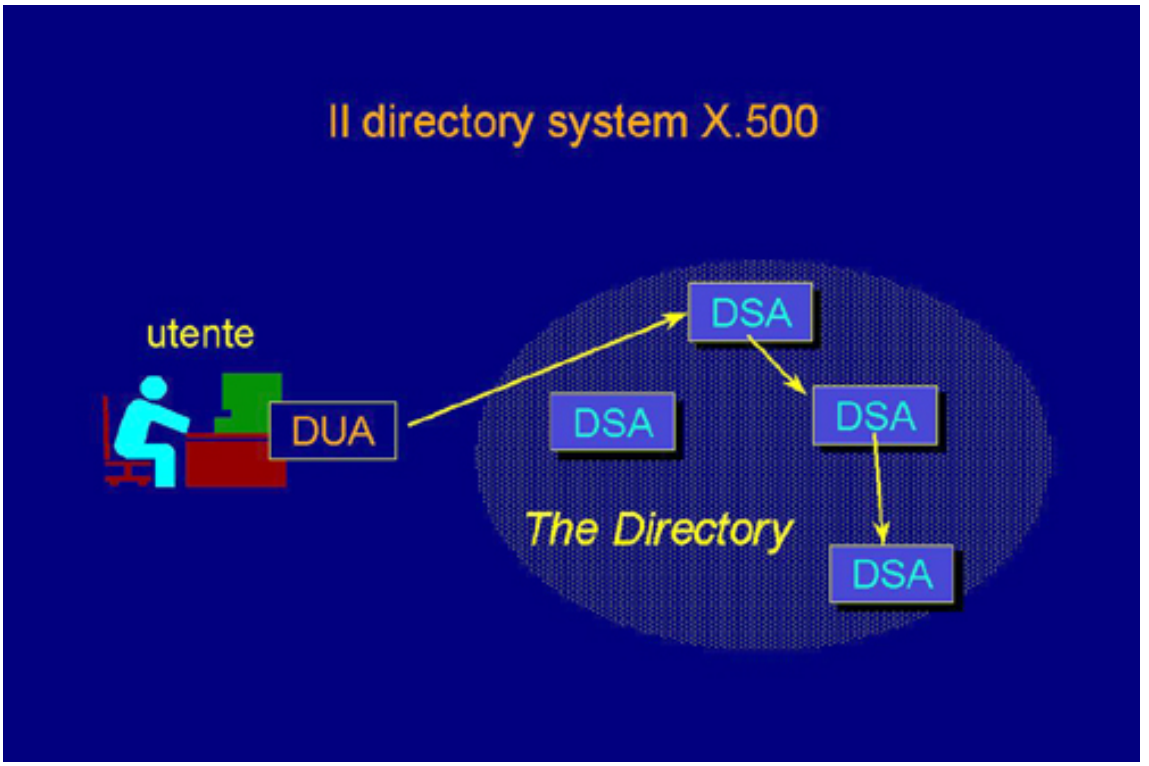
Il directory system X.500 (1/2)



Il directory system X.500 (1/2)

Esistono due modelli base per realizzare un *directory*. Esso è nato molti anni fa e si è imposto all'attenzione informatica con la rete OSI; all'interno di questa rete era stato sviluppato un servizio applicativo chiamato X.500 perché scritto nello standard ITU X.500. Il *directory system* X.500 è schematizzabile nel seguente modo: l'utente che desidera interagire con il *directory system* deve essere dotato di un opportuno *client* chiamato DUA (*Directory User Agent*). Tramite il *Directory User Agent* l'utente può interrogare questa grossa nuvola che contiene i dati. Questi ultimi vengono chiamati genericamente *the Directory* e sono gestiti da singoli *server*; nella terminologia X.500 i *server* vengono a sua volta chiamati DSA (*Directory System Agent*). Ogni *Directory System Agent* mantiene una certa quantità di informazioni, ad esempio potremmo pensare che ogni *Directory System Agent* gestisca le informazioni relative ad un certo dipartimento all'interno di un'organizzazione, oppure nel caso di un'organizzazione soprannazionale potremmo pensare che ogni *directory* gestisca le informazioni relative ad un certo paese. Ogni *directory* ha delle informazioni che altri non hanno, quindi nel momento in cui il nostro utente fa riferimento ad un *Directory System Agent* specifico può darsi che trovi le informazioni che cerca, come può darsi che non le trovi. Nel caso in cui l'utente non trovi le informazioni nel primo *Directory System Agent* lo standard X.500 prevede che i vari *Directory System Agent* possano parlarsi tra di loro tramite l'operazione cosiddetta di *chaining*, cioè concatenazione, con questa operazione i vari *Directory System Agent* fanno automaticamente la ricerca richiesta dall'utente e sono in grado di fornirgli la risposta.

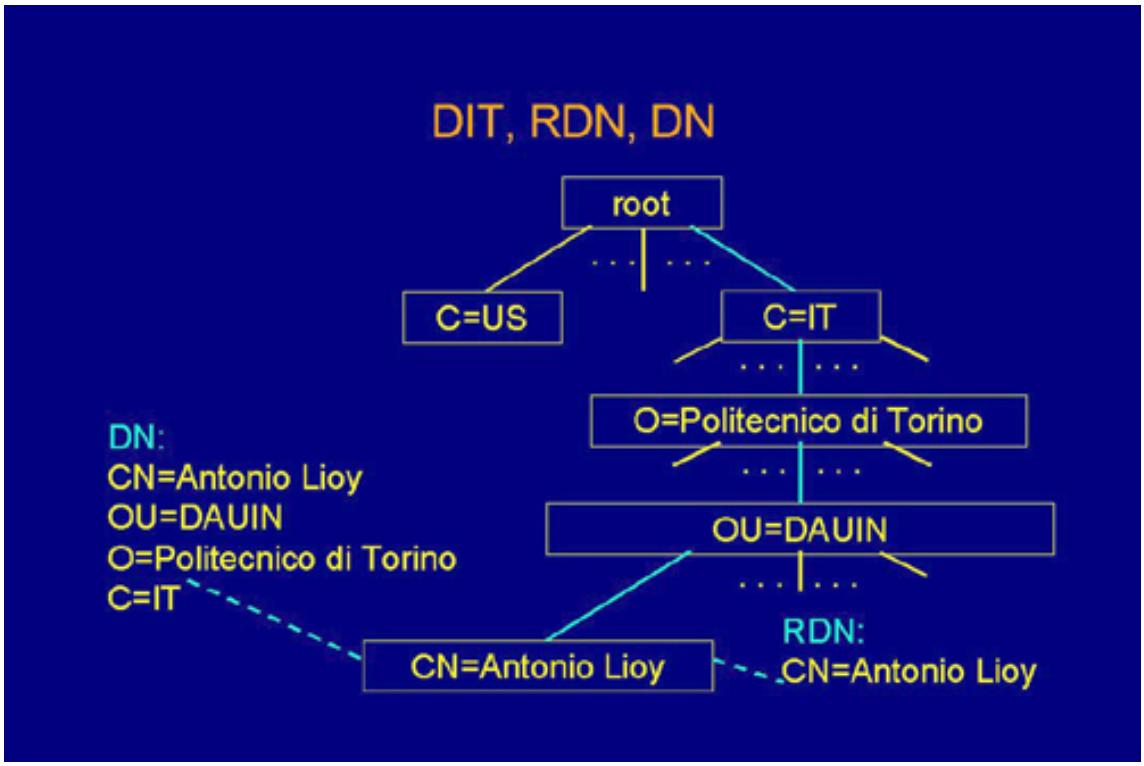
Il directory system X.500 (2/2)



Il directory system X.500 (2/2)

Uno dei più grossi problemi esistenti nel *directory system* X.500 è che il meccanismo di *chaining* è definito a livello di formato, di scambio di dati, ma non è definito a livello organizzativo, ossia la ricerca dell'informazione segue un cammino di amicizia: il gestore di un certo *server* è a conoscenza dell'esistenza di un altro *server* e quindi gli passa le informazioni, ma non esiste un'organizzazione, né a livello nazionale né a livello mondiale, che permetta con certezza di trovare tutte le informazioni esistenti. Il *directory system* X.500 teoricamente sarebbe in grado di creare un *directory* a livello mondiale, in realtà purtroppo è frammentato all'interno di tante piccole realtà. X.500, facendo parte dello standard OSI, ha seguito un po' il fatto di questo standard che è declinato dopo la sua apparizione e gli entusiasmi che inizialmente aveva destato, per la sua complessità di implementazione. Oggigiorno sappiamo tutti quanti che le reti che vanno per la maggiore non sono quelle standard OSI, ma sono quelle standard *TCP/IP*. Nell'ambito di questo *TCP/IP* il *directory* è stato rivisitato, vedremo tra un attimo come.

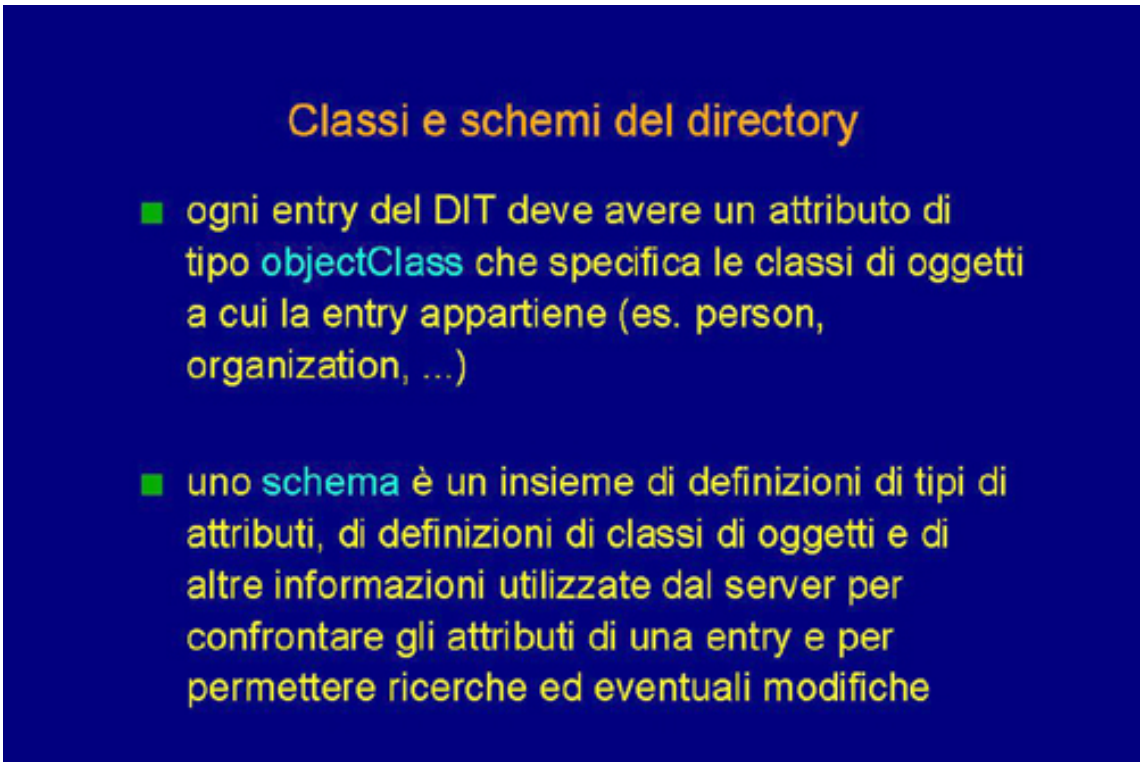
DIT, RDN, DN



DIT, RDN, DN

Prima intendo sottolineare che qualunque sia il tipo di *directory* che noi adottiamo, sia il modello X.500, sia il modello di *TCP/IP*, i dati dentro il *directory* vengono organizzati gerarchicamente, ciò vuol dire che esistono vari nodi, ognuno dei quali corrisponde al cosiddetto *distinguished name* di tipo relativo. Ad esempio, al primo livello i nodi identificano i paesi, indicati con C (*Country*), in questo caso Italia; al secondo livello i nodi identificano le organizzazioni: qui sto seguendo la strada che mi porta al Politecnico di Torino; e, al terzo livello si identificano tipicamente le unità organizzative, in questo caso il mio dipartimento di Automatica ed Informatica. Sotto questo livello esistono ancora degli altri nodi che identificano le persone,o gli oggetti, l'entità di rete, tramite il loro nome comune. La strada che viene seguita a partire dalla radice, dalla base del *directory*, per arrivare all'informazione cercata, quindi la sequenza dei RDN (*Relative Distinguished Name*) costituisce il cosiddetto DN (*Distinguished Name*), il nome distinto, unico di quell'oggetto; in questo caso questa è la sequenza che costituisce il mio *Distinguished Name*.

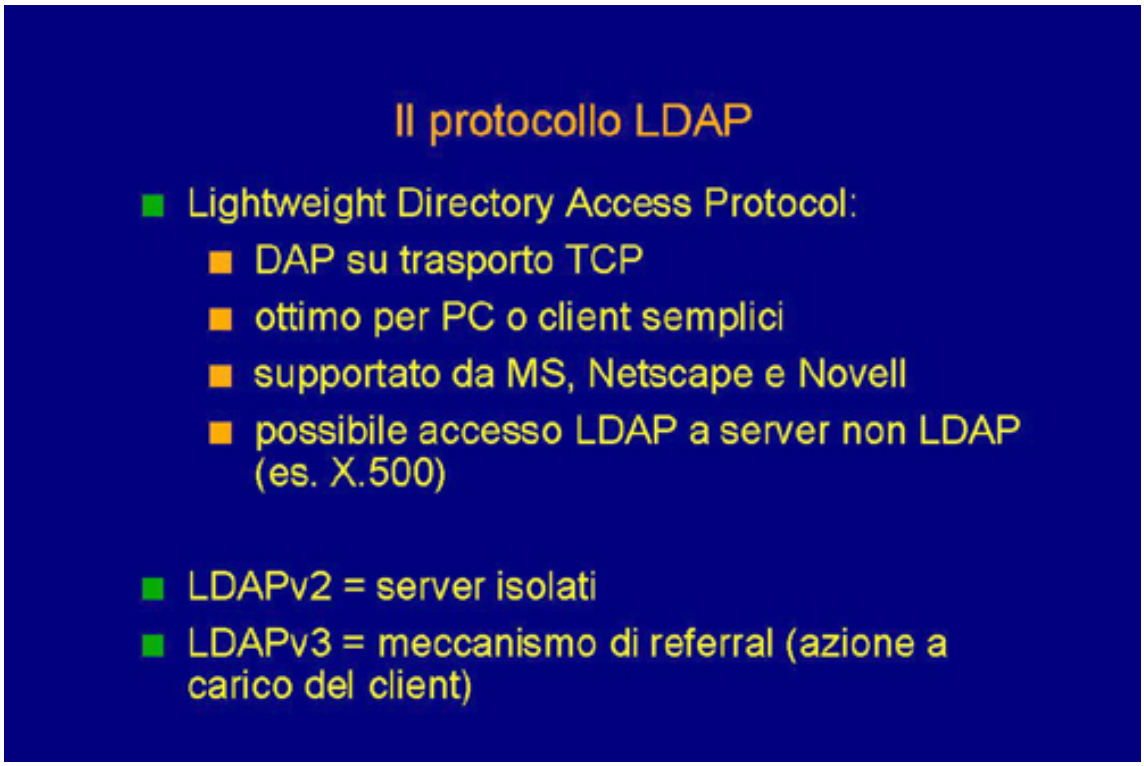
Classi e schemi del directory



Classi e schemi del directory

Per ognuno dei dati che vengono messi in questo albero, detto DIT (*Directory Information Tree*) vengono definiti degli attributi che sono specificati come una classe, ossia ogni *entry* nel *directory* appartiene ad una classe di oggetti per la quale sono stati predefiniti degli attributi, ossia dei dati che lei stessa può avere. Poiché uno stesso *directory* può contenere oggetti di tantissime classi, ecco che è necessario definire quello che viene indicato con schema, cioè l'insieme di tutti quanti i dati che possono risiedere nel *directory*; vedete nella *slide* che ho indicato lo schema come un insieme di tipi, attributi, classi ed altre informazioni genericamente utilizzate dal *server* per effettuare ricerche o modifiche dei dati memorizzati. Come vi dicevo, all'interno delle reti *TCP/IP* non si è seguito lo standard X.500 puro, ma si è seguita una strada X.500 modificata, semplificata, per renderla effettivamente applicabile.

Il protocollo LDAP



Il protocollo LDAP

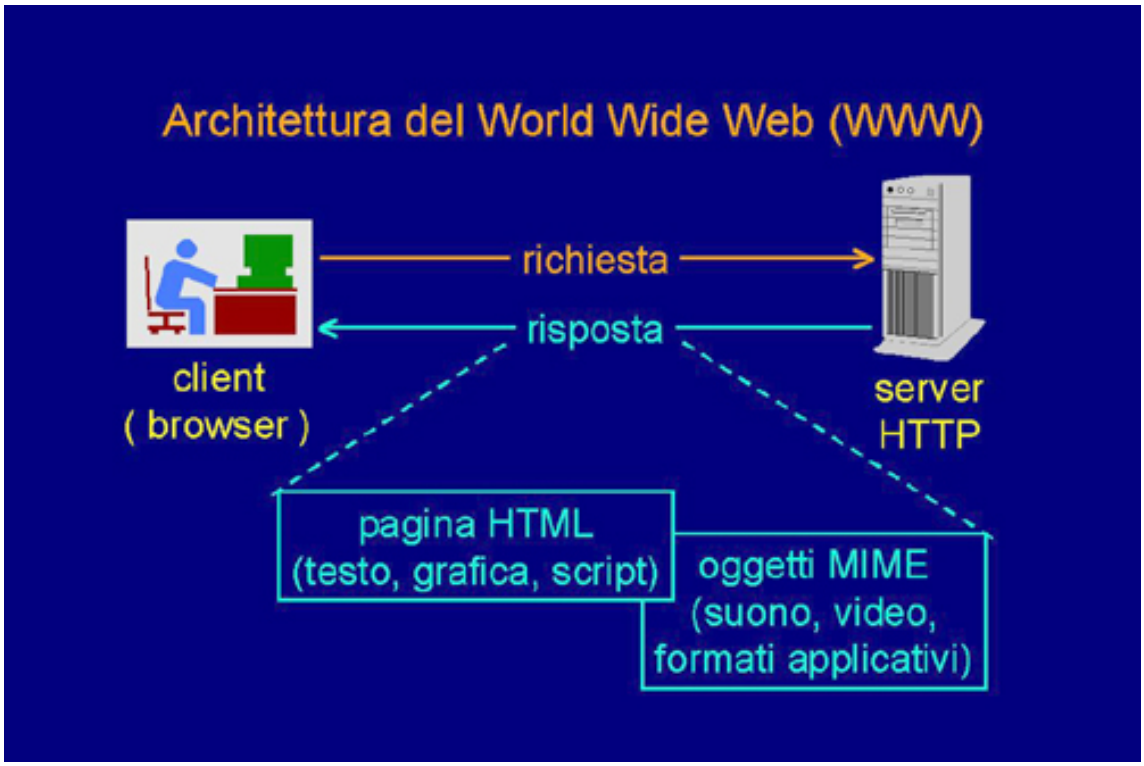
Questa strada ha portato alla definizione di un nuovo protocollo per accedere ai dati, chiamato LDAP ed è un protocollo standard nelle reti *TCP/IP* per accedere dal *client* ai *server* di *directory*. LDAP significa protocollo leggero per l'accesso al *directory*, ossia non è altro che il protocollo DAP, originariamente usato da X.500, portato su un trasporto, su un canale di tipo *TCP*. È bastata questa semplice sostituzione per rendere questo protocollo ottimo per applicazioni anche su *client* molto semplici quali *personal computer* o addirittura dispositivi portatili. Il grande successo di LDAP è stato sancito da una dichiarazione pubblica del Settembre 1997 in cui *Microsoft* e *Netscape* hanno dichiarato che era loro intenzione convergere sul sistema LDAP come sistema per distribuire le informazioni all'interno delle loro applicazioni. Notate che per utilizzare LDAP non è indispensabile avere un *server* LDAP, esistono infatti anche dei *server* X.500 che sono in grado di parlare entrambi i protocolli, ossia sono in grado di parlare DAP verso i *Directory User Agent* e di parlare LDAP verso i *client* più semplici. Di LDAP esistono tre versioni, a parte la prima versione che ha ormai un interesse più storico, oggigiorno le due versioni più utilizzate sono LDAPv2 e LDAPv3. La differenza fondamentale tra queste due implementazioni consiste nel fatto che in LDAPv2 ogni *server* di *directory* è un *server* completamente slegato da tutti gli altri; quindi se noi poniamo la nostra domanda ad un *server* LDAPv2 e questo non conosce la risposta, sarà compito nostro andarci a cercare in giro per il mondo un altro *server* che sia in grado di soddisfare la nostra domanda. In LDAPv3 invece è possibile che il *server* a cui abbiamo fatto la domanda sappia di non avere la risposta, ma nel contempo sappia indicarci lui stesso, tramite il meccanismo detto di *referral*, un altro *server* che potrebbe avere le informazioni che noi cerchiamo. In tal caso il *server* LDAPv3 si limita a fornirci un riferimento, ma l'azione di andare a chiedere la nuova informazione sarà completamente a carico del *client*.

Uso di LDAP da parte delle applicazioni

Uso di LDAP da parte delle applicazioni

LDAP oggi giorno è largamente utilizzato in tantissime applicazioni; in particolare si trovano dei *client* LDAP all'interno di tutti i sistemi più moderni di posta elettronica; in questo modo è possibile realizzare una rubrica globale al posto di una rubrica personale, ossia un elenco di nomi, cognomi, organizzazioni e indirizzi di posta elettronica comuni a tutte le persone che utilizzano quel *directory*; è anche possibile mettere all'interno del medesimo *directory* certificati a chiave pubblica X.509 che servono per fare sicurezza della posta elettronica. I *server* LDAP vengono utilizzati anche per fare funzioni di autenticazione e di autorizzazione: ad esempio ci sono molte apparecchiature di tipo NAS, ossia i *server* che fanno da *front-end* ai modem, quando da casa ci si collega ad *Internet*, che chiedono se si è autorizzati ad entrare ad un LDAP *server*. Questo ad esempio è estesamente utilizzato da tutti quei fornitori che forniscono servizi internet gratuiti, perché permette una fascia di integrazione anche con la gestione delle caselle di posta elettronica e delle *home directory* sul *Web*. Un'ultima parola riguardo il nuovo sistema operativo di *Microsoft*, *Windows* 2000, il quale ha rivoluzionato la propria architettura interna adottando un *directory* unico di tipo LDAP, dentro al quale va a memorizzare informazioni che prima erano suddivise fra il cosiddetto *registry*, tra vari *file* di sistema e include anche le informazioni che classicamente sono memorizzate nel DNS, ossia il sistema che permette di tradurre da nomi ad indirizzi. Quindi LDAP si sta decisamente affermando come il sistema per distribuire informazioni all'interno di un sistema di rete.

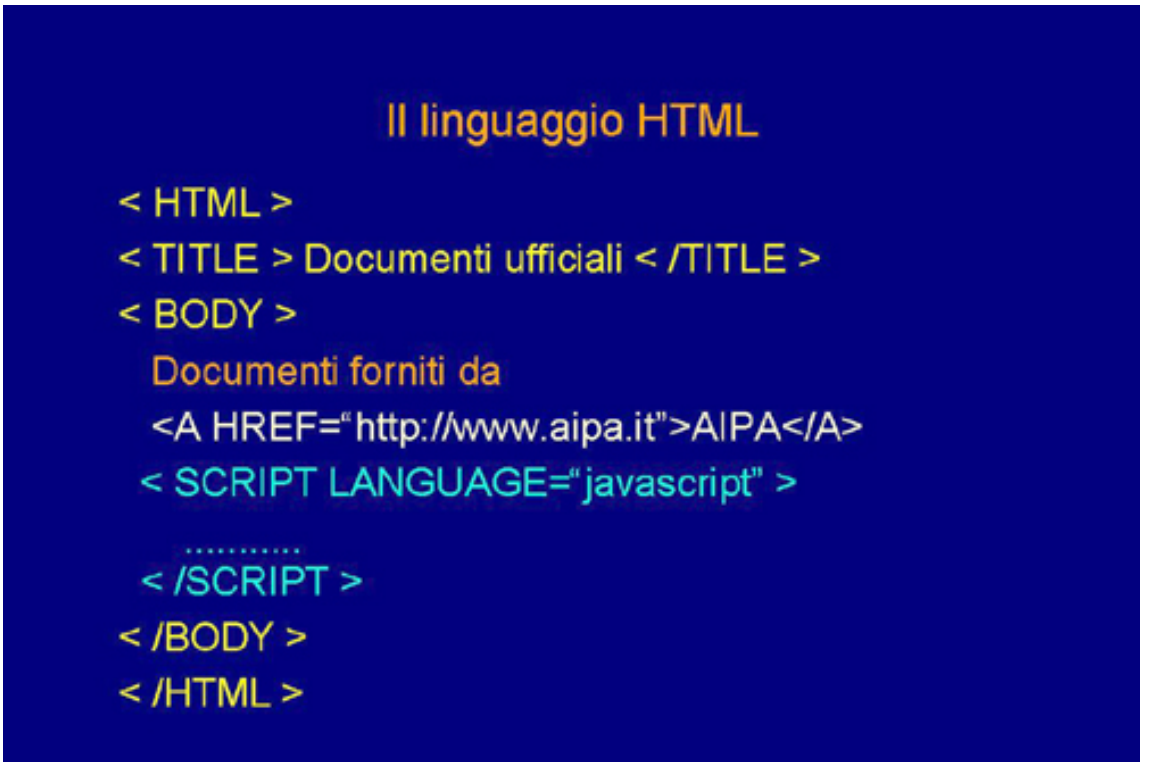
Architettura del World Wide Web



Architettura del World Wide Web

Se da una parte LDAP è uno degli elementi principali di un sistema distribuito, dall'altra parte a livello applicativo la tecnologia più utilizzata è quella del *World Wide Web* (WWW). Vediamo quindi un attimo quali sono i presupposti tecnici del *World Wide Web*. Il WWW si basa sull'esistenza di *client* chiamati *browser*, e di *server*; in particolare i *client* e i *server* devono utilizzare per parlarsi il protocollo HTTP. Tale protocollo prevede che il *browser* invii una richiesta verso il *server*, ad esempio la richiesta di visualizzare una certa pagina. Il *server* a questo punto fornisce la risposta alla domanda del *client* e tale risposta può contenere varie cose; di base contiene una pagina scritta in linguaggio HTML, che è un linguaggio ipertestuale ed ipermediale utilizzato dal *Web*. In questo modo nella pagina è possibile mettere del testo, della grafica ed anche degli *script*, ossia dei piccoli programmi che mi permettono di fare delle azioni un po' particolari. È anche possibile introdurre all'interno della pagina HTML degli oggetti cosiddetti *MIME*, cioè oggetti multimediali, come ad esempio del suono, del video o dei formati applicativi. Bisogna però sottolineare che questi oggetti *MIME* non fanno parte dei dati supportati in modo nativo dai *client* e dai *server Web*, ma devono essere trattati tramite i cosiddetti *plug-in*, ossia del *software* aggiuntivo da installare sul nostro *browser*.

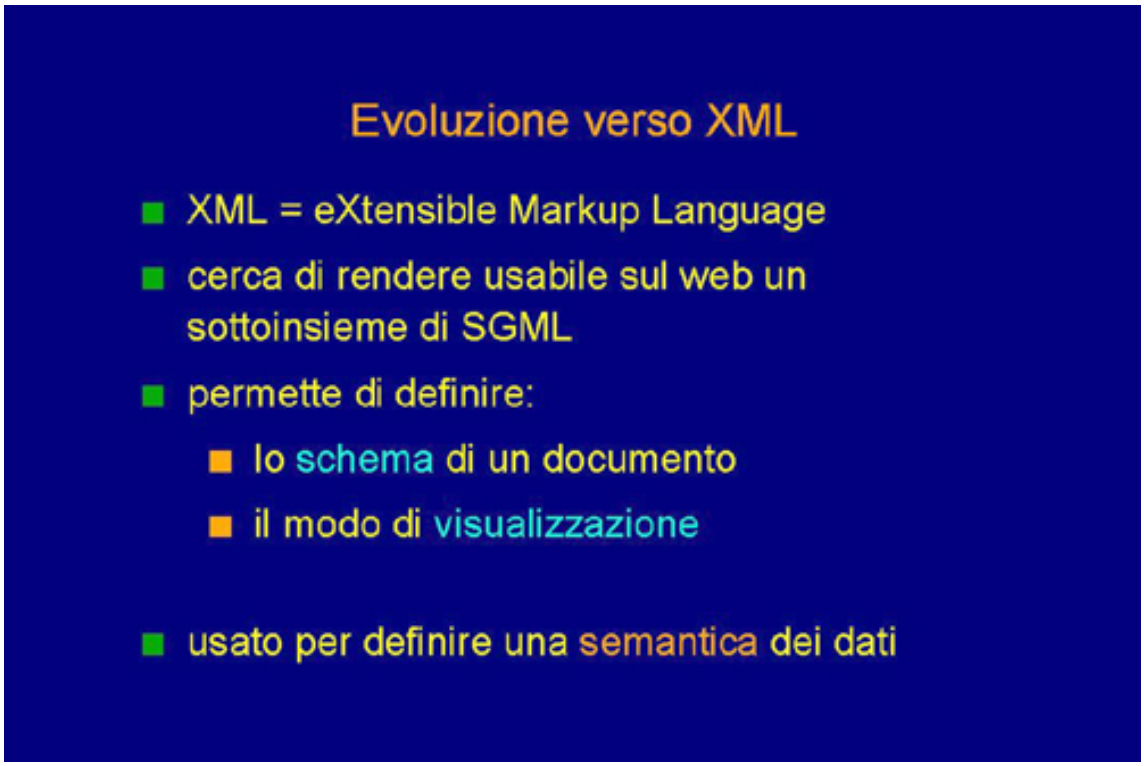
Il linguaggio HTML



Il linguaggio HTML

Ho detto che i dati devono essere espressi secondo il formato HTML, che vediamo indicato in questa *slide* di esempio; l'HTML è un linguaggio basato su *tag*, ossia su delle dichiarazioni, questa, <HTML>, ad esempio dice che qui inizia il *file* HTML, qui invece con <*TITLE*> inizia il titolo del documento e qui finisce, dove troviamo </*TITLE*>.In mezzo alle due troviamo il titolo vero e proprio. Al centro troviamo invece il corpo del documento, tra <*BODY*> e </*BODY*> che contiene non solo del testo, ma anche un link, ossia questa parola, AIPA, se premuta con il *mouse* mi porterà automaticamente verso questo sito *Web*, e contiene anche uno *script*, cioè un mini-programma scritto in un particolare linguaggio di programmazione, ad esempio *Javascript*, e qui ci saranno le informazioni relative a questo *script*. Quindi il linguaggio HTML mi permette di inserire all'interno del medesimo documento vari tipi di informazioni.

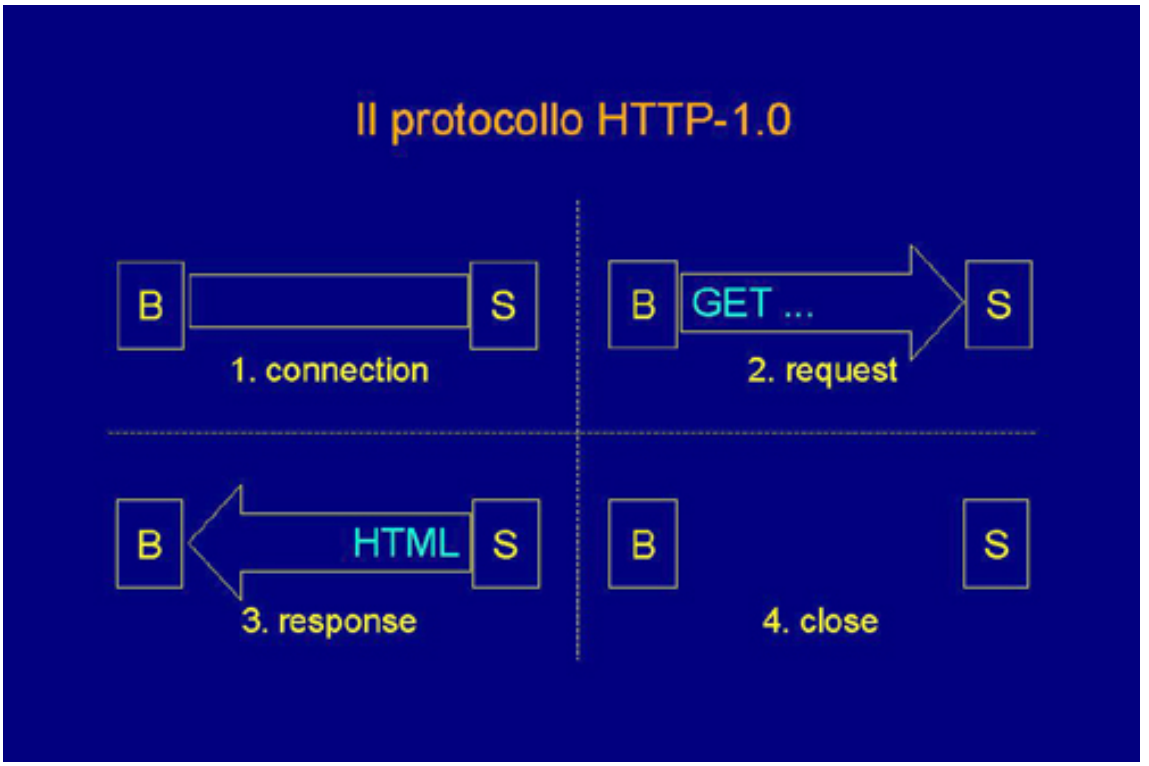
Evoluzione verso XML



Evoluzione verso XML

HTML è stato il linguaggio che ha permesso la nascita del *Web*, ma ha un certo insieme di restrizioni: permette di formattare i documenti per la visualizzazione, permette di formattarli un po' meno bene per la stampa, ma soprattutto non permette di attribuire una semantica ai dati che noi abbiamo inserito. A questo scopo è stato sviluppato un nuovo linguaggio che oggi sta destando molto interesse, chiamato XML, ed è una estensione dell'HTML, infatti il nome stesso XML significa *eXtensible Markup Language*. Tale linguaggio è stato sviluppato per cercare di rendere usabile, tramite il *Web*, un sottoinsieme di un linguaggio ancora più generale chiamato SGML, quest'ultimo è infatti il progenitore sia dell'HTML che del XML. In generale XML permette di definire non soltanto i dati che costituiscono il documento, ma lo schema a cui questi dati ubbidiscono e anche il modo in cui i dati devono essere visualizzati, quindi chi realizza il documento ha più controllo sul modo in cui deve essere visualizzato; ma soprattutto tramite XML è possibile definire una semantica dei dati, ossia è possibile non limitarsi semplicemente a dire che questo è un titolo, ma è possibile dire che questo è un titolo che rappresenta ad esempio il nome e cognome della persona fisica che ha realizzato questa pagina, quindi attribuire un significato alle parole e ai dati presenti dentro un *file* XML. Se questo da una parte arricchisce moltissimo il linguaggio, dall'altra parte presenta potenzialmente grandissime incompatibilità, perché ognuno di noi sarà in grado di definire un proprio schema di interpretazione dei dati XML, ma questo ovviamente contrasta con la necessità di standardizzare. Quindi XML rappresenta sicuramente un grosso passo in avanti nel rendere più ricco di contenuti informativi il *Web*, ma bisogna fare molta attenzione a come viene implementato perché si corre il rischio di creare delle nicchie che non interoperano in alcun modo tra di loro.

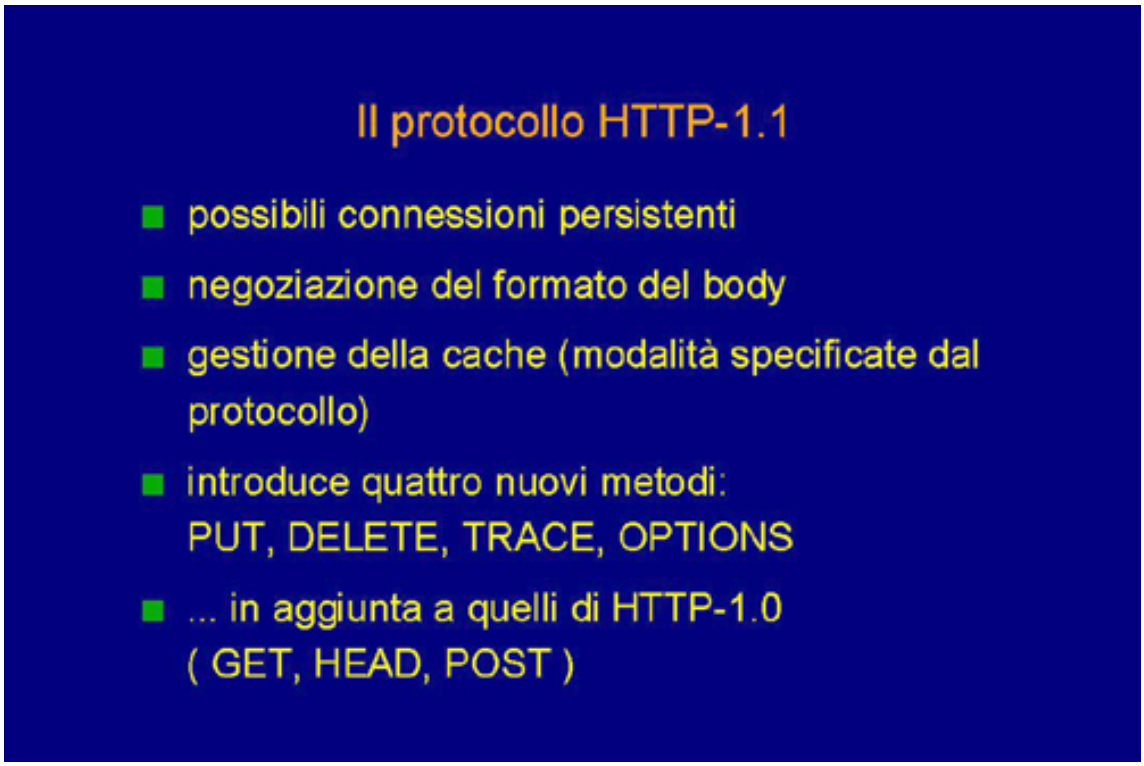
Il protocollo HTTP-1.0



Il protocollo HTTP-1.0

Sia che si tratti di HTML, sia di XML, in entrambi i casi questi dati verranno trasportati tramite il protocollo HTTP; tale protocollo nella versione 1.0 (HTTP-1.0) funziona nel seguente modo: quando il *browser* desidera prelevare dei dati dal *server* istituisce una connessione, tipicamente questo è un canale *TCP* relativo alla porta 80 del *server*. Una volta che il collegamento è stato stabilito, il *browser* manda su questo canale la sua richiesta di informazione, tipicamente questo avviene sottoforma di un comando *GET*, cioè si vuol prendere dei dati. Il *server* riceve questa richiesta di informazioni e fornisce la risposta, anche qui tipicamente tale risposta è sottoforma di *file* HTML , oppure se siamo più moderni XML. Non appena la risposta è arrivata, il *server* chiude il collegamento. Notate che ipoteticamente sarebbe possibile che il *browser* avesse altre domande da fare nei confronti del *server*, ma poiché i *server Web* sono tipicamente molto sovraccarichi è stata scelta questa strada: il *browser* può fare soltanto una domanda alla volta per evitare di sovraccaricare il *server* tenendo aperti dei canali che non si sa quando verranno utilizzati. Quindi se un *browser* accede ripetutamente ad un *server Web*, con il protocollo HTTP-1.0 è obbligato ogni volta ad aprire e chiudere i canali effettuando su ogni canale una sola richiesta per volta. Il protocollo HTTP-1.0 è oggi evoluto verso la versione HTTP-1.1.

Il protocollo HTTP-1.1



Il protocollo HTTP-1.1

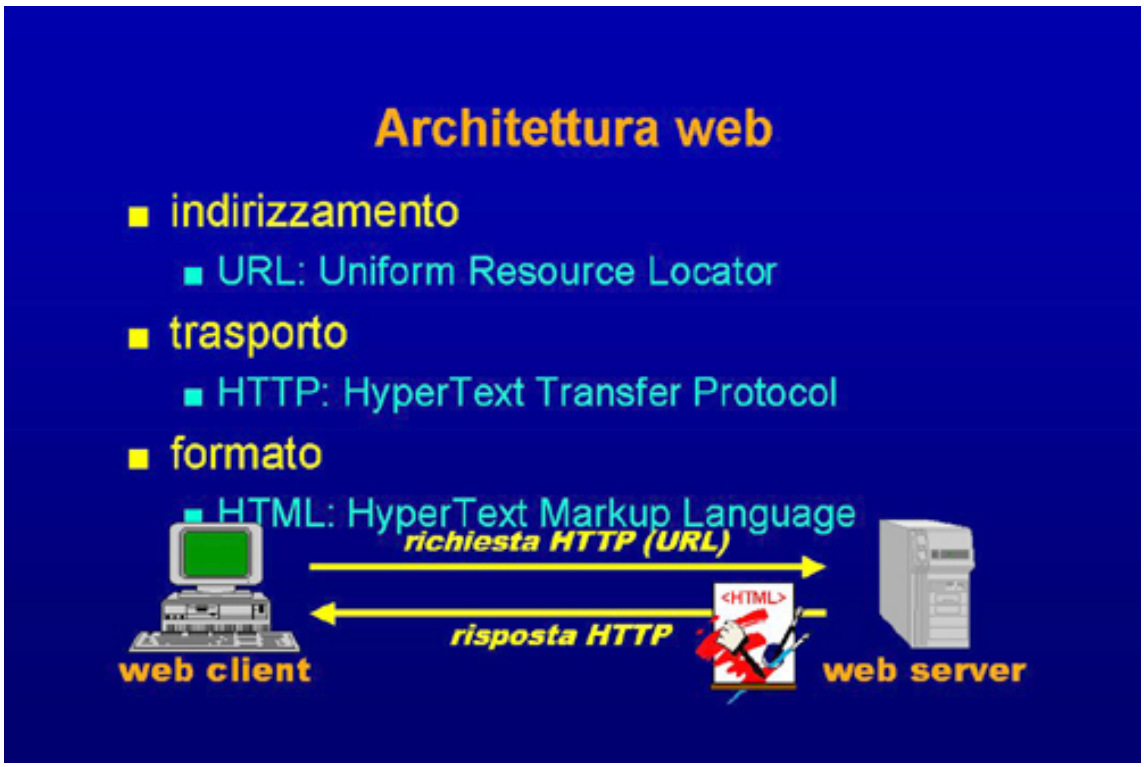
Il protocollo HTTP-1.1 protocollo permette, sotto il controllo del sistemista che gestisce il *server*, di creare connessioni persistenti, ossia permette al *browser* di creare un canale e di mantenerlo aperto facendo più richieste sul medesimo canale; questo è vantaggioso perché allevia il lavoro da parte del *client* e stressa di meno la rete, ma è pericoloso perché potenzialmente porta un grosso sovraccarico da parte del *server*. Nella versione 1.1 è anche possibile, per *browser* e *server*, negoziare il formato dei dati, ossia del *body* delle informazioni che vengono scambiate, in questo modo se il *server* ha a disposizione i dati in più formati diversi, è possibile fornire al *client* i dati nel formato che lui è meglio in grado di accettare. Un'altra innovazione è stata la gestione della *cache*; voi sapete che la *cache* è uno dei meccanismi con cui i *browser* o i nodi intermedi, cosiddetti *proxy*, possono tener copia locale dei dati, per evitare ogni volta di andarli a prendere dal *server*. In questo caso è possibile specificare all'interno del protocollo, quindi da parte di chi gestisce il *server Web*, qual è la modalità per la gestione della *cache*. Inoltre sono stati introdotti quattro nuovi metodi, cioè quattro nuove operazioni che il *browser* può fare nei confronti del *server*; originariamente le operazioni erano soltanto quelle di prendere un documento, prendere solo l'intestazione di un documento, oppure mandare delle informazioni semplici, tipicamente il risultato di un *input*, al *server*. Nell'HTTP 1.1 sono state aggiunte a queste anche le operazioni chiamate *PUT*, che permettono di mettere un intero documento sul *server*, *DELETE*, che permettono di cancellarlo, *TRACE*, che serve a vedere qual è la sequenza dei *proxy* tra il *browser* e il *server* e *OPTIONS*, che servono a conoscere quali sono le opzioni del canale HTTP presente. In questo modo noi stiamo evolvendo verso un'architettura molto più ricca di possibilità. Quindi l'unione del *directory*, con il *server* di tipo *Web*, con il linguaggio che da HTML sta evolvendo verso XML, sono oggigiorno i tre cardini portanti della realizzazione di qualunque sistema informativo distribuito di tipo aperto.

Architetture Web: CGI

Cosimo Laneve

13.3.9. (Progettare e creare un sito Web)

Architettura Web

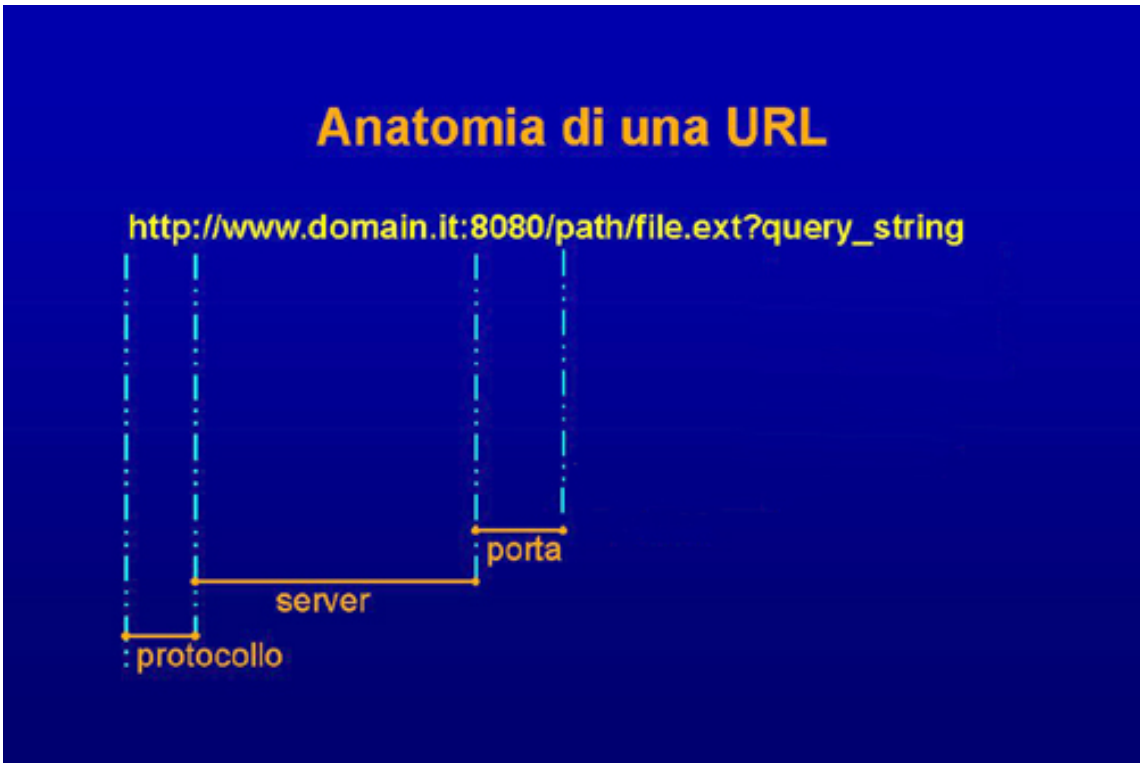


Architettura Web

Benvenuti a questa lezione sulle architetture *Web*. Nella lezione precedente abbiamo appreso quali sono i protocolli utilizzati in *Internet*. Nel corso di questa lezione e delle successive ci occuperemo, in particolare, del protocollo HTTP e del protocollo *CGI*. Andremo a vedere quali sono le architetture *Web* a tre livelli, andando ad analizzare in particolare l'architettura *Server Side Include*, le architetture *Web* API e l'architettura a codice mobile. Ci occuperemo, quindi, di quelle che sono le architetture *Microsoft* per il *Web*, in particolare la tecnologia ASP e la tecnologia *ActiveX*. Infine, andremo a vedere quali sono le applicazioni *Java* che consentono di utilizzare codice mobile su *Internet*. Cominciamo questa lezione sulla architettura *Web*. Dovremmo andare a comprendere quali sono i meccanismi di indirizzamento, quali sono i meccanismi di trasporto e quali sono i meccanismi di formato utilizzati nel *Web*. In particolare, vedremo, che nel *Web* possiamo utilizzare un *Uniform Resource Locator* per indirizzare univocamente ciascuna risorsa. Individuata la risorsa che vogliamo indirizzare utilizziamo come protocollo di trasporto: il protocollo HTTP. HTTP sta per *HyperText Transfer Protocol* ed è un protocollo utilizzato per il trasferimento di ipertesti. Infine, nel corso di questa lezione, vedremo come l'HTML possa essere utilizzato per passare parametri dal *client* al *server Web*. Il protocollo HTTP, come indicato in questa *slide*, prevede essenzialmente due momenti: una richiesta HTTP, con la quale viene invocata una risorsa di cui è stata specificata la corrispondente URL, e una risposta HTTP, con cui il *server Web* invia il documento richiesto. In questo caso si tratta di una pagina HTML

che contiene del testo e delle parti grafiche.

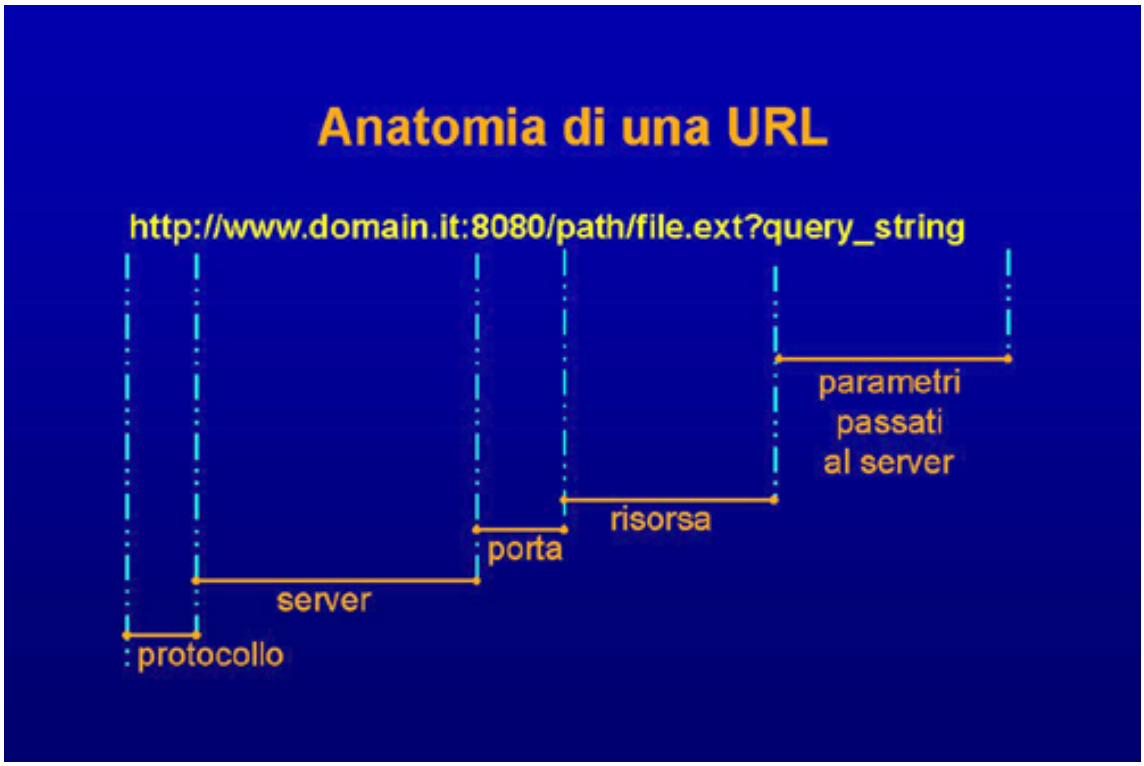
Anatomia di una URL (1/2)



Anatomia di una URL (1/2)

Per comprendere il funzionamento dell'architettura *Web* dobbiamo comprendere quali sono i meccanismi di indirizzamento. In particolare dobbiamo comprendere qual è l'anatomia di una URL, ovvero quali sono gli elementi che costituiscono una URL e qual è il contenuto d'informazione di ciascuno di questi elementi. Una URL è costituita da una serie di elementi che individuano diverse parti della risorsa che vogliamo andare ad individuare. In particolare, la prima parte della URL individua quello che è il protocollo utilizzato per il trasporto. L'URL è un meccanismo di indirizzamento globale che consente, quindi, di utilizzare per il trasporto delle risorse che vogliamo trasportare, non solo il protocollo HTTP, ma anche altri protocolli, per esempio il protocollo FTP o protocolli di questo tipo. In questo caso, questa URL specifica che la risorsa alla quale siamo interessati è accessibile tramite il protocollo HTTP. La seconda parte di una URL individua qual è il *server* sul quale si trova la risorsa alla quale siamo interessati, in questo caso il *server* è quello caratterizzato dall'indirizzo www.domain.it. Il terzo elemento individuato da una URL specifica qual è la porta alla quale è accessibile quel servizio. Normalmente questa parte di URL non è presente, in quanto i *server Web* rispondono alla porta 80. In questo caso non è necessario indicare la porta specifica alla quale viene richiesta questa risorsa.

Anatomia di una URL (2/2)



Anatomia di una URL (2/2)

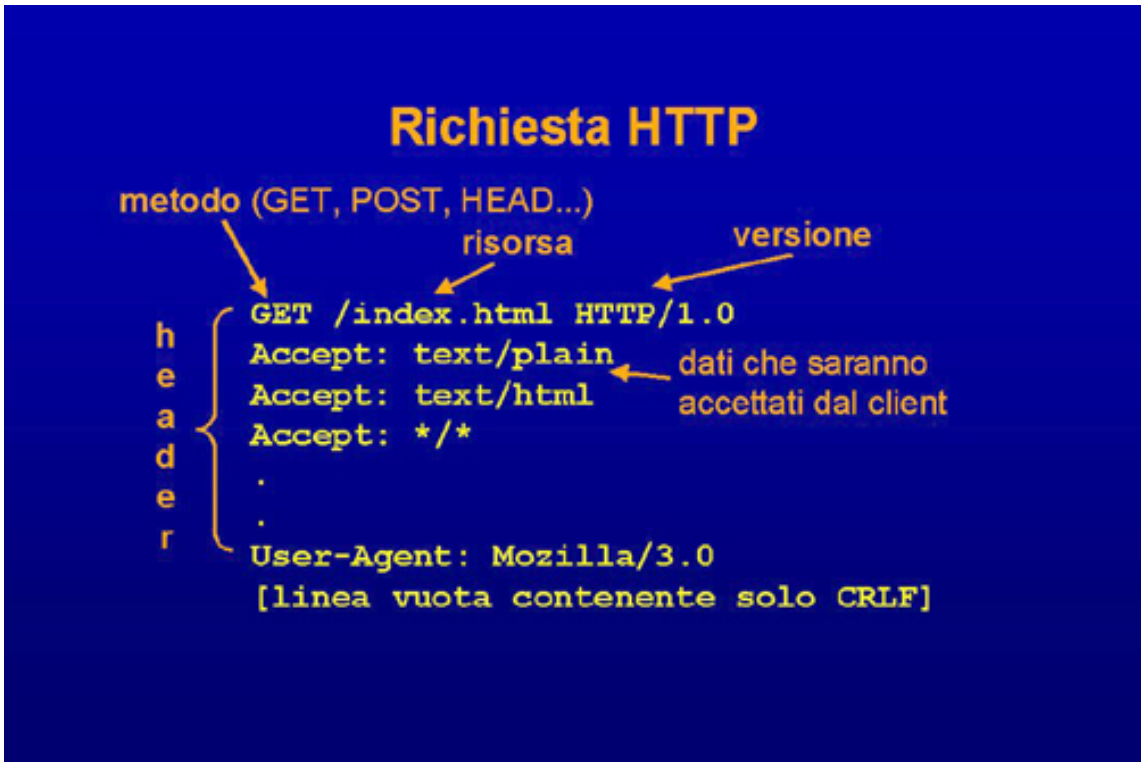
Individuato qual è il *server* e qual è la porta alla quale è distribuita la nostra risorsa, dobbiamo specificare qual è la risorsa alla quale siamo interessati. Questo può essere fatto attraverso il *path name*, che individua in maniera univoca un *file* o un oggetto distribuito da quel *server Web*. Queste quattro parti individuano la risorsa che vogliamo andare a recuperare su di un *server Web*. Vedremo che una URL può essere utilizzata non solo per individuare una particolare risorsa, ma anche per passare parametri dal *client* al *server Web*. In questo caso viene utilizzata una parte della URL, che va sotto il nome di *query_string*. Nel proseguo vedremo che questo elemento, la parte *query_string* di una URL, è utilizzato per inviare dati dal *client* verso il *server*.

Il protocollo HTTP

Il protocollo HTTP

Andiamo a comprendere, ora, quali sono gli elementi che costituiscono il protocollo HTTP, cercando di comprendere appieno quali sono le varie fasi in cui può essere scomposta una connessione HTTP. Il protocollo HTTP è un protocollo *client/server* che prevede quattro fasi: la prima è l'apertura della connessione, con la quale il *client* instaura una connessione diretta verso il *server*. Aperta una connessione il *client* invierà una richiesta HTTP, andando a richiedere una ben precisa risorsa. A questa richiesta seguirà la risposta HTTP, con cui il *server* invia la risorsa che è stata richiesta dal *client*. L'ultima fase della connessione HTTP è la chiusura della connessione stessa, che normalmente viene eseguita dal *server*, ma in alcuni casi potrebbe anche essere eseguita dal *client*, ad esempio perché si è atteso troppo tempo per la risorsa desiderata. È importante notare che HTTP è un protocollo *stateless*, un protocollo nel quale non esiste nessuna relazione tra due richieste, anche se sono sequenziali e anche se si riferiscono ad oggetti che sono contenuti nella stessa pagina HTML. Sarà il *client* a tenere traccia delle relazioni che ci sono tra diverse connessioni HTTP. In particolare, se il *client* intende scaricare una pagina HTML che contiene, oltre a del testo, anche un paio di immagini, ad esempio, sarà il *client* che dovrà eseguire due ulteriori connessioni verso il *server*, per scaricare le due immagini contenute nella pagina HTML.

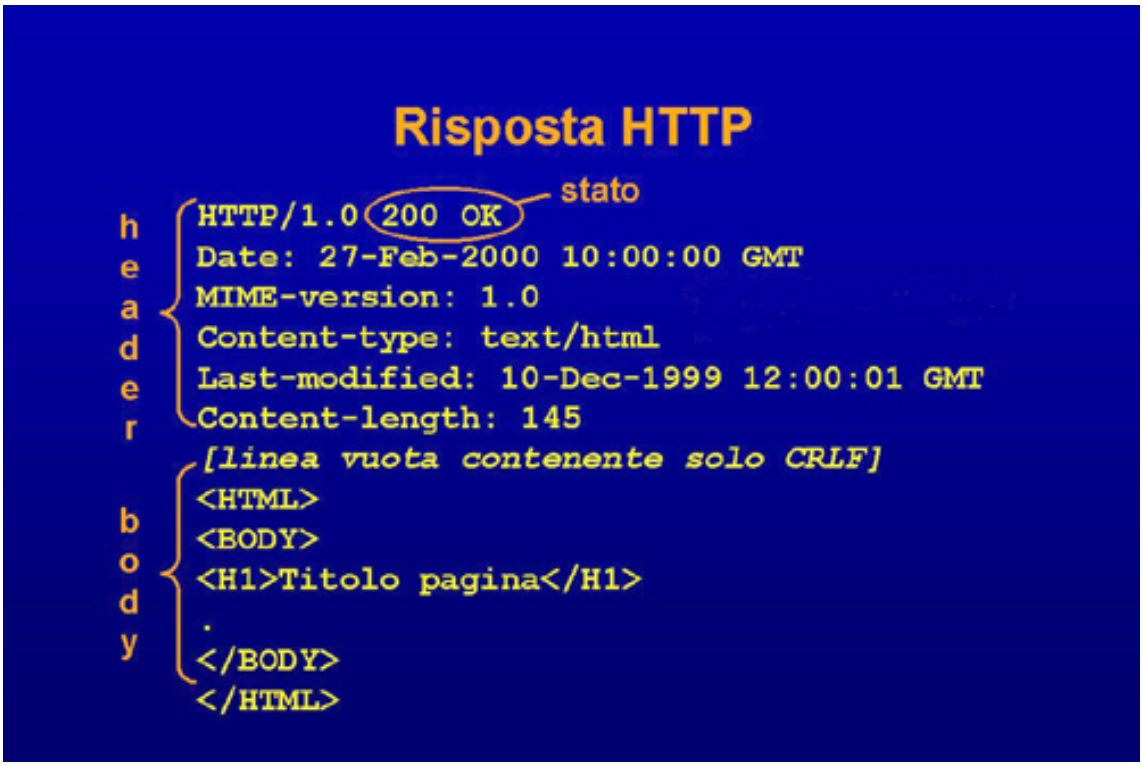
Richiesta HTTP



Richiesta HTTP

Andiamo ad analizzare come sono strutturate sintatticamente le richieste e le risposte HTTP. Questo consentirà di capire in che modo *client* e *server* possono scambiarsi dati utilizzando il protocollo HTTP e in che modo il *client* possa passare dei parametri al *server*. Cominciamo quindi dalla richiesta HTTP. Una richiesta HTTP prevede una serie di elementi e generalmente è composta di due parti, una parte di *header*, che vediamo indicata qui, separata tramite una linea vuota, che contiene soltanto un carattere di ritorno a capo, dalla parte di *body*. La parte di *body* è opzionale e potrebbe non essere presente, come in questo caso. Il primo parametro specificato in una richiesta HTTP è il metodo utilizzato. Si tratta del meccanismo con il quale viene richiesta una determinata risorsa HTTP. Esistono diversi metodi e impareremo a conoscerne le differenze nel corso di questa lezione. Il secondo elemento contenuto in una richiesta HTTP individua qual è la risorsa desiderata. In questo caso stiamo accedendo al *file* index.html a partire dalla radice del *Web server* stesso. Il terzo parametro, che ritroviamo sulla prima riga della richiesta HTTP, specifica quale versione del protocollo HTTP stiamo utilizzando; in particolare, in questo caso, stiamo utilizzando la versione 1.0. Il *client* specifica poi tutta un'altra serie di parametri rivolti al *server*, con la quale descrive alcune delle sue caratteristiche. In questo caso, per esempio, il *client* specifica che accetterà *file* di tipo *text* in formato *plain* o in formato HTML e nell'ultima riga, che vedete nell'*header* HTTP, specifica che si tratta di un *client* di tipo Mozilla, versione 3.0. Questo consente di passare alcune informazioni sul tipo di *client* che andiamo ad utilizzare al *server* che potrà utilizzarle ad esempio per personalizzare l'*output* per un particolare tipo di *server*.

Risposta HTTP (1/3)



Risposta HTTP (1/3)

Andiamo a vedere, ora, in che modo è organizzata la risposta HTTP, ovvero in che modo il *server*, a seguito di una richiesta proveniente dal *client*, specifica i dati da trasmettere al *client*. Così come la richiesta HTTP, anche la risposta HTTP è divisa in due parti. Anche in questo caso la prima parte (*header*) è separata dalla parte di *body* da una linea vuota, contenente soltanto il carattere di ritorno a capo. Nella parte di *body* possiamo riconoscere immediatamente il contenuto di questa risposta HTTP. Si tratta, in questo caso, della pagina HTML che il *server* sta distribuendo al *client*. Riconosciamo le tipiche *tag* che descrivono la struttura di questa pagina HTML. Andiamo ad analizzare però la parte di *header* della risposta HTTP e cerchiamo di comprendere quali sono le informazioni che il *server* invia al *client*. La prima di queste è una informazione di stato: vedete che, in questo caso, il *server* segnala che la richiesta è stata assolta correttamente. In particolare viene utilizzato un codice numerico, il cosiddetto *three digit code*, con cui il *server* specifica lo stato della risposta. La prima cifra di questo *three digit code* specifica il fatto che la richiesta sia andata a buon fine, oppure che ci sia stato un errore. Le altre due cifre consentono di specificare meglio che tipo di errore si è verificato, o se la richiesta è andata a buon fine. Vi sarà familiare, ad esempio, nel corso delle vostre navigazioni *Web*, il codice di errore 404 corrispondente ad una pagina HTML non trovata. Questo è il tipico codice di errore restituito da un *server Web* quando viene richiesta una risorsa non presente su quel *server Web*. Tutti i messaggi di errore di un *server* cominciano con il codice 4.

Risposta HTTP (2/3)

Risposta HTTP (2/3)

Torniamo alla nostra risposta HTTP e andiamo a vedere quali sono gli altri elementi. Il *server* indica anche quale versione del protocollo HTTP andrà ad utilizzare per la sua risposta. E indica un'altra informazione particolarmente importante. Come notate in questa riga, il *server* specifica il tipo ed il sottotipo di documento che andrà a spedire. Questa informazione è fondamentale perché il *client* dovrà comprendere che tipo di documento gli è stato spedito. Non è possibile utilizzare a questo scopo la convenzione, utilizzata localmente dai sistemi operativi, di desumere il tipo di *file* dall'estensione stessa. Potrebbe succedere, infatti, che su un determinato *computer* un estensione abbia un significato e abbia un significato diverso su di un altro *computer*. Ad esempio, molto spesso, nei PC con sistema operativo *Windows* i *file* HTML hanno estensione .htm, mentre molto spesso sulle macchine *Unix* i *file* HTML hanno estensione .html. Questo fatto renderebbe incomprensibile le due estensioni in due macchine che usano un sistema operativo diverso. Per evitare questo problema, come vediamo in questa *slide*, il *server* specifica il tipo ed il sottotipo *MIME* del documento che sta spedendo. *MIME* sta per *Multipurpose Internet Mail Extension* ed è uno standard con cui si può specificare il tipo ed il sottotipo di un determinato documento. Il *client*, leggendo questa informazione, comprenderà che sta ricevendo un documento di tipo testo e in particolare un documento HTML, quindi andrà ad interpretare il documento scaricato come una pagina HTML, andando ad eseguire le formattazioni del caso.

Risposta HTTP (3/3)

Risposta HTTP (3/3)

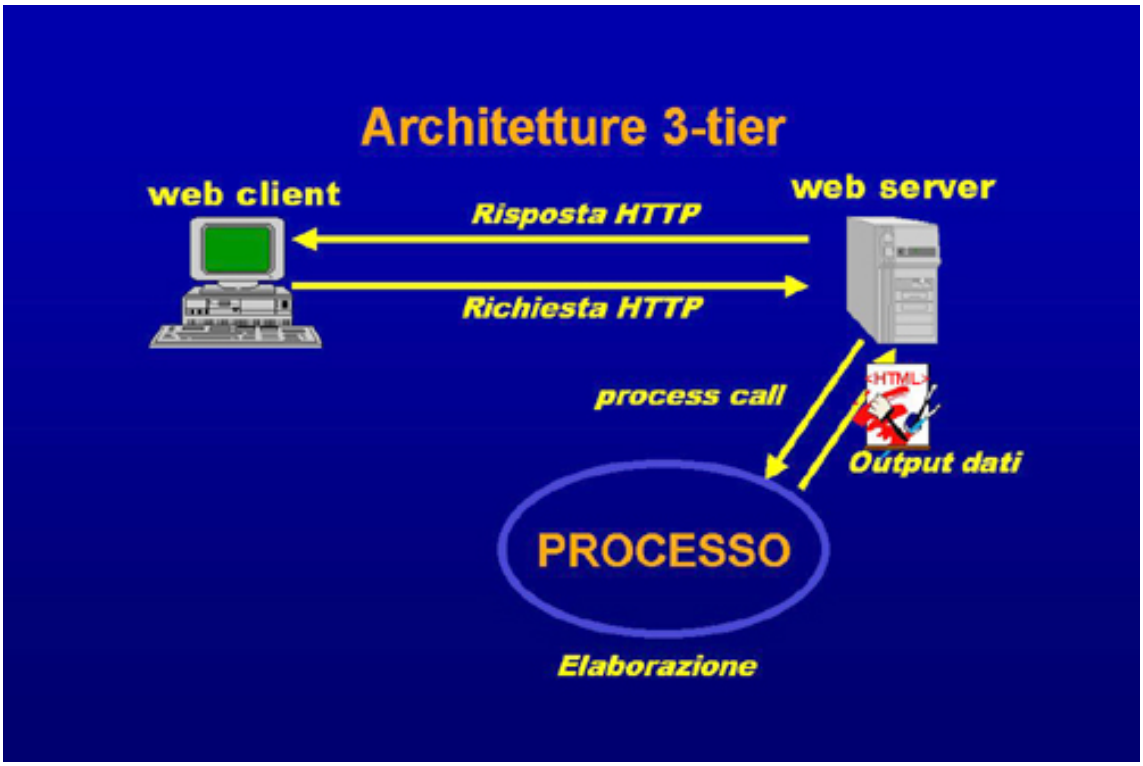
Ci sono, poi, tutta un'altra serie di informazioni che il *server* comunica al *client*: a cominciare dalla data in cui è avvenuta la connessione, in questo caso si tratta di una risposta che è stata spedita il 27 febbraio 2000 alle 10.00, alla data in cui è stato modificato per l'ultima volta questo documento. Questa informazione è particolarmente importante, in quanto consente al *client* di capire se eventuali copie locali di quel documento siano ancora aggiornate o siano state sostituite nel *server* stesso. Notate, infine, che il *server* specifica anche la dimensione del documento che sta trasferendo, in questo caso si tratta di un documento di 145 *byte*. In questo modo il *client* può sapere quale parte di documento ha già ricevuto.

Programmi CGI

Programmi CGI

Abbiamo visto in che modo il protocollo HTTP consente ad un *client* e ad un *server* di trasferire informazioni, di trasferire richieste e trasferire risposte. Il protocollo HTTP può essere utilizzato non solo per trasferire documenti HTML, ma una delle sue applicazioni più interessanti è la possibilità di utilizzarlo per invocare programmi che risiedono sul *server*, a partire da un *client Web*. Questo aspetto di HTTP è particolarmente rilevante, in quanto consente di invocare applicazioni, attraverso un *client Web*, che si trovano su di un *server Web*. In questo caso sarà necessario specificare un protocollo con il quale il *server Web* possa attivare il processo richiesto, possa passare eventuali parametri necessari a quel processo e possa ricevere da quel processo l'*output* HTML da inviare al *client*. Questo protocollo è il protocollo *CGI* (*Common Gateway Interface*), che consente appunto ad un *server Web* di invocare un processo su richiesta di un *client Web*. Vedremo, in particolare, che questo meccanismo consente di sviluppare le cosiddette architetture a tre livelli. Architetture nelle quali l'accesso al sistema informativo avviene non più solo attraverso un *client* ed un *server*, ma avviene attraverso tre distinti livelli: il *client Web*, il *server Web* e il data base applicativo stesso. Vedremo che questa architettura ha una serie di vantaggi, che cercheremo di comprendere nell'ambito di questa lezione.

Architetture 2-tier



Architetture 2-tier

Che cosa significa architettura a tre livelli? Significa che possiamo utilizzare un normale *client Web* standard, senza bisogno di nessuna modifica, per andare ad effettuare una richiesta HTTP che richieda, non una semplice pagina HTML, bensì l'attivazione di un processo che risiede sul *server Web*. Questo processo potrà eseguire una propria elaborazione e potrà generare al volo i dati in *output*. Questi dati, ovviamente, dovranno essere formattati in HTML, in modo che il *Web server* possa girarli al *client Web*, che così otterrà una risposta alla propria interrogazione. Rispetto all'utilizzo di pagine HTML statiche, in questo caso abbiamo l'intervento di un processo che elabora al volo le informazioni, sulla base dei dati passati dal *client* HTTP. Abbiamo realizzato in questo modo, in tutto e per tutto, un meccanismo che consente di invocare delle procedure remote e che consente di eseguire dei processi su di un *server Web*. Dobbiamo comprendere in che modo un *server Web* possa attivare un processo *CGI* e come esso possa essere eseguito. In particolare, i *server Web* individuano nella richiesta HTTP la caratteristica che la richiesta sia relativa ad una pagina HTML, oppure ad un processo *CGI*. In particolare, se la richiesta si trova all'interno di una particolare *directory*, ad esempio la *directory cgi-bin*, il *server* la interpreterà come una richiesta di attivazione di un processo e non come la richiesta di distribuzione di una pagina. Questo processo verrà eseguito in locale e preparerà il proprio *output* per il *client* stesso.

http://130.192.3.92/cgi-bin/time.pl (1/2)

HTTP://130.192.3.92/cgi-bin/time.pl (1/2)

Andiamo a vedere come è possibile scrivere un programma *CGI*, per analizzarne quali sono le caratteristiche e quali sono le potenzialità. In particolare analizzeremo un semplice programma *CGI* scritto in linguaggio Perl. Perl è un linguaggio interpretato, molto utilizzato nelle applicazioni *CGI*, che consente di scrivere in maniera compatta e semplice programmi *CGI*. In questo caso stiamo andando a vedere il codice del programma che corrisponde all'indirizzo 130.192.3.92/cgi-bin/time.pl. Si tratta di una semplice applicazione *CGI* in cui il programma invocato va semplicemente a stampare la data e l'ora corrente, formattando in HTML per presentarla al *client*. Andiamo a vedere come è organizzato questo tipo di programma *CGI*. Allora, in questo caso, il programma deve specificare con quale tipo di interprete debba essere trattato il programma stesso e la prima riga di questo programma ha esattamente questa funzionalità. Dice al *server Web* qual è l'interprete da utilizzare per interpretare il programma descritto di seguito. In questo caso viene utilizzato il programma perl.exe, che si trova nella *directory* perl/bin.

http://130.192.3.92/cgi-bin/time.pl (2/2)

HTTP://130.192.3.92/cgi-bin/time.pl (2/2)

Notate che il programma *CGI*, a questo punto, ha il pieno controllo della connessione HTTP e deve occuparsi di tutti gli aspetti della connessione. In particolare, siccome il programma *CGI* è l'unico a conoscere il tipo di documenti che verrà generato da questa connessione, dovrà specificare che tipo di documento viene generato. In questo caso, vedete, questa istruzione di *print* specifica che il *content-type* del documento generato è di tipo *text*/html. Notate che la sintassi è la stessa che abbiamo visto qualche *slide* fa ed è la sintassi utilizzata e prevista dal protocollo HTTP per specificare il tipo di documento trasferito. A questo punto il *CGI*, vedete, stampa una coppia di ritorno a capo: questo fa si che venga passata dal *server* al *client* una riga contenente solo un carattere di ritorno a capo. Quindi, il *client* interpreta le righe successive come il *body* della risposta HTTP, che dovranno contenere delle informazioni HTML così come specificato dall'*header content-type*. Notate che, a questo punto, il programma *CGI* deve soltanto occuparsi di descrivere l'*output* HTML generato al volo a seconda dei dati corrispondenti a questo istante. In questo caso, il nostro programma *CGI* deve semplicemente stampare la data corrente. Allora, vedete, che andrà a scrivere semplicemente i *tag* HTML necessari a formattare la nostra pagina e poi invocherà la funzione *localtime*() che stamperà la data e l'ora corrente. Quindi, il programma *CGI* eseguirà un'altra *print*, per andare a stampare la chiusura della pagina HTML stessa.

http://130.192.3.92/cgi-bin/time.pl (test)

HTTP://130.192.3.92/cgi-bin/time.pl (test)

In questo modo abbiamo visto come funziona il protocollo *CGI* e come possiamo invocare una risorsa che si trova su *server Web*. Spostiamoci sul *computer* da questo lato e vediamo a quali fasi corrispondono questi procedimenti. Utilizziamo questo *client Web* e andiamo a collegarci ad un *server Web*, che in questo caso risiede sulla stessa macchina, ma questo è un dettaglio, potrebbe trovarsi in un qualsiasi punto della rete *Internet*. Andiamo ad accedere alla risorsa che abbiamo visto descritta nella *slide* precedente. In questo caso probabilmente non si legge, ma andiamo ad indicare che vogliamo accedere all'URL cgi-bin/time.pl, che individua in maniera univoca esattamente il *file* Perl che abbiamo visto poco fa. Invocando questa risorsa il programma *CGI* viene attivato dal *server Web* e vedete che, in questo caso, il programma stamperà un *output* in formato HTML che conterrà la stringa data e ora, così com'era indicato nel programma Perl che abbiamo visto e conterrà l'*output* del comando *localtime*(), che in questo caso riporta semplicemente l'ora in cui è stata eseguita questa connessione. In questo modo abbiamo visto com'è possibile attivare un processo che si trovi su di un *server Web*, andando ad invocarlo attraverso un *client Web*.

Metodi HTTP

Metodi HTTP

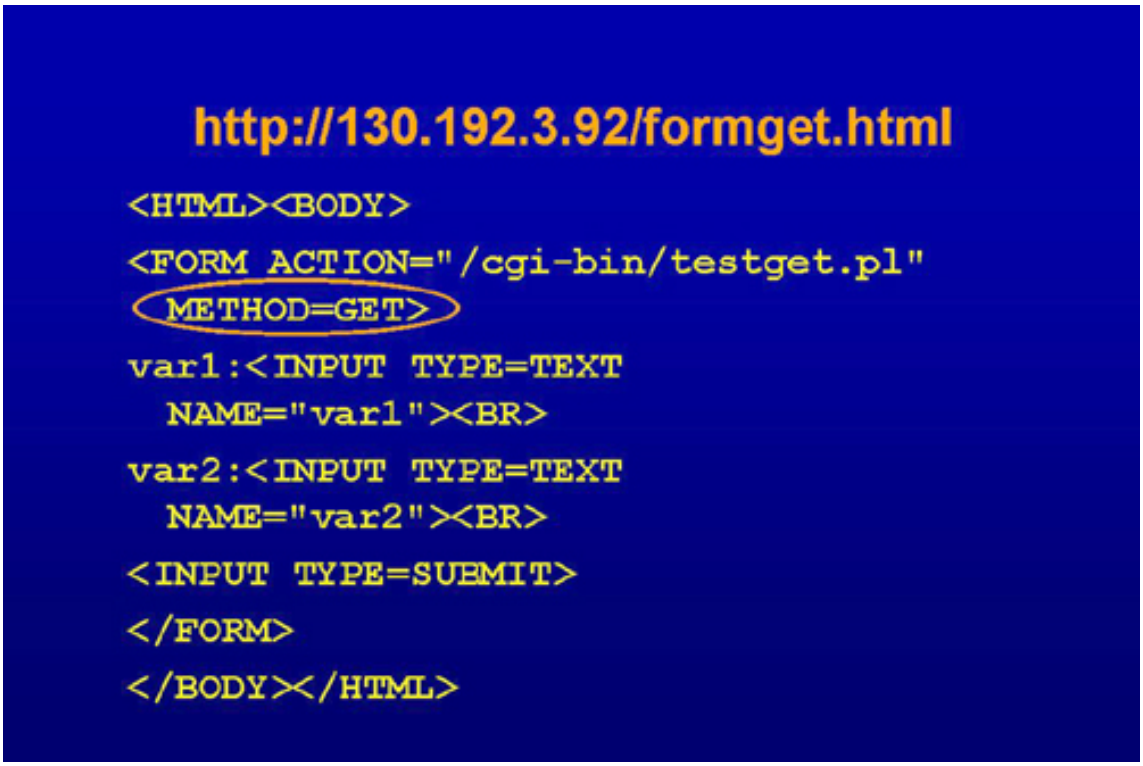
Andiamo a vedere quali sono i meccanismi che HTTP ci mette a disposizione per attivare processi di questo tipo e, soprattutto, in che modo sia possibile passare dei parametri a questi processi. È indispensabile, infatti, quando eseguiamo elaborazioni in tempo reale, che l'utente non solo sia in grado di attivare un processo, ma sia anche in grado di inviare dei dati a quel processo. In particolare, HTTP fornisce due metodi per attivare processi: il primo va sotto il nome di metodo *GET* e il secondo va sotto il nome di metodo *POST*. Nell'utilizzo del metodo *GET* il *server* riceve eventuali parametri dal *client* attraverso il campo *QUERY_STRING*, che viene aggiunto, come abbiamo visto precedentemente, in coda alla URL e li passa al processo *CGI* attraverso una variabile d'ambiente che si chiama, appunto, *QUERY_STRING*. Nel caso di utilizzo del metodo *POST*, invece, il *server* riceve eventuali parametri direttamente nel *body* della richiesta HTTP che, se vi ricordate, non sempre è presente. Quando utilizzeremo il metodo *POST* e quando passeremo parametri dal *client* al *server*, la parte *body* sarà presente e conterrà, appunto, i parametri che il *client* passa al *server*. Il *server*, quindi, gira i parametri così ricevuti al *CGI* invocato, utilizzando lo standard *input* di quel processo. Un processo *CGI* invocato tramite metodo *POST*, andrà a leggere il proprio standard *input* per ritrovare i parametri che gli sono stati passati dal *server Web*.

Form HTML

Form HTML

Occorre specificare un modo con il quale il *client* possa raccogliere dei dati dall'utente e con il quale questi dati possano essere organizzati e passati al *server Web*. Siccome si tratta di organizzare il formato di dati, dovremmo utilizzare a questo scopo l'HTML, che in effetti prevede una serie di *tag* che consentono proprio di inserire dei dati in una pagina HTML e di passarli, attraverso la connessione HTTP, al *server Web*. La sintassi prevista dall'HTML utilizza la *tag* <*FORM*> che consente di specificare una serie di parametri da inviare al *server*. La *tag* <*FORM*> è caratterizzata da due parametri: il parametro *ACTION*, che specifica qual è l'URL obiettivo di questa azione, ovvero quale URL verrà invocato nell'eseguire questa *form*, e il metodo utilizzato. In questo caso, vedete, i parametri verranno passati dal *client* al *server* utilizzando il metodo *GET*. Nella *form* dobbiamo essere in grado di specificare quali sono i campi che devono essere passati al *server* e, in particolare, si utilizza una *tag INPUT* che ha tutta una serie di varianti, delle quali ne vediamo un paio, con la quale è possibile passare dei valori al *server*. Nel primo caso vediamo una *tag INPUT* di tipo *TEXT*, che consente di passare dei dati testuali dal *client* al *server*. Questi dati saranno identificati da un nome, così come specificato nel parametro *NAME*. La penultima riga di questa pagina HTML specifica, invece, un elemento di *input* di tipo *BUTTON*: un bottone la cui pressione provocherà l'esecuzione dell'azione specificata nel parametro *ACTION* della *tag FORM*.

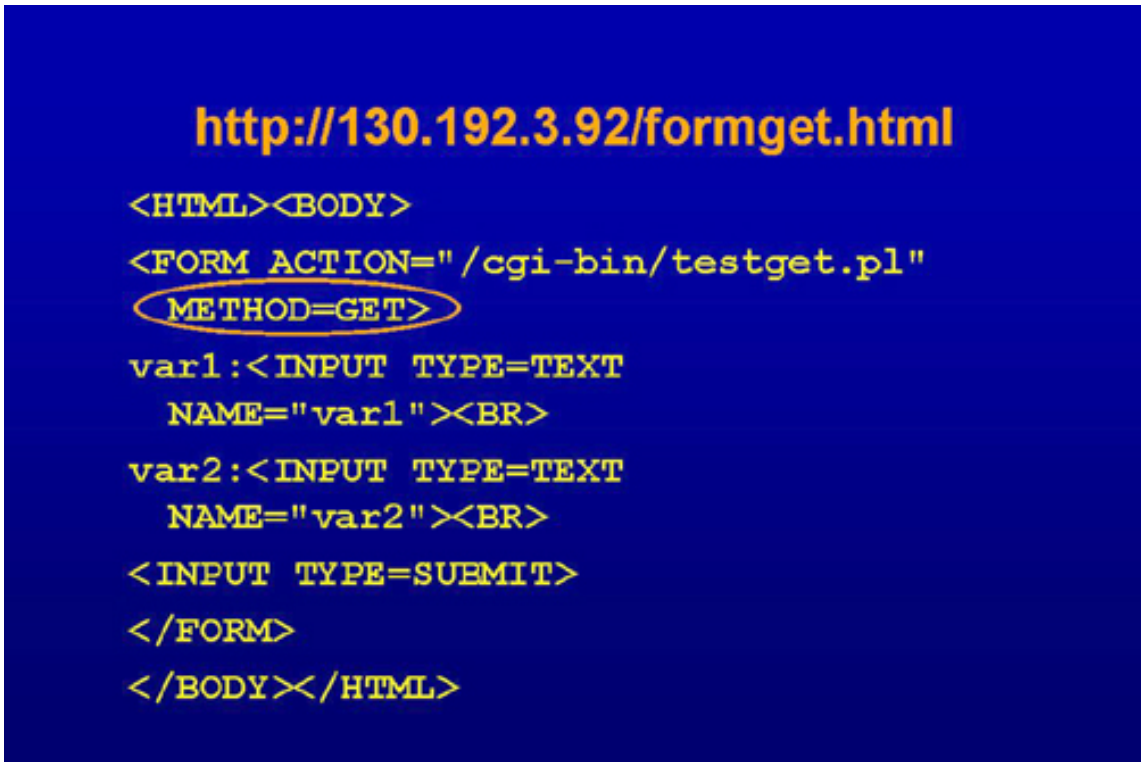
http://130.192.3.92/formget.html



HTTP://130.192.3.92/formget.html

Questi sono tutti gli elementi che abbiamo a disposizione per organizzare la raccolta di dati sul *client*. Alla pressione del pulsante *BUTTON* questi verranno inviati sul *server*. Andiamo a vedere un esempio, semplice, in modo da comprendere esattamente quali sono i meccanismi utilizzati. In questo esempio andremo a vedere una semplice pagina HTML, la pagina formget.html, nella quale vedete specificata una *form*, la quale invoca il *file* cgi-bin/testget.pl, uno *script* Perl che utilizza il metodo *GET*. Notiamo che questa *form* prevede due parametri che vengono passati dal *client* al *server*: entrambi sono parametri di tipo testo, il primo si chiamerà var1 e il secondo si chiamerà var2. Infine, la *form* contiene un pulsante *BUTTON* che consente di inviare semplicemente i dati dal *client* al *server*.

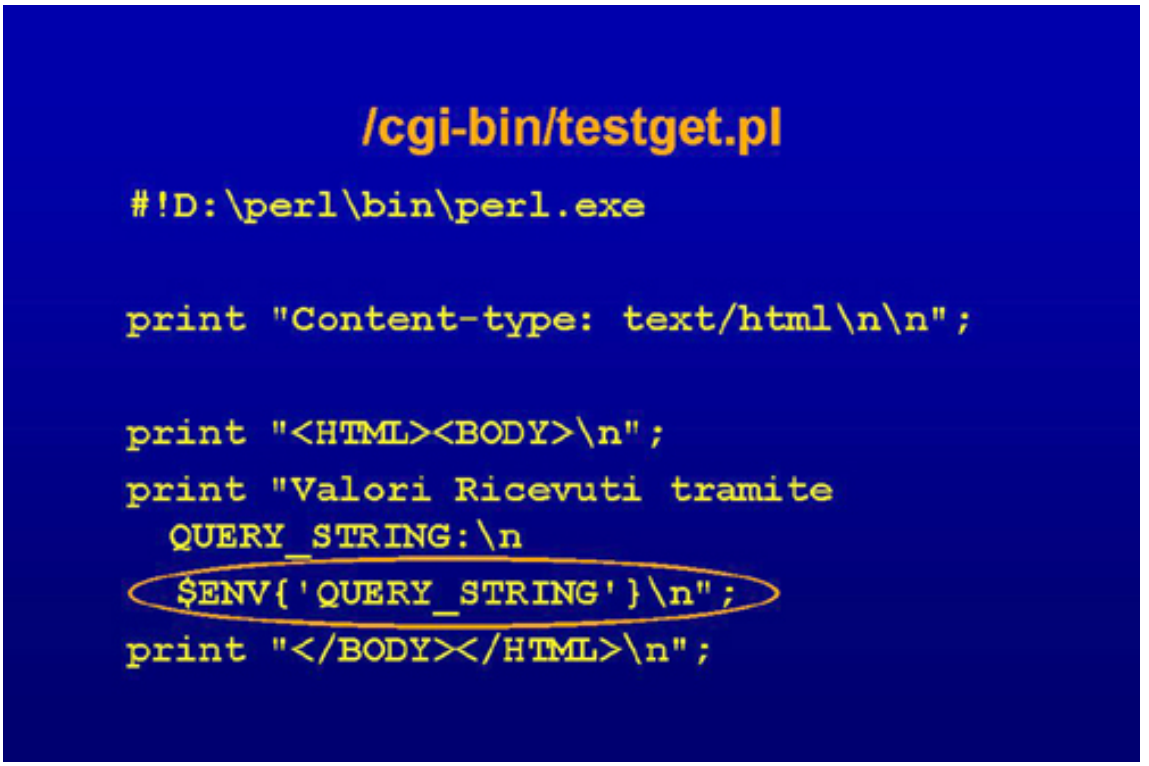
http://130.192.3.92/formget.html (test)



HTTP://130.192.3.92/formget.html (test)

Ci spostiamo sul PC e andiamo a vedere come funzionano questi esempi. Cominciamo dal scaricare la pagina HTML che descrive la *form* che intendiamo utilizzare. In questo caso vado a scrivere formget.html per individuare la pagina HTML descritta dalla *form* che abbiamo appena visto. Come ci aspettavamo, questa *form* ci presenta due campi: sono due campi di tipo testo in cui possiamo inserire due valori. In questo caso inserisco nel campo var1 il valore 1 e nel campo var2 il valore 2. Notate che quando, premendo il pulsante che si trova in fondo a questa *form*, invio questi dati, verrà invocato il processo *CGI* indicato dal parametro *ACTION* della *form*, il quale elaborerà l'*input* ricevuto. Non si può leggere perché i caratteri sono molto piccoli, ma nella *location bar* del mio navigatore sono stati aggiunti esattamente il valore 1 e il valore 2 che ho inserito nella *form* precedente. Questo ha consentito al *client* di inviare, attraverso il metodo *GET*, i parametri al *server*. E notate che, in effetti, il processo *CGI* invocato ha ricevuto dei valori tramite la variabile d'ambiente *QUERY_STRING* e i parametri ricevuti sono esattamente quelli che avevo inviato: valore 1 e valore 2. I dati, in questo caso, sono raccolti in coppie del tipo nome = valore, dove il nome è quello specificato nell'attributo *NAME* della *tag input*.

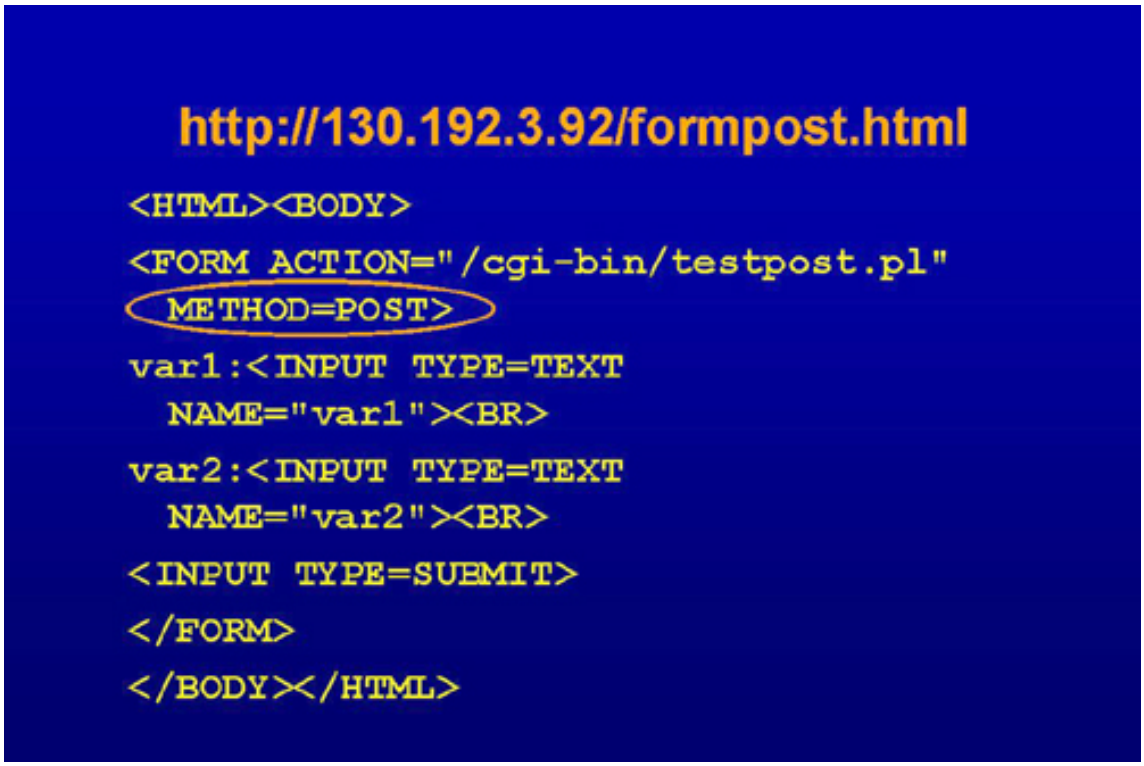
/cgi-bin/testget.pl



/cgi-bin/testget.pl

Il programma *CGI* che eseguirà questo tipo di elaborazione è quello riportato in questa *slide*. Vedete che il programma *CGI*, dopo aver specificato che andrà a produrre un *output* di tipo *text*/html, stamperà semplicemente il valore della variabile di environment *QUERY_STRING*. Quella indicata qui è la sintassi con la quale il Perl consente di stampare il valore di una variabile di *environment*, il cui nome è indicato in questa stringa. Questo è esattamente il valore dei due parametri che abbiamo passato al nostro *server Web*.

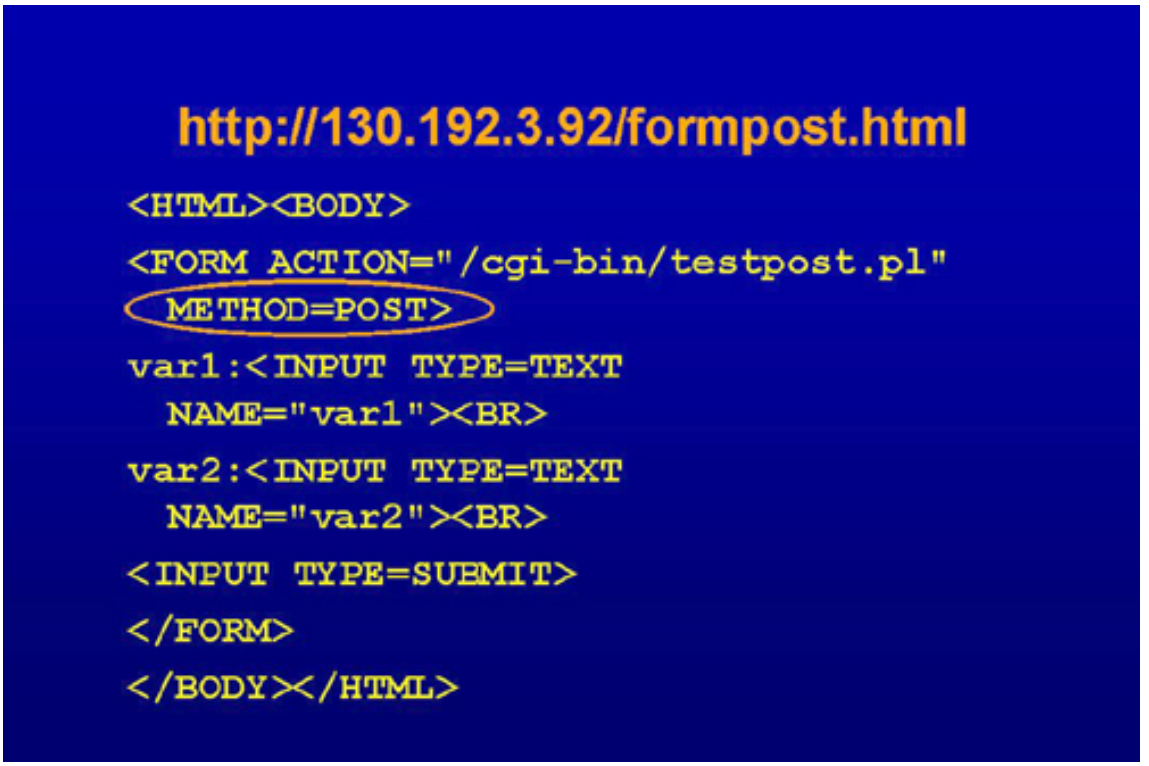
http://130.192.3.92/formpost.html



HTTP://130.192.3.92/formpost.html

Procediamo in questa analisi del protocollo HTTP e del protocollo *CGI*, cercando di comprendere qual è l'altro meccanismo che consente di passare parametri da un *client* ad un *server*. Abbiamo appena visto come funziona il metodo *GET* e abbiamo visto che il programma *CGI* accede ai parametri passati dal *client* attraverso una variabile di ambiente, che ha il nome di *QUERY_STRING*. Vediamo come funziona il metodo *POST*. Osserviamo come è organizzata la pagina HTML che consente di preparare la raccolta dei dati. In questo caso, come vedete, il metodo utilizzato è il metodo *POST* e verrà invocato uno *script CGI* che va sotto il nome di testpost.pl. La *form* passa poi esattamente gli stessi parametri che abbiamo visto prima: ovvero due parametri di tipo *text* dal nome var1 e var2.

http://130.192.3.92/formpost.html (test)



HTTP://130.192.3.92/formpost.html (test)

Ci spostiamo sul PC per vedere come questa pagina HTML possa essere usata per invocare lo *script CGI*, utilizzando questa volta il metodo *POST*. Andiamo a caricare la pagina HTML corrispondente: indicherò questa volta che l'URL al quale sono interessato è formpost.html. Eseguo il *reload* di questa pagina in modo da avere la nuova pagina a disposizione e vado ad inserire i due valori. In questo caso passo i valori val1 e val2 nei due campi var1 e var2. Invio questa *form*, che questa volta utilizzerà il metodo *POST* come meccanismo di trasporto dei parametri, i quali saranno passati allo *script CGI* attraverso lo standard *input*. Se guardate con attenzione nella *location bar*, questa volta vediamo che non è presente nessuna *QUERY_STRING* in coda all'URL, proprio perché il meccanismo di trasporto utilizzato in questo caso è quello del metodo *POST*. Questi parametri, i valori val1 e val2, sono stati passati nel *body* della connessione HTTP e non nel campo *QUERY_STRING* dell'URL.

/cgi-bin/testpost.pl

/cgi-bin/testpost.pl

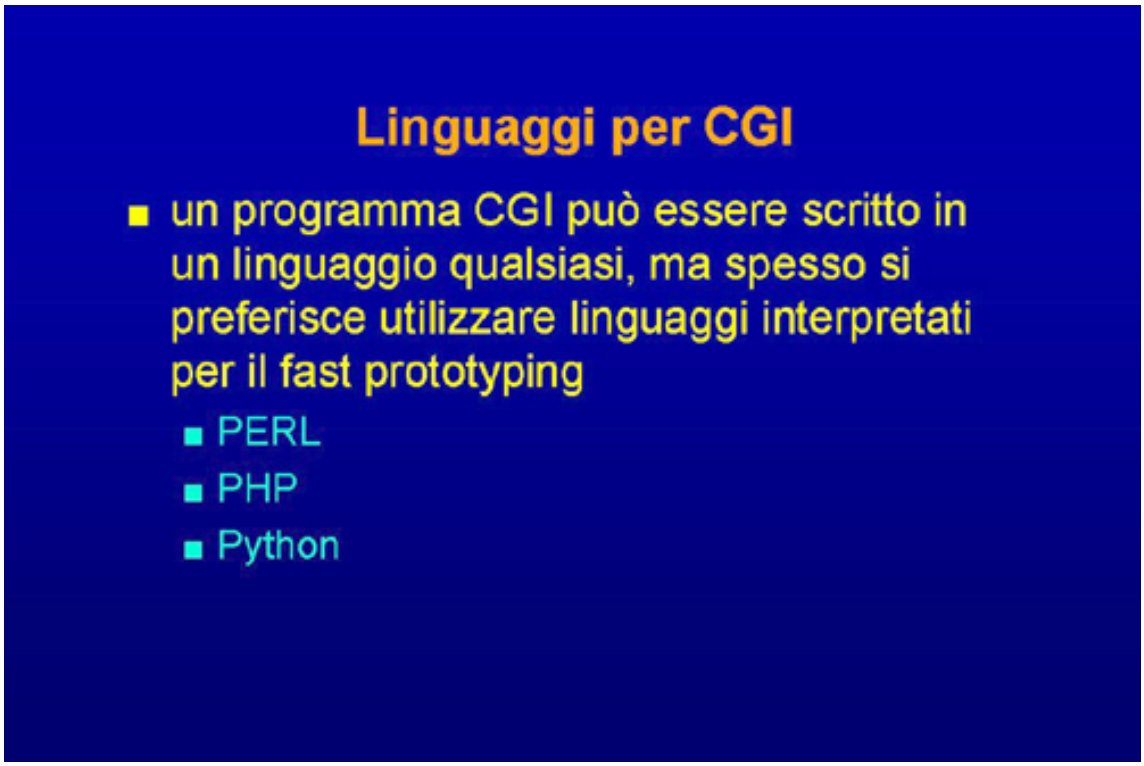
Possiamo spostarci di nuovo sulla *slide* e andremo a vedere qual è in questo caso il codice del programma *CGI* che gestisce questa richiesta. In questo caso, vediamo ancora un semplice programma Perl in cui i parametri, questa volta, non vengono letti attraverso una variabile d'ambiente, ma vengono letti andando a leggere lo standard *input*. Questa è la riga che in Perl consente di leggere una riga dallo standard *input*. In questo caso, la variabile $param conterrà esattamente i parametri che il *client* ha passato al *server Web* utilizzando il metodo *POST*.

URL encoding

URL encoding

Esiste un problema che non abbiamo considerato, ma che va menzionato: il protocollo HTTP utilizza dei caratteri con significato particolare. Dobbiamo, quindi, prevedere un meccanismo per cui se l'utente inserisce uno di questi caratteri particolari, questi vengano transcodificati: ovvero vengano convertiti in caratteri trasmissibili senza problemi sulla connessine HTTP. Utilizziamo l'esempio che abbiamo visto prima e passiamo come valore al nostro *script CGI* una variabile che contenga uno spazio e un carattere ;, che sono entrambi caratteri speciali. Ci accorgiamo che inviando questi valori dal *client* al *server*, il *client* effettuerà una transcodifica, ovvero andrà a convertire questi caratteri particolari, lo spazio e il punto e virgola, in due valori che siamo trasmissibili sul protocollo HTTP. Questo procedimento va sotto il nome di URL *encoding* ed è essenzialmente una procedura eseguita dal *client* prima di trasmettere caratteri particolari. Come riportato in questa *slide*, viene eseguita transcodifica per gli spazi (che vengono transcodificati con il carattere +), per i segni di interpunzione (esclusi i caratteri indicati qui tra parentesi) e per i caratteri ASCII, che vengono sostituiti con il loro codice ASCII esadecimale preceduto dal carattere %. Nell'esempio prima avevamo visto che il carattere ; era stato sostituito dalla sequenza %3b. Questo significa che il codice ASCII esadecimale del carattere ; è il valore 3b. Il programma *CGI* dovrà quindi occuparsi, non soltanto di ricevere i parametri con il metodo *GET* o con il metodo *POST*, ma dovrà occuparsi anche della loro decodifica. Dovrà andare a riconvertire correttamente quei caratteri che hanno subito URL *encoding*.

Linguaggi per CGI



Linguaggi per CGI

Questa analisi che abbiamo portato al termine, ci ha consentito di comprendere quali sono i meccanismi che regolano il protocollo HTTP e in che modo il protocollo HTTP possa essere utilizzato per invocare processi remoti. In particolare, abbiamo visto quali sono i metodi utilizzati dal protocollo HTTP per invocare processi remoti e abbiamo visto alcuni esempi *CGI* scritti in linguaggio Perl. Perl non è l'unico linguaggio utilizzabile per il programmi *CGI*, ma anzi può essere utilizzato un linguaggio qualsiasi. Possono anche essere utilizzati i linguaggi compilati, come ad esempio il C o il C++, ma generalmente si preferisce utilizzare linguaggi interpretati, perché questo consente di sviluppare più rapidamente prototipi. Tra il linguaggi più diffusi vi è sicuramente il Perl. Altri linguaggi molto utilizzati per la programmazione *CGI* sono il linguaggio PHP ed il linguaggio *Python*, che sono delle varianti di Perl che si avvicinano al linguaggio C. Quello che abbiamo visto in questa lezione è quindi un introduzione al protocollo HTTP. Nelle prossime vedremo quali sono le altre architetture a tre livelli e quali sono le caratteristiche di ciascuna di essa.

Importanza del dimensionamento delle risorse

Cosimo Laneve

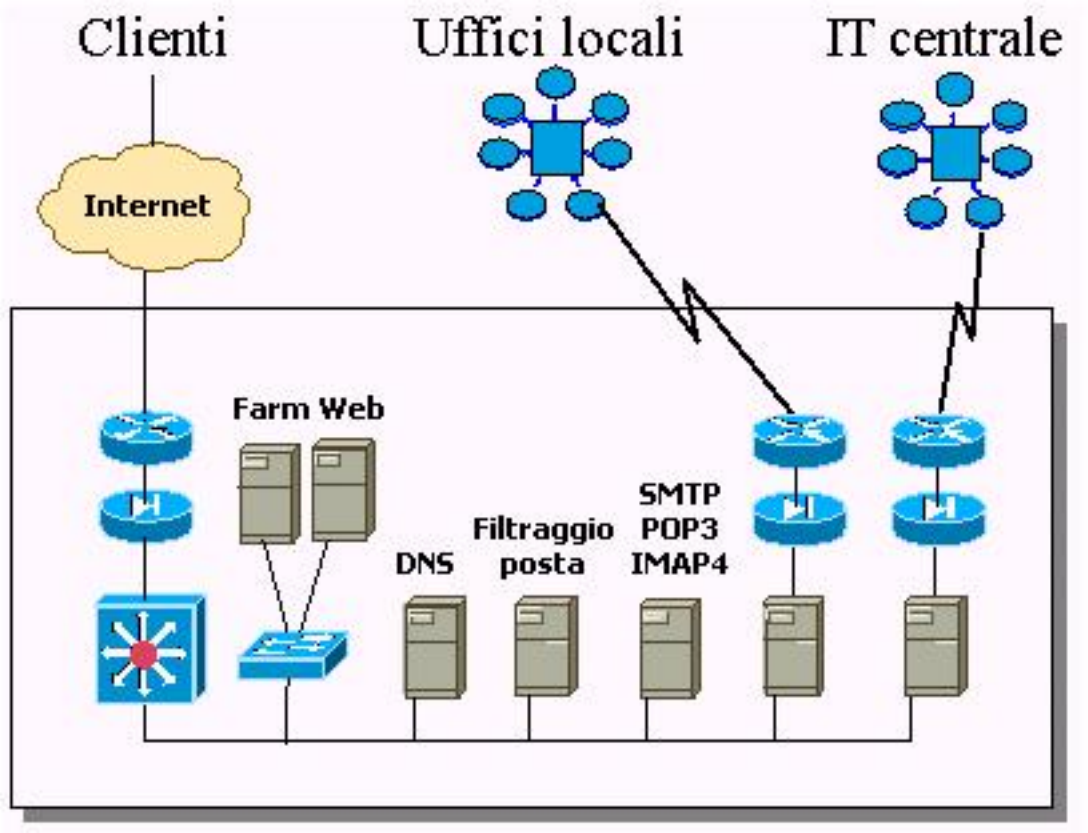
13.1.4 (Valutare un sito Web usando principi di buona progettazione, strutturazione e formattazione)

Importanza del dimensionamento delle risorse

Il dimensionamento delle risorse ha lo scopo di garantire che le capacità IT (ad esempio il supporto applicativo, le infrastrutture fisiche e così via) sulle quali si basano le soluzioni degli ASP (*application services provider*) siano in grado di soddisfare le mutevoli richieste dei clienti, che le risorse esistenti vengano utilizzate al meglio e che i necessari aggiornamenti vengano completati nel modo più rapido ed economico possibile.

Importanza del dimensionamento delle risorse

Per un *provider* di soluzioni applicative è importante riuscire a soddisfare le richieste dei clienti. Se non si dispone delle capacità sufficienti per offrire i livelli di servizio richiesti è possibile incorrere in tempi di risposta prolungati, *time-out*, errori e interruzione delle applicazioni.



risorse in un server Web

Per evitare negazioni di servizio è fondamentale che gli ASP creino infrastrutture

capaci di gestire non solo i livelli *standard* di richieste, ma anche volumi di richieste molto più elevati. Nell'ambito della gestione delle capacità vengono utilizzati sistemi OSS (*Operations Support Systems*) per verificare che l'utilizzo delle risorse esistenti nelle soluzioni ASP sia adeguato ai livelli di servizio stabiliti.

Se eseguito correttamente, il processo di gestione delle risorse consente di valutare la quantità di *hardware* necessaria per gestire il volume di richieste degli utenti relative a una particolare soluzione ASP. Questo tipo di valutazione è utile per individuare i punti deboli delle infrastrutture su cui si basano le soluzioni ASP e i colli di bottiglia che possono causare inefficienza nelle prestazioni. Mediante l'aggiunta di componenti *hardware* o la ridefinizione dei requisiti di una soluzione ASP è possibile ottimizzarne le prestazioni eliminando i punti deboli.

Il dimensionamento è un processo mediante il quale il personale tecnico di un ASP predispone la struttura della soluzione applicativa per futuri ampliamenti che consentano di far fronte a un numero maggiore di richieste. Se per la gestione delle risorse si utilizzano strumenti automatici e adatti a diversi tipi di applicazioni, è possibile ridurre la necessità di intervento umano, aumentare il grado di soddisfazione dei clienti e la redditività del modello ASP.

Concetti fondamentali del dimensionamento delle risorse per gli ASP

Il dimensionamento delle risorse ha la funzione di valutare se l'infrastruttura è in grado di offrire un livello di prestazioni accettabile e adeguato alle esigenze.

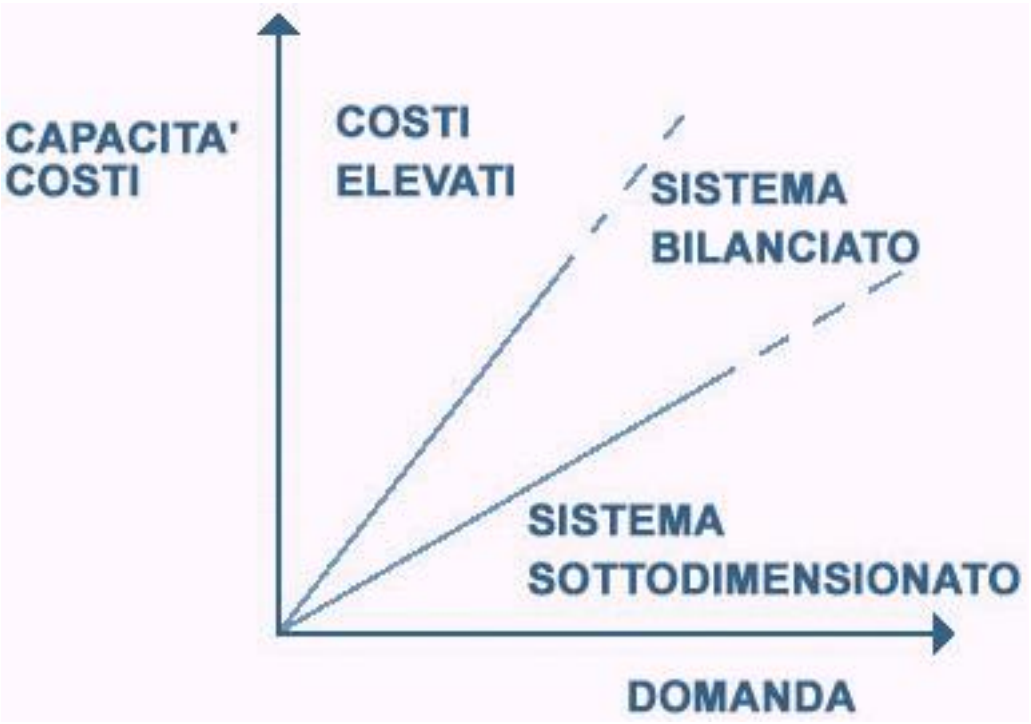


diagramma di bilanciamento dei costi

Il processo include:

- il monitoraggio delle richieste attuali dei clienti relativamente alle risorse IT e la creazione di previsioni in merito ai futuri requisiti dei clienti.
- La canalizzazione delle richieste dei clienti, ad esempio in combinazione con i *Service Level Agreement*, allo scopo di distribuire gli effetti dei picchi di utilizzo.
- L'acquisizione delle risorse necessarie in base ai requisiti dei clienti e la creazione di previsioni in relazione alle esigenze future (come regola generale è preferibile disporre di capacità superiori rispetto alle effettive necessità in modo da essere in grado di fare fronte ad eventuali picchi imprevisti. I costi derivanti dalla mancata capacità di soddisfare le richieste dei clienti sono infatti notevolmente maggiori rispetto ai costi necessari per l'acquisizione delle risorse necessarie).
- Il monitoraggio delle prestazioni e del *throughput* dei servizi ASP.
- Il monitoraggio delle prestazioni dei componenti che costituiscono l'infrastruttura di supporto e l'esecuzione di attività di ottimizzazione che consentano di trarre il massimo vantaggio dalle risorse esistenti.
- La definizione di un piano relativo alla disponibilità di risorse che consenta all'ASP di offrire servizi con un livello di qualità conforme a quanto stabilito nel *Service Level Agreement*.

Il dimensionamento delle risorse ha lo scopo di bilanciare i seguenti fattori:

- **Offerta e domanda**. Il processo consente di verificare che le risorse disponibili (ad esempio la potenza del processore e lo spazio di archiviazione) corrispondano alle richieste attuali e future dei clienti delle soluzioni ASP.
- **Costi e capacità**. È necessario verificare attentamente che i costi sostenuti per l'acquisizione delle capacità siano giustificabili non solo dal punto di vista delle esigenze aziendali, ma anche che le capacità acquisite consentano di utilizzare le risorse nel modo più efficiente possibile.

Un metodo estremamente semplicistico per la valutazione delle capacità consiste nel calcolare il numero di utenti che accedono a una determinata soluzione ASP in relazione al carico di richieste che ogni utente invia ai diversi componenti della soluzione. Questo semplice calcolo può essere utilizzato per potenziare le risorse che possono ridurre le capacità del sistema (CPU, RAM, spazio su disco e larghezza di banda) in modo che siano in grado di supportare i livelli di utilizzo presenti e futuri.

Dimensionamento di un sito

Il processo di dimensionamento delle risorse è costituito da diversi sottoprocessi, che includono a loro volta varie attività.

I processi che contribuiscono al dimensionamento delle risorse

Tali sottoprocessi sono descritti di seguito:

- **gestione delle richieste**. Questo sottoprocesso ha l'obiettivo di garantire che i requisiti aziendali futuri dell'ASP in relazione ai servizi IT vengano considerati, pianificati e implementati in modo tempestivo. A questo scopo, il *team* preposto alla gestione delle risorse ne analizza l'utilizzo attuale all'interno delle varie soluzioni ASP per valutare le tendenze predominanti e sviluppare le necessarie previsioni. I requisiti futuri vengono definiti dalla direzione amministrativa dell'ASP, che ha il compito di analizzare costantemente le esigenze presenti e future della clientela.
- **Gestione del carico di lavoro**. Questo sottoprocesso ha l'obiettivo di interpretare le richieste degli utenti in termini di carico di lavoro sui vari componenti della soluzione ASP (le diverse applicazioni utilizzate per creare la soluzione) e utilizzare questa analisi per determinare le risorse necessarie. Il processo traduce in carico di lavoro sia le richieste presenti sia le richieste previste per il futuro.
- **Gestione delle prestazioni**. Questo sottoprocesso ha l'obiettivo di verificare le prestazioni delle soluzioni ASP utilizzate dai clienti e le prestazioni delle singole risorse sulle quali tali soluzioni si basano. Mediante tale processo è possibile monitorare le prestazioni di tutte le soluzioni ASP (in base ai parametri definiti nei contratti SLA) e delle relative risorse sottostanti. Tutti i dati raccolti vengono registrati, analizzati e documentati. Se necessario, il *team* preposto al dimensionamento delle risorse verificherà che le prestazioni delle soluzioni siano conformi ai requisiti aziendali.

I dati raccolti vengono analizzati e utilizzati per eseguire prove di ottimizzazione e definire profili mediante i quali identificare e impostare valori soglia e avvisi. Quando vengono generati report delle eccezioni o avvisi, è necessario analizzarli e documentarli e quindi intraprendere le necessarie azioni correttive. In genere è preferibile impostare le soglie e gli avvisi su valori inferiori rispetto a quelli stabiliti nel **Service Level Agreement**. In questo modo sarà possibile intraprendere le necessarie azioni correttive prima che si verifichino effettive inadempienze ai contratti SLA.

I servizi e le modifiche

I servizi

La gestione dei servizi rappresenta uno dei punti chiave e ha lo scopo di fornire e supportare i servizi IT che corrispondono ai requisiti aziendali dei clienti dell'ASP. La gestione delle relazioni con i clienti e la gestione dei servizi sono processi strettamente correlati. Tuttavia, l'obiettivo della gestione dei servizi consiste soprattutto nel fornire i servizi concordati con i clienti e, pertanto, privilegia gli aspetti di tipo operativo e tattico, mentre la gestione delle relazioni con i clienti ha una funzione più che altro strategica. La gestione delle relazioni con i clienti consente inoltre di fare previsioni sui costi che sarà necessario sostenere per soddisfare i futuri requisiti dei clienti relativamente alle prestazioni.

Le modifiche

La gestione delle modifiche rappresenta un aspetto molto importante nell'ottica dell'integrità e del corretto funzionamento di un sistema. Il processo di controllo delle modifiche offre l'opportunità di sottoporre le modifiche a una procedura di approvazione e di analizzare attentamente le modifiche richieste dai clienti. Questa analisi consente di valutare le richieste, ordinare le risorse necessarie in tempo utile e adeguare i piani relativi alle capacità in base alle modifiche implementate. Il controllo delle modifiche consente di definire uno schema dettagliato che specifica come e quando le modifiche verranno implementate. In questo modo è possibile ridurre il rischio che vengano apportate modifiche all'ambiente ASP senza che prima ne vengano attentamente valutate le possibili conseguenze sulle richieste dei clienti e sulle risorse.

Durante il processo di gestione delle modifiche è opportuno creare un documento che definisca in modo dettagliato il flusso di lavoro e le modifiche da implementare, incluso lo scopo della modifica, il risultato che questa comporterà e il momento in cui dovrà essere implementata. Questa definizione viene utilizzata dai processi di gestione della disponibilità e delle risorse per determinare gli effetti delle modifiche sulle capacità dei sistemi.

Valutazione dei rischi

Il processo di gestione delle modifiche dovrebbe includere una valutazione dei rischi che le modifiche da implementare potrebbero comportare sulle capacità e le richieste esistenti. Per ogni modifica viene valutato il livello di rischio potenziale (che può essere assente, di *media* entità o molto elevato), che rappresenta un fattore fondamentale di cui è necessario tenere conto durante il processo di approvazione della modifica.

- **Processo di Approvazione**. Il processo di approvazione si basa sulle informazioni finora descritte. Per semplificare lo svolgimento del processo, è consigliabile che le modifiche vengano sottoposte agli organi di approvazione utilizzando un formato *standard*. Ad esempio, è possibile creare un modulo di esempio precompilato che chi propone le modifiche potrà modificare in base alle esigenze. È inoltre importante definire indicazioni di massima in relazione ai tempi che dovrebbero intercorrere tra la proposta e l'approvazione delle modifiche. Per le modifiche più urgenti o che è necessario implementare prima dei tempi stabiliti dovrà essere utilizzato lo stesso formato, sollecitandone tuttavia l'approvazione.
- **Procedure di verifica delle modifiche**. In seguito all'implementazione e all'esecuzione della modifica, è necessario fornire agli organi di approvazione le informazioni necessarie per verificare che la modifica abbia prodotto gli effetti

desiderati. Quindi, in base ai risultati della valutazione, è possibile intraprendere eventuali azioni correttive (ad esempio, se la modifica non ha avuto gli effetti voluti potrebbe essere necessario eseguirne il *roll-back*).

Equazione per il calcolo delle capacità

Per stabilire la quantità di risorse necessaria per un determinato servizio occorre analizzare i **picchi** di utilizzo. Se si è in grado di soddisfare le richieste dei clienti durante i periodi di massimo utilizzo del servizio, le risorse utilizzate saranno sicuramente sufficienti nei periodi in cui il servizio viene utilizzato in misura minore.

L'equazione su cui si basa il dimensionamento delle risorse include tre **fasi**. Nella prima fase vengono valutate le richieste, nella seconda vengono calcolati i carichi di lavoro e nell'ultima fase viene calcolata la quantità di risorse necessarie. Di seguito viene riportato un esempio semplificato dell'equazione appena descritta.

Calcolo delle richieste

Richieste totali = Numero di utenti simultanei x Richieste dei singoli utenti

Calcolo dei carichi di lavoro

Carico di lavoro A = Richieste totali x Carico di lavoro per le singole richieste dell'applicazione A

Carico di lavoro B = Richieste totali x Carico di lavoro per le singole richieste dell'applicazione B

Calcolo delle risorse

Spazio di archiviazione (MB) =
Carico di lavoro A x MB necessari per ogni unità del carico di lavoro A
+ Carico di lavoro B x MB necessari per ogni unità del carico di lavoro B

Potenza CPU = Carico di lavoro A x Potenza CPU necessaria per ogni unità del carico di lavoro A
+ Carico di lavoro B x Potenza CPU necessaria per ogni unità del carico di lavoro B

Larghezza di banda =
Carico di lavoro A x Larghezza di banda per ogni unità del carico di lavoro A
+ Carico di lavoro B x Larghezza di banda per ogni unità del carico di lavoro B

Nell'esempio si presuppone che entrambe le applicazioni risiedano nello stesso *computer*. Risolvendo l'equazione sarà possibile stabilire la quantità di utenti che l'applicazione può gestire nei periodi di massimo utilizzo. Questo metodo consente di calcolare facilmente il numero di utenti che l'applicazione è in grado di supportare. Facciamo inoltre alcune considerazioni:

- diminuendo il carico di lavoro sull'applicazione correlato alle richieste dei singoli utenti è possibile incrementare il numero di utenti supportati. A questo scopo è necessario pianificare, programmare e configurare adeguatamente l'applicazione per ottimizzare l'utilizzo delle risorse.
- incrementando le capacità delle risorse che possono limitare le prestazioni

del sistema (ad esempio RAM, larghezza di banda, licenze e *server*) è possibile incrementare il numero di utenti supportati.

La scalabilità **verticale** consente di aggiungere capacità a una risorsa esistente. Un esempio di scalabilità verticale è rappresentato dall'aggiunta di più processori, RAM o spazio su disco a un *server* esistente. La scalabilità **orizzontale** implica invece l'aggiunta di risorse, quali *server*, connessioni di rete o personale.

Risorse che possono ridurre le capacità del sistema

Il primo passaggio nell'identificazione delle risorse che possono ridurre le capacità del sistema consiste nell'analizzare i termini del cosiddetto *Service Level Agreement* (ciò che si è offerto al cliente). Generalmente, gli ASP sono responsabili della gestione dei seguenti aspetti delle soluzioni:

- infrastruttura della soluzione applicativa;
- connettività di rete;
- interfaccia del cliente con l'applicazione.

Indipendentemente dal metodo di distribuzione utilizzato per l'applicazione, ai fini della pianificazione delle capacità è necessario monitorare quattro risorse fondamentali, ovvero processore, memoria (RAM), array/sottosistema di dischi e rete. Di seguito vengono descritti gli aspetti di ognuna di queste risorse che è necessario monitorare nel contesto di una soluzione ASP:

- **Processore**. L'utilizzo del processore da parte di un determinato componente *hardware*. Un processore che viene utilizzato per più dell'**80%** può indicare l'esistenza di un problema di insufficienza delle capacità. Non è raro che un'applicazione includa processi a *thread* singolo, ovvero processi che possono accedere a un solo processore per eseguire le transazioni. È possibile quindi che un processo poco efficiente utilizzi un processore al 100 percento mentre tutti gli altri processi nella stessa partizione lo utilizzano allo 0 percento. Una situazione di questo tipo indica dei problemi nel codice dell'applicazione, che dovrebbe pertanto essere riscritta.
- **Memoria**. L'utilizzo della memoria dovrebbe essere monitorato per tutte le partizioni di memoria allocate in modo indipendente. La memoria insufficiente rappresenta uno dei problemi più comuni per gli ASP. In genere, i problemi di utilizzo della memoria si manifestano inizialmente attraverso incrementi di utilizzo del processore e dell'*array* di dischi, in quanto queste risorse vengono utilizzate maggiormente per gestire la memoria virtuale ripartendone i dati tra l'*array* di dischi e la RAM disponibile.
- **Dischi**. I segnali che indicano un problema di utilizzo dell'*array* di dischi consistono generalmente nella mancanza di spazio su disco o nell'utilizzo **eccessivo** di una parte dell'*array*.
- **Rete**. Per rete si intendono tutti gli aspetti che riguardano le risorse che risiedono sulla LAN aziendale, inclusi l'utilizzo delle singole schede di rete, i tempi di latenza, la larghezza di banda disponibile e le prestazioni di *switch* e *router*.

Per ognuna di queste risorse è possibile eseguire un monitoraggio più o meno dettagliato. In genere, il sistema operativo o i sottosistemi di dischi dispongono di meccanismi in grado di monitorare queste risorse in modo molto preciso.

Siti ridimensionabili

Cambia la risoluzione del monitor e il sito cambia. E' un errore frequente di chi si trova a sviluppare per la prima volta pagine *Web*. Spesso si ragiona basandosi su parametri cartacei, quando tutto rimane fermo, fisso e immutabile. Così si cerca di ottimizzare il sito in modo che non compaiano barre orizzontali o verticali, ma poi ci si dimentica di domandarsi come si vedrà il sito a una risoluzione del monitor diversa da quella in cui si lavora?.

Uno dei pregi del *Web* è infatti la relativa indipendenza dei **contenuti** dai **contenitori**: le pagine *Web* possono essere viste infatti in moltissimi modi. Il che vuol dire anche che posso girare nel *Web* a partire da qualsiasi piattaforma (*Windows*, *Linux* o *Macintosh*), con qualsiasi *browser* (*Internet Explorer*, *Netscape Navigator*, Mozilla, Opera...), e a qualsiasi risoluzione dello schermo (640x480, 800x600, 1024x768, eccetera). Ma potrei vedere pagine *Web* anche su PALM!

Bisogna sapere adattare il proprio sito a molteplici situazioni. Oggi la maggior parte degli utenti *Internet* imposta il proprio monitor in questo modo (dati tratti dal sito **Netmechanic**):

- il 50% degli utenti utilizza una risoluzione di 800x600;
- il 40% naviga a 1024x768;
- circa il 10% utilizza ancora una risoluzione di 640x480.

I dati sono solo indicativi e non tengono conto del *target*: infatti se dovessimo sviluppare un sito di ICT, dovremmo anche tenere conto del fatto che chi opera nel ramo informatico di solito utilizza monitor piuttosto grandi che gli consentono una risoluzione di 1024 x 768 (con cui è più semplice lavorare...).

In ogni caso, visto che oramai gli utenti che utilizzano risoluzioni del monitor di 800x600 e quelli che invece preferiscono 1024x768 si equivalgono, è opportuno sviluppare il sito in modo che si veda in maniera corretta a entrambe le risoluzioni.

Come impostare un sito indipendente dalla risoluzione

Iniziamo dunque a ottimizzare il nostro sito a 800 x 600: si tratta di evitare che nel sito compaiano barre orizzontali e verticali (in quest'ultimo caso solo se lo si ritiene opportuno). Impostiamo dunque il sito a una grandezza di 780x450 (con *Windows* XP, che utilizza delle barre più grosse, queste grandezze diventano: 779x430).

Dopodiché abbiamo diverse possibilità. A risoluzioni superiori di 800x600:

- il sito rimane allineato a sinistra;
- il sito rimane centrato a qualsiasi risoluzione;
- il sito si ridimensiona, occupando la totalità dello schermo;

A seconda del tipo di *layout* potremo preferire una soluzione oppure l'altra. Ma

analizziamo le varie situazioni nel dettaglio.

Utilizziamo le tabelle in percentuale

Abbiamo visto che per far sì che un sito si ridimensioni a tutta pagina occorre mantenere varie celle fisse e lasciare una o più celle libere di ridimensionarsi a proprio piacimento: saranno queste celle libere a colmare lo spazio che altrimenti risulterebbe vuoto. Potrebbe sembrare particolarmente laborioso l'utilizzo delle tabelle in percentuale. A prima vista saremmo infatti portati a credere di dover utilizzare le proporzioni matematiche per ri-calcolare la percentuale delle celle di tutto il sito.

Nulla di più sbagliato. Esiste infatti un piccolo escamotage (che funziona correttamente con tutti i *browser*), che ci permette di risparmiare un sacco di lavoro.

Incominciamo a procurarci una gif trasparente di 1 *pixel* x 1 *pixel*. Si chiama *shim* ed è un'immagine vuota che nell'ordinaria costruzione di un sito ci è utile in una molteplicità di situazioni:

- si sa che, se non c'è nulla all'interno di una cella, *Netscape* 4 non la vede. È allora indispensabile riempire la cella con qualsiasi cosa: un *break* (
), un non-*breaking-space* ( ), oppure la nostra immagine vuota. (*Netscape* in origine aveva creato un tag apposito - lo <SPACER> - per risolvere il problema delle celle).
- Un altro problema che di verifica con *Netscape* 4 è che, quando ci sono delle tabelle annidate, da una cella interna non è possibile vedere lo sfondo che sta sopra. Anche in questo caso la situazione si risolve mettendo un'immagine vuota di sfondo. Così:

```
<table width="500" border="1" cellspacing="0" cellpadding="0" height="400">
  <tr>
    <td background="imgs/sfondoOmino.gif" align="center" valign="middle">
      <table width="100%" border="0" cellspacing="0" cellpadding="0">
        <tr> <td background="imgs/shim.gif"> </td>
          <td background="imgs/shim.gif"> </td>
          <td background="imgs/shim.gif"> </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

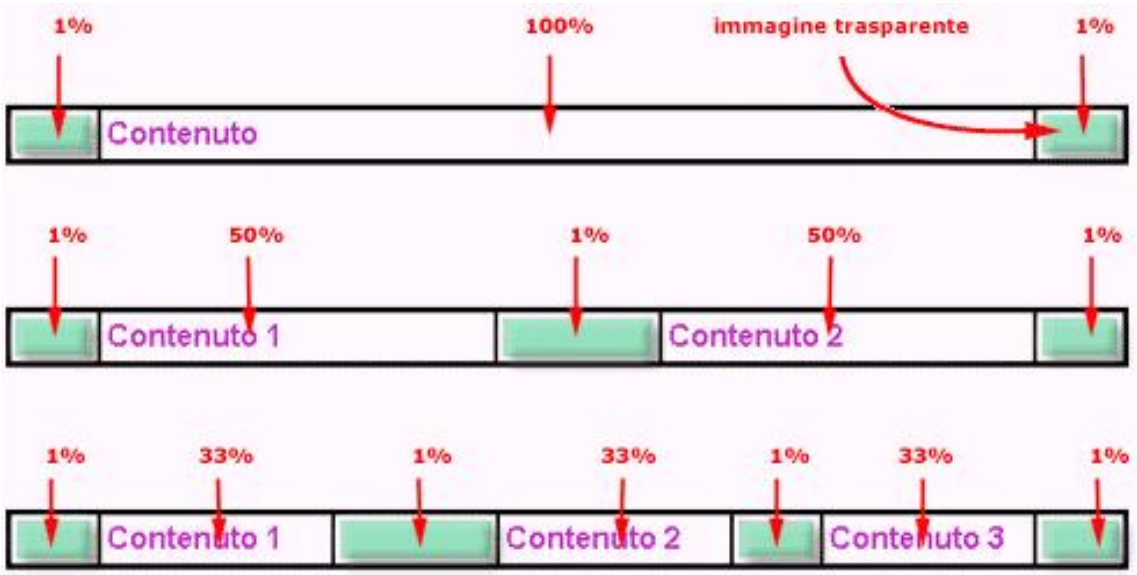
- Se non ci fosse l'immagine trasparente, con *Netscape* 4 la tabella interna avrebbe uno sfondo bianco.

La nostra immagine vuota, lo shim, ci viene in aiuto anche nel caso delle tabelle in percentuale. Per quanto possa sembrare strano, infatti, non occorre utilizzare delle percentuali realistiche, ma basta dare a una cella il valore dell'1% perché questa si schiacci sul proprio contenuto, basta darle invece un valore del 100% perché si allarghi al massimo. È evidente, a questo punto, che le dimensioni non vanno più *date* alle celle (che saranno dell'1% o del 100%), ma va attribuito al contenuto.

Si tratta dunque di prendere il nostro shim, dargli le dimensioni che preferiamo, e quindi schiacciargli contro la cella: la dimensione della cella viene così tenuta dallo *shim*.

Stesso procedimento nel caso in cui avessimo del contenuto vero contro cui schiacciare la cella (ricordarsi di usare il NOWRAP per le scritte).

La pagina non risulta appesantita dall'utilizzo di tutte queste immagini vuote: l'immagine viene infatti caricata una sola volta. Ovviamente se le celle ridimensionabili sono più di una, il 100% deve essere diviso fra tutte le celle che devono essere ridimensionate. Nella figura sottostante si trovano tre esempi.



esempi di pagine con celle ridimensionabili

Ecco il codice corrispondente al terzo esempio:

```
<table width="100%" border="2" cellspacing="0" cellpadding="3" bordercolor="#000000">
  <tr>
    <td width="1%"><img src="imgs/shim.gif" width="40" height="20"></td>
    <td width="33%">Contenuto 1</td>
    <td width="1%"><img src="imgs/shim.gif" width="80" height="20"></td>
    <td width="33%">Contenuto 2</td>
    <td width="1%"><img src="imgs/shim.gif" width="40" height="20"></td>
    <td width="33%">Contenuto 3</td>
    <td width="1%"><img src="imgs/shim.gif" width="40" height="20"></td>
  </tr>
</table>
```

Come si vede la somma totale delle celle è di 103% e le proporzioni non rispettano la realtà, ma il *layout* viene visualizzato correttamente, perché la larghezza del 100% è espressa nel tag <TABLE>, che controlla le celle sottostanti. Proprio per questo stesso motivo (che cioè le percentuali non rispecchiano più la realtà), non ci sarà da stupirsi se per avere tre celle della stessa misura e che si ridimensionano allo stesso modo, dovremo dare, ad esempio, alla prima il 60% alla seconda il 40% e alla terza il 30%.

Bibliografia

Introduzione

13.2 Strumenti di produzione

Marco Calvo, Gino Roncaglia, Fabio Ciotti, Marco A. Zela*Internet 2000;* 2000Editori Laterza

Charles J. Lyons*Progettare siti Web, dall'analisi all'implementazione;* 2001Tecniche Nuove Editore

13.3 Sviluppo di pagine e siti

Castro E.*HTML 4 per il World Wide Web;* 2000*Addison Wesley*

R. Boschin*HTML 4;* 2ed. 2000Apogeo

S. Isaacs*Inside Dynamic HTML;* 1999Mondadori informatica

Charles J. Lyons*Progettare siti Web, dall'analisi all'implementazione;* 2001Tecniche Nuove Editore

Consorzio W3C; http://www.w3.org/

Guida HTML; http://html.it/

Informazioni sui motori di ricerca; http://searchenginewatch.com/

Approfondimenti

13.4 Concetti avanzati di HTML

Castro E.*HTML 4 per il World Wide Web;* 2000*Addison Wesley*

R. Boschin*HTML 4;* 2ed. 2000Apogeo

S. Isaacs*Inside Dynamic HTML;* 1999Mondadori informatica

Charles J. Lyons*Progettare siti Web, dall'analisi all'implementazione;* 2001Tecniche Nuove Editore

Consorzio W3C; http://www.w3.org/

Guida HTML; http://html.it/

Informazioni sui motori di ricerca; http://searchenginewatch.com/

13.5 Importanza del dimensionamento delle risorse

AA VV*Microsoft Site Server 3.0 Commerce Edition;* 2001Mondadori

AA VV*Microsoft Windows 2000 Active Directory;* 2000Mondadori

Availability Management; 2000ITIMF

Capacity Management; 2000ITIMF

Service Level Management; 2000ITIMF

Prestazioni di siti Internet; http://www.aspindustry.org/

definizione di servizi;
http://www.microsoft.com/windows2000/guide/platform/performance/reports/

reports; http://www.microsoft.com/windows2000/library/technologies/terminal/

Glossario

Anchor : Sinonimo di rimando (o link o *hyperlink*).

Applet : Un piccolo programma che può essere prelevato velocemente dalla rete e usato da qualsiasi computer dotato di un *browser* capace di eseguire codice Java.

Applicazione : Un programma (software) che svolge determinate funzioni per l'utente finale. Esempi di applicazioni sono i *client* FTP, Telnet, **E-mail** e i *browser*.

Bookmark : n riferimento ad una pagina Web, di solito memorizzato in un apposito elenco mantenuto *browser*. Costituisce una sorta di segnalibro in modo da ricollegarsi velocemente ad una determinata pagina.

Browser : Un programma che consente di leggere un ipertesto. Il *browser* consente di visualizzare i contenuti dei nodi (or pages) e di navigare da un nodo all'altro.

E-mail : Abbreviazione di electronic mail (posta elettronica). È un sistema che consente la trasmissione e la ricezione su una rete informatica, di messaggi indirizzati secondo indirizzi a struttura standard. È una comunicazione di tipo asincrono perché anche in caso di indisponibilità dell'utente ricevitore, i messaggi vengono memorizzati nel *server* a cui questo utente riferisce.

Frame : (html) Un'area dello schermo, all'interno della finestra del *browser*, che visualizza una pagina Web. Sullo schermo possono così esistere più pagine contemporaneamente, ognuna in un frame diverso. Tale tecnica è consentita dalle ultime versioni di **HTML**.

HTML : (Hyper Text Markup Language) Linguaggio di realizzazione di ipertesti (interpretato), utilizzato per la realizzazione di pagine web trasmesse mediante protocollo applicativo HTTP. Una pagina **HTML** può contenere testo, immagini, brani audio e sequenze video con vari formati e trasmessi come file dal *server* al *client*.

Internet : internet (con la i minuscola) indica una qualsiasi connessione tra due reti. Internet (con la I maiuscola) è la più grande rete di calcolatori al mondo, basata sull'architettura di rete TCP/IP.

Ipertesto : Qualsiasi testo che contiene rimandi ad altri documenti; singole parole o intere frasi del documento possono essere scelte dal lettore in modo da proseguire la lettura in un altro punto del documento o di un altro documento.

Portale : Un sito web che mira ad essere un punto di ingresso per il World-Wide Web. Esso offre tipicamente un motore di ricerca, e/o link a pagine utili, notizie e altri servizi. Questi servizi sono di solito gratuiti, nella speranza che gli utenti possano utilizzare il sito come loro home page, oppure visitarlo spesso. Esempi molto noti sono Yahoo e MSN.

SGML : (Standard Generalized Markup Language) Un linguaggio di programmazione che usa dei tag per definire il formato della pagina. L'**HTML** è derivato da questo linguaggio.

Sito : (web) Qualunque computer sulla rete **Internet** che sta utilizzando un processo *server* World-Wide-Web. Un particolare sito web è identificato dall' hostname e dalla **URL**.

Tag : Una specie di comando, racchiuso tra parentesi angolari, che costituisce l'elemento caratterizzante l'**HTML** (per es.

URL : (Universal Resource Locator) Indica l'indirizzo logico di una pagina su **Internet**, specificando il nome del *server* ed il percorso nel suo file system fino alla pagina specifica. Il formato per una **URL** è

WYSIWYG : (What You See is What You Get) Modalità di visualizzazione dei documenti tipica di interfaccia grafica, per significare che quello che si vede (a video) è ciò che si otterrà (in stampa).

XML : Una iniziativa del consorzio W3C per definire un dialetto di **SGML** estremamente semplice da essere usato in World-Wide Web.

Autori

Hanno realizzato il materiale di questo modulo:

Prof. Cosimo Laneve

Professore Straordinario di Informatica presso l'Università di Bologna, dove insegna Linguaggi di Programmazione e Qualità del *Software*. Ha ricevuto il Dottorato di Ricerca in Informatica dall'Università di Pisa ed è stato *Research Associate* presso l'INRIA di *Sophia Antipolis* in Francia. Attualmente è coordinatore di progetti nazionali e internazionali che riguardano i fondamenti teorici e l'implementazione di linguaggi di programmazione distribuiti, di verifica statica di programmi e di teoria dei tipi. Relativamente a queste tematiche, ha pubblicato su numerose riviste e conferenze internazionali.