RINGRAZIAMENTI

Desidero ringraziare la Professoressa Sonia Bergamaschi, l'ing. Maurizio Vincini e l'ing. Yuri Debbi per il prezioso aiuto fornitomi durante lo svolgimento dell'attività progettuale presso la facoltà.

Un sincero ringraziamento va alla mia famiglia per il suo costante sostegno durante il percorso di studi e alla mia fidanzata Silvia per aver sopportato, e perché dovrà sopportarne, noiosi discorsi di ambito informatico.

Non potrei MAI dimenticare di ringraziare tutti i miei compagni di corso e di studio con cui ho condiviso gioie e dolori in questi tre anni di Università.

Infine vorrei ringraziare Larry Page e Sergey Brin, per aver creato Google, uno strumento che mi ha aiutato nelle innumerevoli ricerche on-line che ho svolto per fare al meglio il mio lavoro di stage.

Nicholas Paganelli

INDICE

Introduzione

Negli ultimi tempi Internet ha avuto anche nel nostro Paese uno sviluppo consistente: la diffusione dell'uso di questo importante mezzo di comunicazione in larghe fasce di popolazione è finalmente una realtà che si consolida di giorno in giorno. Eppure a volte non ci si sofferma più di tanto a considerare che a un numero troppo grande di persone, i disabili, è precluso l'utilizzo di Internet, o per meglio dire, l'accessibilità al WEB è fortemente limitata soprattutto per l'indifferenza, o l'ignoranza del problema da parte di molti webmasters. E che spesso sarebbero sufficienti solo alcuni accorgimenti per consentire ai disabili l'accessibilità cui hanno diritto.
Scopo di questa tesi, nel quale verranno proposti anche alcuni esempi tratti dal lavoro svolto presso la Facoltà, è proprio quello di fornire le informazioni sulle tecniche appositamente pensate e codificate per far sì che un sito sia interamente navigabile dai disabili. Per fare ciò si è tenuto conto delle linee-guida elaborate nel progetto WAI - Web Accessibility Initiatives - del Consorzio W3C (World Wide Web Consortium) e delle direttive dettate dalla Legge n° 4 del 09/01/2004, denominata Legge Stanca. Per tutte le informazioni relative alle leggi italiane e alle raccomandazioni dettate dal W3C si può far riferimento alla tesi di laurea “ Progetto e sviluppo dell'Offerta Didattica del Portale Web di Facoltà in conformità alla Legge Stanca ” di Caterina Barbieri con cui ho collaborato per 3 mesi di lavoro.

Verrà ora esposta la struttura secondo la quale è organizzata questa tesi:

Nel primo capitolo, dopo una breve introduzione sul linguaggio XHTML si passa ad analizzare i vantaggi che questa offre.
In particolare, si espongono le regole di base per avere pagine XHTML valide e si analizza interamente un “prototipo di documento” con alcuni esempi pratici utilizzati sul portale WEB della Facoltà di Ingegneria di Modena.
Infine descrive la procedura per l’inserimento di script e il supporto dei browser rispetto alle pagine XHTML.

Il secondo capitolo si occupa di un argomento cruciale per la realizzazione di un sito accessibile: i CSS (Cascading Style Sheets). Vengono esposte le potenzialità dei CSS, le regole che devono seguire, le tecniche che offrono per realizzare un layout di grande effetto.

Nel terzo capitolo vengono rappresentate in dettaglio le metodologie che sono state valutate per realizzare il portale della Facoltà di Ingegneria in modo bilingue e le motivazioni della scelta effettuata.

Il quarto capitolo descrive il lavoro svolto presso l’università per la realizzazione del portale accessibile e del bilinguismo. Vengono discussi i principali problemi riscontrati e le soluzioni utilizzate.

L’ultimo capitolo riporta le conclusioni ed i possibili sviluppi futuri relativi al portale di Facoltà.

Capitolo 1: XHTML

1.1 Cos'è l'XHTML

Per definire cos'è XHTML si può iniziare con una semplice espressione:

HTML + XML = XHTML

Esaminiamo i termini dell'operazione.

1.1.1 HTML

HTML (Hyper Text Markup Language) è un linguaggio di marcatura per presentare i contenuti di una pagina web. La sua semplicità è la base dell'esplosione di Internet. L'ultima raccomandazione rilasciata dal W3C è la 4.01 nel dicembre 1999.

1.1.2 XML

XML (eXtensible Markup Language) è una sorta di "super-linguaggio" che consente la creazione di nuovi linguaggi di marcatura. Potente, flessibile e rigoroso è alla base di tutte le nuove specifiche tecnologiche rilasciate dal W3C e adottate ormai come standard dall'industria informatica. I principali obiettivi di XML, dichiarati nella prima specifica ufficiale (ottobre 1998), sono pochi ed espliciti: utilizzo del linguaggio su Internet, facilità di creazione dei documenti, supporto di più applicazioni, chiarezza, comprensibilità e portabilità.

1.1.3 XHTML

XHTML è la riformulazione di HTML come applicazione XML. Ciò significa essenzialmente una cosa: un documento XHTML deve essere valido e ben formato. Si tornerà in seguito su questi concetti. Per ora è importante chiarire il punto di vista del W3C su XHTML.
Piuttosto che creare una nuova versione del linguaggio, un HTML 5.0, il W3C ha compiuto un'opera di ridefinizione delle regole sintattiche dell'ultimo linguaggio esistente, HTML 4.01, senza aggiungere nuovi tag, attributi o metodi.
Se si comprende questo fatto, sarà chiaro come XHTML risponda a due esigenze fondamentali:

1. portare HTML nella famiglia XML con i benefici che ciò comporta in termini di estensibilità e rigore sintattico.

2. mantenere la retro-compatibilità con i software che supportano HTML 4.0 o precedenti.

Possiamo dire che l' XHTML è un ponte tra passato e futuro. E' un modo per imparare a pensare in XML partendo da un linguaggio che conosciamo, senza dover rinunciare, dunque, alle conoscenze già acquisite.

1.1.4 Versioni di XHTML

Le versioni di XHTML attualmente disponibili e pubblicate come raccomandazioni dal W3C sono tre: XHTML 1.0, XHTML Basic e XHTML 1.1

XHTML 1.0 è stato pubblicato il 26 gennaio 2000 e seguita da una versione rivista dell'ottobre 2001. Consiste, come detto, nella riscrittura in XML di HTML 4.0 e si basa sulle tre DTD già definite per questo linguaggio:

- DTD Strict
- DTD Transitional
- DTD Frameset

L'**XHTML Basic** è una versione "ridotta" del linguaggio. Specificamente pensata per dispositivi mobili (PDA, cellulari), contiene solo gli elementi che si adattano a questi dispositivi (esclude, ad esempio, i frames che non hanno ovviamente senso in tale contesto). E' destinata a sostituire WML come linguaggio di base per le applicazioni WAP.

Basata sulla DTD Strict della versione 1.0, l' **XHTML 1.1** rappresenta la prima formulazione pratica del concetto di modularità. In questa visione, gli elementi fondamentali (l'insieme di tag che definiscono la struttura di un documento) sono raggruppati in una serie di moduli indipendenti, che possono essere implementati o esclusi secondo le necessità. Per il W3C è la base della futura estensione di XHTML con altri set di linguaggi o moduli, anche personalizzati.

1.2 Vantaggi di XHTML

Quando si propone una nuova tecnologia è fatale sollevare dubbi e obiezioni. Se consideriamo il quadro d'insieme degli attuali linguaggi web scopriamo che milioni di pagine sono scritte in HTML. Una gran parte di esse (circa il 98%) non è valida rispetto alle raccomandazioni del W3C, ma funziona. L'adozione di un nuovo linguaggio (XHTML) non è obbligatoria ne forzata: tutti i browser in commercio sono in grado di interpretare HTML 4.X e l'XHTML non introduce nuove features rispetto al suo predecessore.
Ma allora perché si dovrebbe riscrivere in XHTML un sito già esistente?
Ecco alcune possibili risposte ed un semplice elenco dei vantaggi di XHTML.

1.2.1 Codice pulito e ben strutturato

Il passaggio ad XHTML può essere visto come un ritorno alle origini. HTML è nato come linguaggio per definire la **struttura** di un documento che non ha nulla a che vedere con la presentazione. Eppure in tutti questi anni lo si è utilizzato per svolgere compiti per cui non è stato creato. Il caso delle tabelle è quello più eclatante.
Le tabelle sono nate per la presentazione di dati tabulari ma da sempre sono state utilizzate come l'unico mezzo per costruire il layout della pagina in modo facile e veloce.

Un altro esempio è quello del tag <font> che venne introdotto quando si sentì la necessità di gestire la tipografia dei documenti (grassetto, dimensioni, carattere). La conseguenza è quella di pagine cariche di elementi inutili, più pesanti, difficili da modificare.
La responsabilità principale per questo uso improprio di HTML non ricade sugli sviluppatori. Nel 1996 il W3C ha rilasciato la versione definitiva di CSS1, la tecnologia destinata a definire la presentazione dei documenti.
L'idea era chiara: HTML per la struttura del documento, CSS per lo stile e il layout.
Il problema fu creato indirettamente dai browser che non supportavano in modo adeguato la tecnologia dei CSS , fino all'anno 2000-2001; così facendo sono passati quattro anni, tre generazioni di browser e milioni di siti costruiti non utilizzando in modo opportuno le tecnologie disponibili.

Con XHTML, almeno nella sua versione Strict, si torna ad un linguaggio che definisce solo la struttura. Semplicemente, se inserite elementi non supportati (per esempio font, larghezza per le celle di tabelle o margini per il body) il documento non è valido, quindi si è obbligati a usare i CSS per la formattazione.
Così facendo si ha codice più pulito e più gestibile; inoltre le dimensioni delle pagine vengono ridotte visibilmente.

1.2.2 Portabilità

La portabilità rappresenta la capacità e la possibilità di un documento di essere visualizzato e implementato efficacemente su diversi sistemi: PC, PDA, cellulari WAP/GPRS, WebTV. Se si pensa alle limitazioni in termini di memoria, ampiezza dello schermo e usabilità di un terminale mobile, si capisce subito l'importanza di un linguaggio essenziale, centrato sulla struttura del documento. Ciò che ha senso è avere un titolo della pagina, un'intestazione, un paragrafo, una lista per scandire gli argomenti.
Sulla portabilità poggia l'enfasi con cui aziende del calibro di Nokia, Motorola, Ericsson o Siemens guardano ad XHTML.
Nella immagine seguente, tratta da un documento di Nokia, è chiarissimo lo schema di distribuzione delle informazioni fondato su questa interazione.

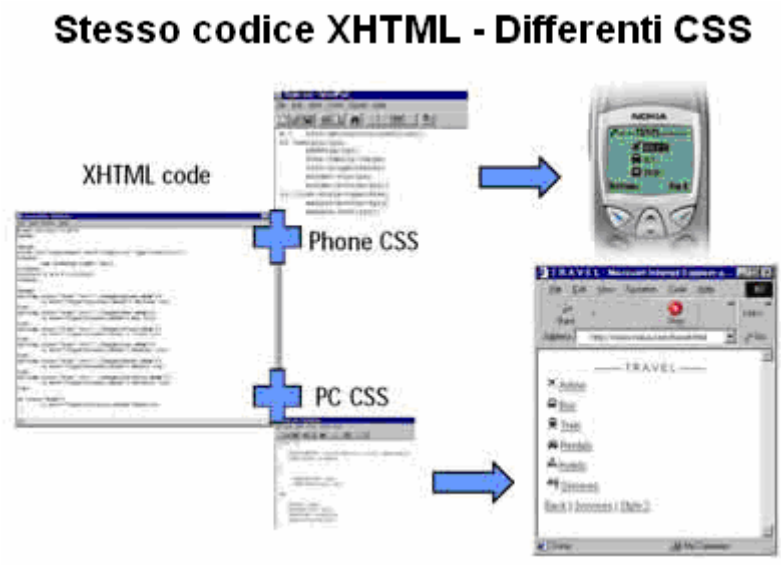


FIGURA 1

In breve:

- nella pagina XHTML incorporiamo diversi CSS per ciascun supporto
- il browser viene identificato
- su un PC vedremo il layout standard
- su un cellulare visualizziamo un layout "ridotto" e adatto alle caratteristiche del mezzo
- ciò che non cambia sono i contenuti

1.2.3 Estensibilità

Dal momento che XHTML è XML diventa estensibile. Significa che sarà facilissimo incorporare in un documento parti scritte in uno dei tanti linguaggi della famiglia XML.
Per esempio se si dovesse inserire in una pagina delle formule matematiche complesse basterà dichiarare il namespace relativo al linguaggio MathML e inserire nella pagina i tag specifici di quest'ultimo (il codice è tratto dal sito del W3C).

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
   <head>
         <title>A Math Example</title>
    </head>
    <body>
         <p>The following is MathML markup:</p>
         <math xmlns="http://www.w3.org/1998/Math/MathML">
              <apply> <log/>
                   <logbase> <cn> 3 </cn> </logbase>
                   <ci> x </ci>
              </apply>
         </math>
     </body>
</html>
```

I frutti di questo approccio, che è il più semplice dei tanti possibili, sono ancora lontani dalla maturità. L'implementazione da parte dei browser è infatti carente, ma non mancano esempi funzionanti forniti grazie ad Opera o Mozilla.

1.3 Regole di base

Entriamo nel vivo della conoscenza di XHTML esponendo le sue regole di base. Sono quelle che rendono un documento strettamente conforme alla specifiche del W3C e che ne definiscono i requisiti minimi essenziali.

1.3.1 Validazione

Un documento deve essere convalidato rispetto ad una delle tre DTD XHTML del W3C. Vedremo in un prossimo paragrafo come si effettua la convalida. Per ora è sufficiente chiarire che essa controlla essenzialmente due cose: che il documento sia valido e ben formato.

Un documento ben formattato rispetta le regole della sintassi XML: presenza di un elemento radice, un corretto annidamento degli elementi, chiusura obbligatoria dei tag vuoti, etc…
Affronteremo nei dettagli questi aspetti nei paragrafi successivi.
Un documento è valido se usa correttamente un linguaggio, vale a dire se usa nel modo giusto solo elementi specifici e consentiti.
Per XHTML (e per XML in genere, anche se in questo ambito si sta sviluppando l'uso degli schemi) le regole sono definite nelle DTD (Document Type Definition). Una DTD identifica gli elementi (tag) consentiti, il loro significato e come devono essere trattati (ad esempio, stabilisce quali sono gli attributi possibili per ciascun elemento). In un documento XHTML la DTD deve essere obbligatoriamente specificata all'inizio.

1.3.2 Elemento radice

Ogni documento XML deve contenere un elemento radice. Si tratta dell'elemento che contiene al suo interno tutti gli altri:

```
<html>
    <head>
        <title> Titolo della pagina </title>
    </head>
    <body>
        …Corpo della pagina…
    </body>
</html>
```

In un documento XHTML l'elemento radice deve essere <html>.

1.3.3 Namespace XHTML

Il namespace, in italiano spazio dei nomi, è una collezione di nomi di entità, definite dal programmatore, omogeneamente usate in uno o più file sorgente. Lo scopo dei namespace è quello di evitare confusione ed equivoci nel caso siano necessarie molte entità con nomi simili (o uguali), fornendo il modo di raggruppare i nomi per categorie: attualmente il concetto di namespace è presente esplicitamente in XML e XHTML.
Per questo motivo anche l'elemento radice <html> deve contenere la dichiarazione di un namespace XML tramite l'attributo xmlns. Il namespace usato deve essere "http://www.w3.org/1999/xhtml".
Una spiegazione più approfondita del namespace si tratterà nel paragrafo 1.4.2 .

1.3.4 Dichiarazione DOCTYPE

In un documento XHTML l'elemento radice deve essere preceduto da un elemento <!DOCTYPE>. All'interno di questo elemento è necessario specificare la DTD di riferimento e il suo URI (Uniform Resource Identifier).

1.4 Analisi di un documento XHTML

Ecco come si presenta il codice di una semplice pagina XHTML basata sulla DTD Strict:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
      La mia prima pagina XHTML
    </title>
  </head>
  <body>
    <h1>Benvenuto!</h1>
    <p>Questo è il mondo di XHTML!</p>
  </body>
</html>
```

FIGURA 2

Nell'immagine sono stati evidenziati con colori diversi le quattro sezioni essenziali di un documento XHTML:

- in rosso: il prologo
- in verde: l'elemento radice (<html>)
- in viola: la testata (<head>)
- in giallo: il corpo del documento

1.4.1 Il prologo

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Figura 3

L’immagine mostra il prologo di un documento XHTML. Esso risulta composto da due parti: la dichiarazione XML e la definizione del DOCTYPE.

1.4.1.1 Dichiarazione XML

La pagina inizia con la riga di codice: <?xml version="1.0"?>.
La sua funzione è semplice: rendere esplicito il fatto che il documento è XML. Non è obbligatoria, ma è il suo uso è consigliato dal W3C per tutti i documenti XML. Quando viene usata non deve essere preceduta da altre istruzioni.
All'interno della dichiarazione è possibile usare tre attributi.
L'unico obbligatorio è “version” (il solo valore possibile è "1.0", in quanto non esistono altre versioni del linguaggio).
Un altro attributo, opzionale, ma spesso usato è encoding. Serve a specificare la codifica del linguaggio:

```
<?xml version="1.0" encoding="UTF-8"?>
```

L'ultimo attributo possibile è standalone. Se il valore è impostato su "yes" significa che il documento non fa riferimento ad entità esterne.

Per ottenere la massima compatibilità si può omettere la dichiarazione XML in quanto molti browser hanno mostrato problemi così come alcuni editor come Dreamweaver e Frontpage.

1.4.1.2 Definizione del DOCTYPE

La dichiarazione del DOCTYPE è composta da due sezioni:

1. Un FPI (Identificatore Formale Pubblico) riferito ad una delle tre DTD XHTML
2. L'URI della DTD

Segue un esempio di dichiarazione della DOCTYPE dove in grassetto è evidenziato la FPI adottata con il relativo URI(sottolineato).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Il DOCTYPE ha lo scopo di stabilire a quale delle tre DTD XHTML intendiamo conformarci e di riferire al browser dove essa è collocata. Nell' esempio la DTD di riferimento è quella Strict, collocata sul sito del W3C.
Il DOCTYPE, inoltre, non ha alcun effetto sulla presentazione della pagina, serve solo al validatore per stabilire le regole della convalida.
Nella prima riga è definita anche la parola chiave "PUBLIC". Significa che la DTD è pubblica, creata dal W3C. In effetti, in XML, è possibile definire DTD private, specifiche per la nostra applicazione. In tal caso si usa la parola chiave "SYSTEM".

1.4.1.3 Le DTD XHTML 1.0

Giunti a questo punto dovrebbe essere ormai chiara la funzione di una DTD: descrivere in un linguaggio comprensibile da una macchina la sintassi e la grammatica di un linguaggio XML. Il tutto allo scopo di verificare la validità di un documento che a quella DTD fa riferimento.
Le DTD XHTML 1.0 sono contenute in tre documenti che si possono scaricare sul proprio pc, sia per imparare come sono fatte sia per poterle usare direttamente in un sito WEB.
In questo modo si deve modificare l'URI nella dichiarazione DOCTYPE, ad esempio:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.ing.unimo.it/DTD/xhtml1-strict.dtd">
```

Se ciò può sembrare strano ora non lo sarà forse tra qualche anno quando milioni di browser tenteranno di convalidare documenti XHTML con le DTD del W3C. E' evidente che il pericolo è quello di rallentamenti e strozzature a causa di un elevata domanda.

1.4.1.4 Il contenuto di una DTD

Il codice di una DTD è quanto di più criptico si possa immaginare, ma non mancano manuali che spiegano bene come interpretarle.

Basterà fare un esempio per rendere più leggibile il codice.
Ecco come viene definito nella DTD XHTML 1.0 Strict l'uso dell'elemento <img>:

```
<!ELEMENT img EMPTY>
<!ATTLIST img
%attrs;
src %URI; #REQUIRED
alt %Text; #REQUIRED
longdesc %URI; #IMPLIED
height %Length; #IMPLIED
width %Length; #IMPLIED
usemap %URI; #IMPLIED
ismap (ismap) #IMPLIED
>
```

L'elemento immagine inizialmente è vuoto (EMPTY).
Può assumere tutti gli attributi fondamentali (%attrs;), cioè quelli comuni alla maggior parte degli elementi (id, class, style, title, etc…).
Altri attributi possibili sono: src, alt, longdesc, width, height, usemap, ismap.
Il valore (#REQUIRED) indica che sono obbligatori ,gli altri opzionali (#IMPLIED).

Conoscere il contenuto di una DTD è dunque importante. Se non si inserisce, per esempio, l'attributo alt per un'immagine e si prova a validare la pagina, verrà segnalato l'errore e si potrà correggerlo. Ma se si conoscono le regole è certamente meglio, si eviterà di procedere per prova e correzione d'errore.
Per fortuna non è necessario imparare una DTD. Esistono per lo scopo ottime reference, elenchi di tutti i tag, degli attributi e degli eventi consentiti che spiegano in dettaglio il loro uso.

1.4.1.5 Caratteristiche principali delle DTD

Esistono 3 tipi DTD in XHTML1.0 e sono*: Strict, Transitional e Frameset.*

La DTD Strict è la più rigida, centrata esclusivamente sulla struttura del documento. Elimina diversi elementi ed esclude tutti gli attributi che definiscono la presentazione. Per questo scopo vanno usati i CSS.
Oltre agli elementi non consentiti particolare attenzione va posta ad attributi molto usati nella comune pratica del web design.
Elenchiamo alcuni casi:

- sono esclusi tutti gli attributi del tag <body> tranne quelli comuni
- non si può usare align per l'allineamento del testo in paragrafi e altri elementi
- non è supportato l'attributo target per i link e i form
- per una tabella (<table>) non si possono specificare il bordo, il colore di sfondo (bgcolor) o l'allineamento (align)
- le celle di tabella (<td>) non supportano il colore di sfondo, la larghezza (width), l'altezza (height). Supportano invece l'allineamento del testo (align)

La DTD Transitional è basata sull'omologa DTD di HTML 4.0 che è attualmente quella più usata. La spiegazione è semplice. Nelle intenzioni del W3C essa deve essere una sorta di passaggio verso una ridefinizione più rigida del linguaggio. In effetti è utile quando si vuole passare ad XHTML mantenendo il massimo grado di compatibilità con i vecchi browser. Supporta tutti gli elementi e gli attributi di presentazione di HTML 4.0, anche quelli ritenuti

sconsigliati. Si possono ancora usare le tabelle per il layout o fare uso del tag <font>,ma non è ammessa dalla legge Stanca, requisito n° 1.

Per ultima troviamo la DTD Frameset che è' identica alla Transitional, ma usata quando si utilizzano i frame. L'unica sostanziale differenza è la sostituzione del tag <body> con <frameset> nella pagina principale.

In appendice A sono elencati tutti gli elementi sia blocco che inline definiti nelle tre DTD XHTML 1.0.

1.4.2 L'elemento radice

Riprendiamo l'analisi delle quattro sezioni della nostra pagina XHTML.
Consideriamo l'elemento radice:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

FIGURA 4

Come si è detto in precedenza, l'elemento radice di un documento XHTML deve essere <html>, che deve a sua volta contenere tutti gli altri elementi.

L'elemento <html> può assumere questi attributi:

dir	Determina la direzione del testo
lang	Specifica il linguaggio di base dell'elemento quando è interpretato come HTML
xml:lang	Specifica il linguaggio di base dell'elemento quando è interpretato come XML
xmlns	Specifica il namespace predefinito per XHTML

L'unico attributo obbligatorio è **xmlns**. Il W3C, come visto, specifica anche il valore obbligatorio di tale attributo: "http://www.w3.org/1999/xhtml" .
La necessità di un namespace (spazio di nomi) dipende dalla derivazione da XML, linguaggio estensibile per definizione.
E' possibile, infatti, estendere il set di tag di XHTML con elementi di altri linguaggi, anche creati personalmente. Specificare uno o più namespace evita la possibilità di conflitti tra i tag. Chiariamo anche qui con un esempio.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//OVERFLOW//DTD XHTML-FML 1.0//EN"
"http://www.mozquito.org/dtd/xhtml-fml1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:x="http://www.mozquito.org/xhtml-fml">
    <head>
        <title> Titolo </title>
    </head>
    <body>
        <x:form> …Form… </x:form>
    </body>
</html>
```

Il codice riporta l'implementazione di FML (Form Markup Language) in un documento XHTML. FML è un linguaggio che ridefinisce e potenzia l'uso dei form. Come si può notare all'interno dell'elemento <html> sono stati definiti due namespace: in rosso è evidenziato quello standard di XHTML, in blue quello di FML.
La differenza tra i due, a parte l'URI, è nel prefisso: Il primo non ne ha mentre il secondo ha come prefisso **"x"**. Significa che i tag non preceduti da prefisso sono quelli standard di XHTML e come tali verranno intepretati.
Nel corpo della pagina è stato inserito un form ed il tag è preceduto da **x.**
Significa infatti che esso appartiene al namespace FML e come tale deve essere trattato dal browser. Senza specificare il prefisso avremmo avuto un classico form XHTML.

1.4.3 La testata del documento

Presente sin dalle prime specifiche di HTML, la testata di un documento è supportata in tutte le DTD XHTML:

```
<head>
  <title>
    La mia prima pagina XHTML
  </title>
</head>
```

FIGURA 5

La funzione principale della sezione <head> è quella di contenere informazioni che non vengono direttamente visualizzate nella pagina, ma che sono comunque di grande rilievo.

Ecco l'elenco degli elementi che possono apparire nella testata:

<base>	Usato per definire l'URL di base della pagina. Utilissimo per la risoluzione dei link relativi.
<isindex>	E' un modo per creare elementi simili alle caselle di testo. Sconsigliato.
<link>	Contiene informazioni su documenti esterni collegati. Usato soprattutto per i CSS.
<meta>	Specifica informazioni di vario tipo sul documento.
<noscript>	Usato per visualizzazioni alternative nei browser che non supportano gli script.
<object>	Racchiude un oggetto.
<script>	Contiene script di programmazione .
<style>	Definisce le regole di formattazione per il documento corrente
<title>	Specifica il titolo del documento che compare nella barra del titolo del browser

Di questi elementi l'unico richiesto nelle DTD XHTML 1.0 è <title>.
In realtà se esso viene omesso, il validatore del W3C non segnala errori. Rimane il fatto che dare un titolo ad una pagina è una buona norma da seguire per l'accessibilità ed è anche utilissimo per i motori di ricerca, che sempre più considerano questo elemento piuttosto che le classiche keywords definite nei meta tag.

1.4.4 Il corpo del documento

```
<body>
  <hl>Benvenuto!</hl>
  <p>Questo è il mondo di XHTML!</p>
</body>
```

FIGURA 6

Il corpo del documento è la sezione in cui si sviluppa il contenuto. E' racchiusa, come in HTML, tra i tag <body> e </body>. Gli elementi che possono comparire all'interno del corpo sono in genere suddivisi in due categorie: elementi blocco ed elementi inline.

1.4.4.1 Gli elementi blocco

Gli elementi blocco sono quelli che definiscono la struttura del documento. Possono contenere altri elementi blocco, elementi inline o testo. Quando sono inseriti danno origine ad una nuova riga nel flusso della pagina.
Nella strutturazione del documento è fondamentale rispettare alcune semplici regole relative alla gerarchia e all'annidamento degli elementi blocco. Il primo elemento della gerarchia dovrebbe essere , che definisce una sezione del documento. Al suo interno trovano posto gli altri elementi. La cosa importante è evitare annidamenti errati, che i browser fanno passare senza problemi, ma che il validatore segnala impietosamente in quanto violano le regole delle DTD.

Esempio:

```
<p><div> Qui inserisco il mio testo </div></p>
```

Se si inserisce questa breve riga di codice in un documento verrà visualizzato regolarmente il testo ma in realtà il documento non è valido, in quanto l'elemento <p> non può contenere altri elementi blocco.

Il corretto annodamento è :

```
<div><p> Qui inserisco il mio testo </p> </div>
```

In appendice si trova la lista degli elementi blocco supportate dalle 3 DTD XHTML 1.0.

1.4.4.2 Gli Elementi inline

Gli elementi inline si distinguono da quelli di tipo blocco per due motivi: quando sono inseriti non danno origine a una nuova riga e possono contenere solo dati (essenzialmente testo) o altri elementi inline.
Esempio:

```
<p>Questo testo è <b>grassetto</b> </p>
```

La parte delimitata dai tag e non sarà posta su una nuova riga.

Anche per gli elementi inline va posta molta attenzione all'annidamento perché sono tollerati dai browser ma non reggono al giudizio della validazione in quanto un elemento inline non può contenerne uno di tipo blocco.

1.5 XHTML E HTML a confronto

Si è già evidenziato come XHTML sia essenzialmente una ridefinizione di HTML. Al di là delle differenze tra le tre DTD, non si sono incontrati nuovi tag o elementi inusuali. Anche quelle che possono sembrare novità assolute, come la dichiarazione del DOCTYPE, erano in realtà già previste nelle precedenti versioni di HTML.
Le novità sostanziali riguardano piuttosto la sintassi, il modo in cui tag, attributi e valori vengono usati.
Verranno ora esaminate una per una, proponendo sempre il confronto con HTML.
Attenzione: l'uso di queste convenzioni è normativo. XHTML è un'applicazione XML e alla sintassi di questo deve conformarsi. Non rispettare queste regole significa non avere documenti validi e ben formati. La cosa che le accomuna è che queste regole pongono fine ad una serie di procedure che HTML consentiva e che ora sono invece vincolate ad usi ben definiti.

1.Gli elementi devono essere correttamente annidati

HTML	XHTML
un test	un test

Il primo esempio non è corretto in XHTML. Il tag si sovrappone a . La seconda colonna mostra invece un corretto annidamento degli elementi. La prima pratica è consentita in HTML; il browser dovrà interpretare qualcosa che è ambiguo, ma alla fine restituirà un testo in grassetto-corsivo. In XHTML non possono sorgere ambiguità, tutto segue una regola.

2. I nomi di elementi e attributi devono essere in minuscolo

HTML	XHTML
<TABLE><TR><TD>un test</TD></TR></TABLE>	<table><tr><td>un test</td></tr></table>

XML è un linguaggio *case sensitive*. Significa che il browser interpreterà in modo diverso le lettere maiuscole da quelle minuscole. In XHTML è consentito solo l'uso delle minuscole per i nomi di elementi e attributi.

3. Gli elementi non vuoti devono essere chiusi

HTML	XHTML
Testo1 Testo2	Testo1 Testo2

In HTML la pratica esposta a sinistra è tollerata, in XHTML non lo è. Ogni elemento non vuoto (sono quelli che contengono testo o altri elementi) deve avere dopo il tag di apertura quello di chiusura.

4. I valori degli attributi devono essere posti tra virgolette

HTML	XHTML
<img src=test.gif>	<img src="test.gif">

Anche questa pratica è tollerata in HTML. E, soprattutto, è causata da molti editor WYSIWYG(What You See Is What You Get) che velocizzano la costruzione di pagine WEB attraverso interfacce grafiche ma trascurano la sintassi degli elementi.

5. Ogni attributo deve avere un valore

HTML	XHTML
<option selected>test</option>	<option selected="selected">test</option>

Alcuni attributi di HTML non hanno un valore (*standalone*). E' il caso dell'attributo “selected” usato per identificare l'opzione di un menu a tendina selezionata all'apertura del documento. In XHTML anche questi attributi devono essere valorizzati.

6. Gli elementi vuoti devono terminare con “/>”

HTML	XHTML
  <img src="test.gif">	

In XML tutti i tag devono essere chiusi, anche gli elementi vuoti. Si tratta di quelli che non possono contenere nulla al loro interno ma semplicemente ricevere attributi: <meta>, <br>, <hr>, <img> sono i più comuni. In XHTML tali elementi vengono chiusi terminando la dichiarazione con i caratteri “/>”.

7. Uso di id e name

HTML	XHTML
<a name="test>	<a id="test"> **Per compatibilità:** **<**a id="test" name="test">

Per identificare un elemento si deve usare l'attributo id e non name.In questo modo si ha una perfetta conformità con XML dove id è l'attributo standard per l'identificazione dei frammenti. In realtà qui il cambiamento con HTML è notevole, perchè per elementi come <img> o <a> l'attributo di identificazione è proprio name. Il passaggio a id non pone problemi nei browser più recenti, ma con altri (Netscape 4, I.E. 5) non funziona. In questo caso e se la retro-compatibilità è assolutamente necessaria, la stessa specifica XHTML 1.0 suggerisce di usare entrambi gli attributi, anche se ciò è contro le regole.
Nelle specifiche successive (XHTML 1.1) l'attributo name è stato abolito completamente.

1.6 Gli scipt in XHTML

Uno degli argomenti che ha sollevato più problemi relativi a XHTML è la gestione dei linguaggi di programmazione, tradizionalmente incorporati nel documento con il tag <script>.

1.6.1 Uso di <script>

In HTML il tag <script> serve a incorporare nel documento codice di programmazione. Il linguaggio più comunemente usato è Javascript. Il tag è supportato anche in XHTML e va ugualmente inserito nella sezione tra <head> e </head>. L'elemento <script> supporta i seguenti attributi:

charset	Specifica la codifica dei caratteri
defer	Dice al browser che lo script non genera documenti
langauage	Specifica il linguaggio di scripting. E' obbligatorio quando l'attributo src non è specificato.
src	Contiene l'URL di uno script contenuto in un file esterno
type	Indica il tipo MIME dello script: es "text/javascript". E' obbligatorio.
xml:space	Usato per mantenere la spaziatura del codice

Dunque, uno script può essere direttamente incorporato nella pagina oppure inserito in un file esterno e richiamato tramite l'attributo SRC.
Vediamo due esempi:

Script incorporato

```
<script type="text/javascript" language="javascript">
      <!--
        document.open()
        document.writeln("Questo è XHTML!")
        document.close()
      //-->
</script>
```

Script collegato

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

1.6.2 Caratteri pericolosi

Il problema sorge quando nello script si utilizzano caratteri "pericolosi" (*sensitive characters* nella definizione XHTML). Si tratta di caratteri che possono generare confusione e fraintendimenti perchè il processore XML li interpreta in un determinato modo,mentre nel linguaggio di scripting hanno tutt'altro significato.
L'esempio per eccellenza è il carattere '**>**'
In Javascript '**>'** significa "**maggiore di**"; in XML indica la chiusura di un tag.

Per questo motivo W3C suggerisce due vie, ma entrambe presentano punti deboli.

Il primo metodo è l'utilizzo delle sezioni CDATA.
Nella specifica si suggerisce di racchiudere gli script all'interno di una sezione CDATA. Esse vengono usate in documenti XML per racchiudere blocchi di testo contenenti caratteri che potrebbero essere interpretati come elementi di marcatura.
Una sezione CDATA inizia con la stringa "<![CDATA[e termina con]]>".
Ecco come lo script visto in precedenza apparirebbe:

```
<script type="text/javascript" language="javascript">
    <![CDATA[
                document.open()
                document.writeln("Questo è XHTML!")
                document.close()
    ]]>
</script>
```

Il punto debole di questo approccio è che i browser non gestiscono affatto bene le sezioni CDATA.
E qui arriviamo al secondo metodo.
Si potrebbe ovviare scrivendo gli script in un file esterno e collegandolo al documento. Ma se bisogna modificare lo script, magari con ASP o comunque con variazioni lato server, non è possibile farlo.

In effetti, se la situazione è quella appena prospettata, non rimane che usare il vecchio metodo. Si avrà un documento non conforme al 100% ma non a causa del programmatore, in attesa che XHTML sia in grado di incorporare i linguaggi di scripting diversamente.

1.7 Compatibilità

La specifica XHTML 1.0 contiene una serie di linee guida per mantenere la compatibilità con i browser esistenti. Riguardano aspetti molto importanti e sono da tenere sempre presenti quando la retro-compatibilità è fondamentale.
Di seguito sono elencati quelli principali, dal momento che ad alcune di esse si è già fatto cenno nel paragrafo 1.5 .

1. Elementi vuoti
Quando si usano elementi vuoti è opportuno, chiudendoli, lasciare uno spazio prima del carattere '/'. Usando la sintassi standard (<br/>) si hanno problemi con alcuni browser. Dunque usate sempre la forma: <br />, <img... />, <hr />

2. Contenuto degli elementi vuoti
Capita frequentemente di lasciare vuoti elementi che richiedono un contenuto. Spesso, ad esempio, si usano paragrafi vuoti per creare spazio nel documento. In tal caso è preferibile usare la forma completa di chiusura e non quella minimizzata. Useremo quindi e non .

3. Codifica dei caratteri
Se si usa la codifica dei caratteri usare sempre tale dichiarazione sia nella dichiarazione XML sia in un meta tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<meta http-equiv="Content-type" content='text/html; charset="UTF-8"'
/>
```

4. Uso del carattere '&'
Se si fa uso della e commerciale (**&**) in valori di attributi è obligatorio esprimerlo con un riferimento ad una entità di carattere: "**&"**. Ad esempio, in un link di questo tipo:

```
<a href="http://www.miodominio.it/indice.asp?nome=nicholas&cognome=paganelli">
```

si scriverà:

```
<a href="http://www.miodominio.it/indice.asp?nome=nicholas&amp;cognome=paganelli">
```

5. CSS
Quando si collegano CCS esterni è opportuno che questi utilizzino le minuscole.

1.8 Supporto dei browser

La questione del supporto di XHTML da parte dei browser è delle più interessanti. Nel corso dei precedenti paragrafi si è più volte accennato all'argomento.
Di seguito verranno raccolte le idee e fissati i punti chiave del problema.

1.8.1Compatibilità con il passato

XHTML 1.0 è un linguaggio che garantisce la compatibilità con il passato. Basta salvare il documento con estensione ".html".
Tutti i browser saranno in grado di rendere il documento visibile, soprattutto se si rispettano alcune semplici regole.
Ne ricordiamo due che risolvono problemi con Netscape 4 e con IE 4 e 4.5 per Mac:

1. lasciare uno spazio prima della slash nei tag vuoti: <br /> e non
2. omettere la dichiarazione XML

1.8.2 Compatibilità con il futuro

Uno dei concetti su cui tante volte si è insistito è quello di documento valido e ben formato. I browser recenti (quelli di quinta e sesta generazione) sono tutti in grado di gestire, più o meno bene, il formato XML. Ognuno di essi ha un parser XML, un processore che analizza il documento. I parser XML sono di due tipi:

- **processori non validanti**: verificano soltanto che il documento sia ben formato.
- **processori validanti**: oltre alla correttezza formale devono verificare che il documento sia conforme alla DTD di riferimento.

Quelli implementati nei principali browser sono parser del tipo **non validante.**

I browser, sia validanti che non, se trovano una formattazione non corretta della pagina interrompono la visualizzazione e segnalano l'errore. Questo perché è esplicitamente richiesto nella specifica XML: quando incontra un errore un processore XML deve interrompere l'elaborazione del documento.

Capitolo 2: I CSS

2.1 Cosa sono i CSS

Dietro l' acronimo CSS (Cascading Style Sheets - Fogli di stile a cascata) si nasconde uno dei fondamentali linguaggi standard del W3C. La sua storia percorre binari paralleli rispetto a quelli di HTML, di cui vuole essere l'ideale complemento. Da sempre infatti, nelle intenzioni degli uomini del W3C, HTML e XHTML dovrebbero essere semplicemente linguaggi strutturali, alieni da qualunque scopo attinente la presentazione di un documento. Per questo obiettivo, ovvero arricchire l'aspetto visuale ed estetico di una pagina, lo strumento designato sono i CSS. L'ideale perseguito da anni si può sintetizzare con una nota espressione: separare il contenuto dalla presentazione.

La prima specifica ufficiale di CSS (CSS1) risale al dicembre 1996. Nel maggio 1998 è stata la volta della seconda versione: CSS2. Niente stravolgimenti, ma molte aggiunte rispetto alla prima. CSS2 non è altro che CSS1 più alcune nuove proprietà, valori di proprietà e definizioni per stili non canonici come quelli rivolti alla stampa o alla definizione di contenuti audio. E' attualmente allo stato di Working Draft (bozza) la nuova specifica CSS3.

A cosa servono

I CSS servono a realizzare tutte quelle funzioni per cui l'XHTML non è stato progettato.
Finalmente, ad esempio, si può dare al testo delle pagine un aspetto da word-processor: non solo con il colore o i font ma con un sistema di interlinea pratico e funzionale, con le decorazioni che si desiderano, riuscendo a spaziare lettere e parole, impostando stili diversi per titoli e paragrafi, sfruttando i benefici dell' indentatura o della giustificazione.
Si possono finalmente distanziare gli elementi della pagina in maniera semplice e intuitiva con un potente meccanismo di gestione dei margini. In questo modo tutti quegli escamotage come le immagini trasparenti usate per formattare la pagina possono essere finalmente cancellati.
Gli effetti di sfondo si possono applicare anche solo ad una parte del documento: infatti si può ripetere in una sola direzione, in due o per niente ed associare ad ogni contenitore uno sfondo differente.

Ma la funzionalità vincente dei CSS è la gestione separata della presentazione del documento.
Infatti se si vuole modificare lo sfondo ad un sito di molte pagine basterà cambiare il file richiamato nel foglio di stile e non cambiarlo in ogni singola pagina. Il risultato sono pagine più leggere e facili da modificare. E milioni di byte di banda risparmiati durante la trasmissione client-server.

2.2 Inserire i fogli di stile in un documento

Se CSS è un solo linguaggio, vari sono i modi per inserire i fogli di stile in un documento. Per capire il meccanismo è necessario chiarire la fondamentale distinzione tra fogli esterni e interni.

2.2.1 CSS esterni e interni

E' esterno un foglio di stile definito in un file separato dal documento. Si tratta di semplici documenti di testo editabili anche con il Blocco Note o Kwrite ai quali si assegna l'estensione ".css".
Un foglio di stile si dice invece interno quando il suo codice è compreso in quello del documento.

A seconda che si lavori con un CSS esterno o interno variano sintassi e modalità di inserimento. Rispetto a queste diverse modalità si parla di fogli di stile collegati o in linea.

2.2.2 Fogli collegati

Per caricare un foglio esterno in un documento esistono due possibilità. La prima e più compatibile è quella che fa uso dell'elemento <link>. La dichiarazione va sempre collocata all'interno della sezione <head> del documento XHTML:

```
<head>
        <link rel="stylesheet" type="text/css" href="stile.css" media=”screen” />
</head>
```

L'elemento <link> presenta una serie di attributi di cui è importante spiegare significato e funzione:

1. **rel**: Obbligatorio. Descrive il tipo di relazione tra il documento e il file collegato. Per i CSS2 sono i valori possibili: "stylesheet" e "alternate stylesheet".

2. **type**: Obbligatorio. Identifica il tipo di dati da collegare. Per i CSS l'unico valore possibile è "text/css".

3. **href**: Obbligatorio. Serve a definire l'URL assoluto o relativo del foglio di stile.

4. **media**: Opzionale. Con questo attributo si identifica il supporto (schermo, stampa, casse) cui applicare un particolare foglio di stile.

2.2.3 Fogli in linea

Un secondo modo per formattare un elemento con un foglio di stile consiste nell'uso dell'attributo <style>. Esso fa parte della collezione di attributi XHTML definita Common: si tratta di quegli attributi applicabili a tutti gli elementi. La dichiarazione avviene a livello dei singoli tag contenuti nella pagina e per questo si parla di fogli di stile in linea. La sintassi generica è la seguente:

```
<elemento style="regole_di_stile">
```

Se, ad esempio, si vuole formattare un titolo in H2 e in modo che abbia il testo di colore blu e lo sfondo giallo, scriveremo:

```
<h2 style="color: blue ; background: yellow;"> TITOLO COLORATO </h2>
```

Come valore di style si possono dichiarare più regole di stile. Esse vanno separate dal punto e virgola. I due punti si usano invece per introdurre il valore della proprietà da impostare.

2.2.4 Quale usare?

Il punto di partenza nella risposta deve essere questo: i risultati nella formattazione del documento non cambiano. La giusta soluzione sarà quindi quella richiesta dalla applicazione in cui verrà inserito. Se gli elementi da modificare sono presenti solo in certe pagine basta usare uno stile in linea, altrimenti, se le pagine fossero molteplici, tutto diventerebbe più complicato perché andrebbero modificate tutte una ad una. Invece basta fare un nuovo CSS esterno e collegarlo al documento.

L'uso estensivo di fogli in linea rischia di compromettere uno dei principali vantaggi dei CSS, ovvero avere pagine più leggere e facili da gestire. Intervenire all'interno di una pagina per andare a modificare uno stile e ripetere l'operazione per quante sono le pagine del sito può diventare frustrante e ripetitivo. Del resto, il loro uso è ultimamente deprecato anche dal W3C.

2.3 Come è fatto un CSS

Quanto visto finora riguarda essenzialmente il rapporto tra CSS e XHTML: tutti gli elementi, gli attributi e le funzionalità analizzate fanno parte della specifica del secondo linguaggio.

Un foglio di stile non è altro che un insieme di regole, accompagnate, volendo, da qualche nota di commento.

2.3.1 Le regole

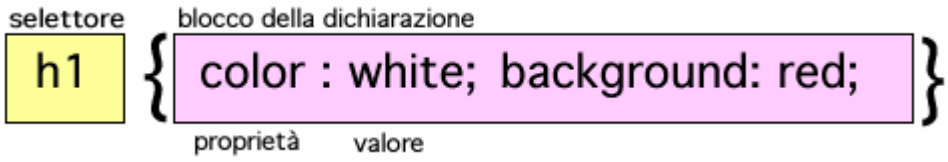


FIGURA 7

L'illustrazione mostra la tipica struttura di una regola CSS. Essa è composta da due blocchi principali:

- il selettore
- il blocco delle dichiarazioni

Il **selettore** serve a definire la parte del documento a cui viene applicata la regola. In questo caso verranno formattati tutti gli elementi <h1>: lo sfondo sarà rosso, il colore del testo bianco. I selettori possono essere diversi e a queste varie tipologie si parlerà in un prossimo paragrafo.

Il **blocco delle dichiarazioni** è delimitato rispetto al selettore e alle altre regole da due parentesi graffe. Al suo interno possono trovare posto più dichiarazioni. Esse sono sempre composte da una coppia:

- Proprietà
- Valore

La proprietà definisce un aspetto dell'elemento da modificare (margini, colore di sfondo, etc…) secondo il valore espresso. Proprietà e valore devono essere separati dai due punti. Una limitazione fondamentale da rispettare è questa: per ogni dichiarazione non è possibile indicare più di una proprietà, mentre è spesso possibile specificare più valori.
Questa regola è pertanto errata:

```
body {color background: black;}
```

Mentre questa è perfettamente valida e plausibile:

```
p {font: 12px Verdana, Arial;}
```

E’ da ricordare che gli spazi bianchi lasciati all'interno di una regola non influiscono sul risultato;il consiglio, anzi, è di lasciare sempre uno spazio tra le varie parti per una migliore leggibilità.

2.3.2 I Commenti

Per inserire parti di commento in un CSS bisogna racchiudetelo tra questi segni:

- /* come segno di apertura
- */ come segno di chiusura

Esempio:

```
p {text-align: center}    /* centra il testo di <p>*/
```

2.3.3 I selettori

La parte preponderante della specifica CSS2 è dedicata all'analisi delle diverse proprietà in grado di definire l'aspetto visuale di elementi e sezioni di una pagina. Prima di tutto, però, è fondamentale capire come e a cosa queste proprietà possono essere assegnate.
Fondamentalmente una regola CSS viene applicata ad un selettore. La parola parla da sé: si tratta di una semplice dichiarazione che serve a selezionare la parte o le parti di un documento soggette ad una specifica regola.

2.3.3.1 Selettori generici

Quella che segue è una lista commentata dei vari tipi di selettore (almeno dei più utilizzati)

I. Selettore di elementi

È il più semplice dei selettori. È costituito da uno qualunque degli elementi di XHTML.

```
h1 {color: #000000;}                /*h1 e p sono selettori*/
p {background: white; font: 12px Verdana, Arial, sans-serif;}
```

II. Selettore di raggruppamento

È possibile nei CSS raggruppare diversi elementi al fine di semplificare il codice. Gli elementi raggruppati vanno separati da una virgola.

Il raggruppamento è un'operazione molto conveniente. Pensate a questo scenario:

```
h1 {background: white;}
h2 {background: white;}
h3 {background: white;}
```

Tutti e tre gli elementi hanno uno sfondo bianco. Invece di scrivere tre regole separate si può fare così:

```
h1, h2, h3 {background: white;}
```

III. Selettore universale (universal selector)

Anche nei CSS esiste un carattere jolly. Il selettore universale serve a selezionare tutti gli elementi di un documento. Si esprime con il carattere * (asterisco).

```
* { color: black; }      /*tutti gli elementi avranno il colore del testo nero */
```

2.3.3.2 ID e classi

I CSS non sarebbero uno strumento così potente senza questi tipi di selettori. Classi e ID sono davvero una delle chiavi per sfruttare al meglio questo linguaggio.

Il valore dell'attributo class o id deve trovare una corrispondenza in un foglio di stile esterno.

Per esempio nella pagina XHTML verrà richiamata la classe:

```
<div class=” testocentrato ” >
```

Mentre nel foglio di stile verranno settati i valori che si desiderano che la classe acquisti:

```
.testocentrato { text-align: center; color: blue;}
```

Lo stesso meccanismo è valido per i selettori di tipo ID.
In un documento XHTML l'attributo id è usato per identificare in modo univoco un elemento.
Se viene assegnato ad un paragrafo un identificatore, non si potrà più usare questo valore nel resto della pagina. Di conseguenza, l’identificatore dichiarato nel CSS trasformerà solo quel paragrafo specifico. Una singola classe, al contrario, può essere assegnata a più elementi, anche dello stesso tipo.

Chiariti i concetti di base, passiamo ad analizzare usi e sintassi di ognuno.

Le Classi

Per definire una classe si usa far precedere il nome da un semplice punto:

```
.nome_della_classe { proprietà: valore; }
```

Questa è la sintassi di base. Un selettore classe così definito può essere applicato a tutti gli elementi di un documento XHTML.

Esiste un secondo tipo di sintassi:

```
<elemento>.nome_della_classe { proprietà: valore; }
```

Esso è più restrittivo rispetto alla sintassi generica. Se infatti si definisce questa regola:

```
p.testoblu {color: blue;}
```

lo stile verrà applicato solo ai paragrafi che presentino l'attributo class="testoblu". Il secondo tipo di sintassi va usato solo se si applica una classe ad uno specifico tipo di elemento (solo paragrafi o solo div, e così via). Se invece si ritenete di doverla applicare a tipi diversi bisogna usate la sintassi generica.

Gli ID

La sintassi di un selettore ID è semplicissima. Basta far precedere il nome dal simbolo cancelletto :

```
#nome_ID { proprietà: valore; }
```

Con questa regola:

```
#testoblu {color: blue;}
```

si assegna il colore blu all'elemento che presenti una definizione come questa:

```
<h1 id=" testoblu " > TITOLO </h1>
```

Come per le classi è possibile usare una sintassi con elemento:

```
p#nome_ID { proprietà: valore; }
```

In realtà questa modalità è assolutamente superflua. Se l'id è univoco non si avrà alcun bisogno di distinguere l'elemento cui verrà applicata. Inoltre: la sintassi generica si rivela più razionale e utile. Se si dichiara un ID semplice è possibile assegnarlo a qualunque tipo di elemento mentre con quest'ultima sintassi si è vincolati ad un elemento.

2.3.3.3 Le pseudo-classi e gli pseudo-elementi

Le pseudo-classe e gli pseudo-elementi non definiscono un elemento ma un particolare stato di quest'ultimo. In buona sostanza imposta uno stile per un elemento al verificarsi di certe condizioni.

A livello sintattico le pseuso-classi non possono essere mai dichiarate da sole, ma per la loro stessa natura devono sempre appoggiarsi ad un selettore. Il primo costrutto che esaminiamo è quello con un elemento semplice:

```
a:link {color: blue;}
```

La regola esplicita che i collegamenti ipertestuali (<a>) che non siano stati visitati (:link) avranno il colore blue. Da qui risulta più chiaro il concetto espresso all'inizio: la pseudo-classe ":link" definisce lo stile (colore blue) solo in una determinata situazione, ovvero quando il link non è stato attivato. Appena ciò dovesse avvenire, il testo non sarà più blu, perchè la condizione originaria è venuta meno.

La sintassi della pseudo-classe inizia con i due punti, segue senza spazi l'elemento e subito dopo si crea nel modo consueto il blocco delle dichiarazioni.

Una pseudo-classe può anche essere associata a selettori di tipo classe. I costrutti possibili sono due. Il primo è quello sancito nella specifica CSS1. La pseudo-classe doveva seguire la dichiarazione della classe in questo modo:

```
<elemento>.classe:pseudo-classe {proprietà: valore;}
```

A partire dalla specifica CSS2 è consentita anche questa sintassi:

```
<elemento:pseudo-classe>.classe {proprietà: valore;}
```

in cui la classe segue la pseudo-classe. Significato e risultati sono comunque identici al primo esempio. Il primo tipo di sintassi garantisce una maggiore compatibilità con i browser più datati.

2.3.4 Le @-rules

Le @-rules rappresentano vie alternative, spesso più flessibili e potenti, per realizzare funzioni attuabili in altri modi. A livello sintattico le @-rules possono essere definite nel corpo del documento, per l'esattezza all'interno dell'elemento <style> o direttamente nel codice di un CSS esterno.

Ognuna delle diverse @-rules presenta poi scopi e criteri diversi di costruzione.
Di seguito vengono elencate e descritte quelle più importanti.

I. @media

Grazie ad esso siamo in grado di impostare un foglio di stile per ogni supporto su cui la nostra pagina verrà distribuita. Una possibilità davvero interessante e che andrà sempre più diffondendosi con l'ampliarsi dei mezzi di diffusione delle pagine XHTML.
I valori ammessi per @media sono:

- all. Il CSS si applica a tutti i dispositivi di visualizzazione.
- screen. Valore usato per la resa sui normali browser web.
- print. Il CSS viene applicato in fase di stampa del documento.
- projection. Usato per presentazioni e proiezioni a tutto schermo.
- aural. Da usare per dispositivi come browser a sintesi vocale.
- braille. Il CSS viene usato per supporti basati sull'uso del braille.
- embossed. Per stampanti braille.
- handheld. Palmari e simili.
- tty. Dispositivi a carattere fisso.
- tv. Web-tv.

La sintassi generica è questa:

```
@media <valore> {regole CSS}
```

@media va quindi seguito dal nome di uno dei supporti scelti come target specifico e dalle regole di stile racchiuse tra parentesi graffe.

II. @import

@import viene usata per collegare un foglio di stile esterno al documento. La sintassi generica è la seguente:

```
<style type="text/css">
        @import url(foglio_di_stileA.css);
        @import url(foglio_di_stileB.css);
</style>
```

Come si vede la direttiva è accompagnata dall'indicazione “url” che precede l'indirizzo del CSS. Questo è racchiuso tra parentesi tonde. È possibile importare all'interno di un singolo tag <style> più fogli di stile,basta separarli da un punto e virgola.
Le regole di tutti i CSS collegati in questo modo saranno applicate al documento secondo l'ordine e i criteri stabiliti dalle norme sull'importanza e la specificità che vedremo al paragrafo 2.4.
Un uso interessante di @import è che può essere usata anche all'interno di un foglio di stile per incorporare un altro CSS esterno:

```
@import url(foglio2.css) screen, handheld;
```

Supponendo che il codice sia dentro un file CSS, avremo il risultato di incorporare al suo interno il contenuto del secondo foglio di stile (foglio2.css). È possibile definire all'interno della direttiva @import anche il supporto cui applicare il CSS, in modo simile a quanto abbiamo visto a proposito dell'attributo media.

III. @charset

La direttiva @charset serve a specificare l'impostazione relativa alla codifica dei caratteri di un documento. Svolge la stessa funzione che in XHTML è possibile ottenere con l'uso dei meta-tag:

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

La sintassi è semplicissima:

```
@charset "iso-8859-1"
```

La direttiva non può essere usata in fogli incorporati. Quando si vuole usarla in un foglio esterno essa deve essere la prima dichiarazione del CSS per rendere valido quest’ultimo.

IV. @font-face

La direttiva @font-face può essere usata per descrivere un font usato nel documento. In genere potrebbe essere usata per specificare l'url di un font particolare da scaricare su una macchina che non lo veda presente tra i suoi font.
La sintassi classica è questa:

```
@font-face {font-family: miofont; src: url("http://www.mioserver.it/fonts/miofont.tt");}
```

Il supporto non è garantito e l'utilità effettiva poco più che nulla.

2.3.5 Valori e unità di misura

In questa paragrafo si trattatano i tipi di valore e le unità di misura con cui è possibile impostare le tante proprietà dei CSS. Prima di tutto, però, è opportuno chiarire due fondamentali regole di base.

1. I valori di una proprietà non vanno mai inclusi tra virgolette. Uniche eccezioni i valori espressi da stringhe di testo e i nomi dei font formati da più di una parola (esempio: "Times New Roman").

2. Quando si usano valori numerici con unità di misura, non bisogna lasciare spazio tra numero e sigla dell'unità. E' corretto quindi scrivere 15px oppure 5em. E' invece sbagliato usare 15 px o 5 em. In questi casi la regola sarà ignorata o mal interpretata.

Unità per le dimensioni
Questa è la lista delle unità di misura usate per definire dimensioni, spazi o distanze. Nella pratica comune solo alcune di esse sono realmente usate. Verranno elencate comunque tutte per completezza.

- in (inches - pollici): classica misura del sistema metrico americano. Praticamente nullo il suo valore su un supporto come un browser web viste le variabili relative a risoluzione e ampiezza dei monitor.
- cm (centimetri): classica misura del sistema metrico, la difficoltà maggiore sta nella resa su monitor, che è altra cosa rispetto alla carta stampata.
- mm (millimetri): come centimetri.
- pt (points - punti): unità di misura tipografica destinata essenzialmente a definire la dimensione dei font.
- pc (picas): unità poco usata. 1pica equivale a 12 punti.
- em (em-height): unità di misura spesso usata dagli autori CSS. 1em equivale all'altezza media di un carattere per un dato font. E' un unità di misura relativa.
- ex (ex-height): poco usata. 1ex equivale all'altezza del carattere ‘x’ minuscolo del font scelto.
- px (pixels): unità di misura ideale su monitor. E' quella più usata e facile da comprendere.
- % (percentuale):Un valore espresso in percentuale è da considerare sempre relativo rispetto ad un altro valore, in genere quello espresso per l'elemento parente. Si esprime con un valore numerico seguito (senza spazi) dal segno di percentuale (%).

Stringhe
Alcune proprietà dei CSS possono avere come valore una stringa di testo da inserire come contenuto aggiunto nel documento. I valori espressi da stringhe vanno sempre racchiusi tra virgolette. Le proprietà in questione sono tre: content, quotes, text-align (ma solo per le tabelle definite con i CSS).

Valori URI
Si tratta di URL che puntano a documenti esterni (in genere immagini, come negli sfondi). Possono essere URL assoluti o relativi. In questo caso il path fa sempre riferimento alla posizione del foglio di stile e non del documento XHTML. La definizione dell'indirizzo è

sempre introdotta dalla parola chiave url e va inserita tra parentesi tonde senza virgolette. Esempio:

```
@import url(immagini/sfondo.gif)
```

2.4 Ereditarietà, cascade e conflitti tra stili

I. Ereditarietà

Secondo questo meccanismo le impostazioni stilistiche applicate ad un elemento ricadono anche sui suoi discendenti. Almeno fino a quando, per un elemento discendente, non si imposti esplicitamente un valore diverso per quella proprietà.
Non tutte le proprietà sono ereditate come si vedrà durante l'analisi di ciascuna di esse. In genere sono quelle attinenti la formattazione del box model: margini, bordi, padding e background le più importanti. Il motivo è semplice. Ereditare un bordo è senza senso. Se ne viene impostato uno per un paragrafo <p> sarebbe assurdo che un elemento <a> o un testo in grassetto vengano circondati dallo stesso bordo.
L'ereditarietà però non basta a spiegare le molteplici possibilità di relazione tra le regole di un CSS. Per questo è stato introdotto il concetto di peso.

II. Peso e origine

Verranno affrontati ora un'altra serie di concetti fondamentali tutti riconducibili comunque ad uno stesso ambito: i conflitti possibili tra gli stili e le regole.

Il primo concetto da apprendere è quello di peso. Si riferisce alla maggiore o minore importanza da assegnare a ciascuna regola. Un primo criterio di importanza è dato dall'origine del foglio di stile. Quando visualizziamo una pagina XHTML possono entrare in gioco nel modificare lo stile degli elementi tre diversi fogli di stile in questo ordine:

1. foglio dell'autore
2. foglio dell'utente
3. foglio predefinito del browser

Tutti i software di navigazione di ultima generazione consentono una gestione di questo aspetto. E' possibile, ad esempio, far sì che il browser ignori i CSS definiti dall'autore delle pagine e formattare queste ultime con un CSS realizzato dall'utente. E ancora è possibile modificare questa gerarchia con l'uso della parola chiave "!important".

III. Specificità

La specificità, come il nome può suggerire, descrive il peso relativo delle varie regole all'interno di un foglio di stile. Esistono regole ben precise per calcolarla e sono quelle che applica lo user agent di un browser quando si trova davanti ad un CSS.

I fattori del calcolo sono tre e ciascuno di essi rappresenta il valore di una tripletta.
Per prima cosa si conta il numero di selettori ID presenti nella regola. Si passa quindi a verificare la presenza di classi e pseudo-classi. Infine si conta il numero di elementi definiti nella regola.
Mai come in questo caso urge l'esempio. Prima regola:

```
#titolo {color: black;}
```

Calcolo: un ID, 0 classi, 0 elementi. Tripletta dei valori: 1-0-0

```
.classe1 {background: #C00;}
```

Calcolo: 0 ID, 1 classe, 0 elementi. Tripletta: 0-1-0

```
h1 {color: red;}
```

Calcolo: 0 ID, 0 classi, un elemento. Tripletta: 0-0-1

Il peso specifico della prima regola è il maggiore mentre quello dell'ultima è il minore.
In pratica si deduce che gli ID pesano più delle classi che pesano più dei singoli elementi.

IV. Il concetto di cascade

Si capisce già dall'acronimo del CSS (Cascading Style Sheets) che il concetto e il meccanismo di cascade è un elemento chiave.
Per rendere più comprensibile tale concetto verrà descritto il procedimento di un browser quando incontra un foglio di stile e lo visualizza sul monitor del computer.

1. Per prima cosa viene controllato il target stabilito con l'attributo media o con dichiarazioni equivalenti. Lo user agent scarta quindi tutti gli stili riferiti alla stampa o ad altri supporti. Allo stesso tempo vengono scartate tutte le regole che non trovino corrispondenza negli elementi strutturali del documento.

2. Comincia ad ordinare per peso e origine secondo le regole viste sopra. Se è presente un CSS definito dall'autore verrà usato quello. Altrimenti verificherà la presenza di un foglio di stile utente e in sua assenza applicherà le sue regole stilistiche predefinite.

3. Quindi calcola la specificità dei selettori e in caso di conflitto tra regole usa questo criterio di prevalenza.

4. Se non ci sono conflitti o se peso, origine e specificità coincidono, viene applicata la regola più vicina all'elemento nel codice del documento. L'ordine, se le dichiarazioni degli stili sono fatte nell'ordine più corretto e logico, è quindi il seguente: gli stili in linea prevalgono su quelli collegati.

V. Importanza

Ed ora il concetto di importanza. Semplice e lineare la regola: se una dichiarazione viene accompagnata dalla parola chiave "!important" essa balza al primo posto nell'ordine di applicazione a prescindere da peso, origine, specificità e ordine.

2.5 Il Layout

2.5.1 Il Box Model

Di seguito verrà spiegato il meccanismo che governa la presentazione dei vari elementi di una pagina. Nel capitolo 2 si è detto che le pagine XHTML non sono altro che un insieme di box rettangolari, indifferentemente che si tratti di elementi blocco o di elementi inline. Tutto l'insieme di regole che gestisce l'aspetto visuale degli elementi blocco viene in genere riferito al cosiddetto box model.
Ogni box comprende un certo numero di componenti di base, ciascuno modificabile con proprietà dei CSS. La figura qui sotto mostra visivamente tali componenti:

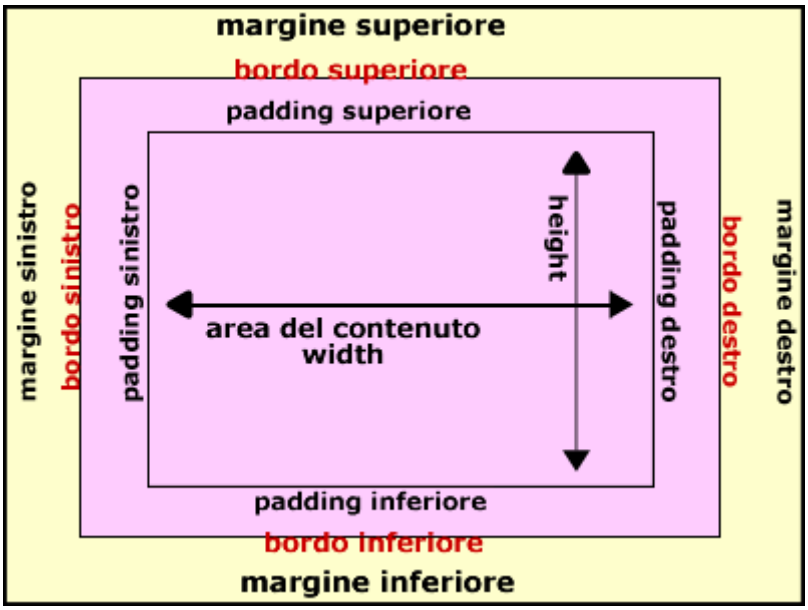


FIGURA 8

Partendo dall'interno abbiamo:

- **l'area del contenuto**. E' la zona in cui trova spazio il contenuto vero e proprio, testo, immagini, animazioni Flash. Le dimensioni orizzontali dell'area possono essere modificate con la proprietà width. Quelle verticali con height.
- **il padding**. E' uno spazio vuoto che può essere creato tra l'area del contenuto e il bordo dell'elemento. Come si vede dalla figura, se si imposta un colore di sfondo(viola) per un elemento questo si estende dall'area del contenuto alla zona di padding.
- **il bordo**. E' una linea di dimensione, stile e colore variabile che circonda la zona del padding e l'area del contenuto.
- **il margine**. E' uno spazio di dimensioni variabili che separa un dato elemento da quelli adiacenti.

Queste nozioni non sono state introdotte con i CSS, ma fanno parte del normale meccanismo di rendering di un documento. Quando realizziamo una pagina XHTML senza fogli di stile è il browser ad applicare per alcune di queste proprietà le sue impostazioni predefinite.

La novità è che con i CSS si può controllare con precisione tutti questi aspetti.

Il box model è governato da una serie di regole di base concernenti la definizione di un box e il suo rapporto con gli altri elementi.

1. Larghezza del box
Bisogna distinguere tra la larghezza dell'area del contenuto e la larghezza effettiva di un box . La prima è data dal valore della proprietà width. La seconda è data da questa somma:

“Margine Sinistro” + “Bordo Sinistro” + “Padding Sinistro” +
“Area del contenuto” + “Padding Destro + “Bordo Destro + Margine Destro”

Come si vede infatti nella figura X margini, padding e bordi devono considerarsi a tutti gli effetti parte dell'area complessiva dell'elemento.

2. Larghezza ed elemento contenitore
Se non si imposta alcun valore per la proprietà width o se il valore usato è “auto” la larghezza di un box è uguale a quella dell'area del contenuto dell'elemento contenitore.

3. Uso del valore auto
Solo per tre proprietà è possibile impostare il valore “auto”: margini, altezza e larghezza.
L'effetto è quello di lasciar calcolare al browser l'ammontare di ciascuna di esse in base alla risoluzione dello schermo e alle dimensioni della finestra.

4. Margini verticali e orizzontali tra gli elementi
Per due box adiacenti in senso verticale che abbiano impostato un margine inferiore e uno superiore la distanza non sarà data dalla somma delle due distanze. A prevalere sarà invece la distanza maggiore tra le due. Questo meccanismo è chiamato “margin collapsing”. Tale meccanismo non si applica ai box adiacenti in senso orizzontale.
I margini sono gli unici elementi del box model che possono avere valori negativi. Ciò non è consentito per padding, bordi, altezza e larghezza.

2.5.2 Gestione dello sfondo

In questo paragrafo verranno esaminate le proprietà relative alla gestione dello sfondo. Sin dalle origini di HTML è stato possibile definire un colore o un'immagine di sfondo per le nostre pagine web. La scelta si restringeva comunque a due elementi: il corpo principale della pagina (<body>) o le tabelle. Un'altra limitazione riguardava il comportamento delle immagini di sfondo: esse venivano ripetute in senso orizzontale e verticale fino a riempire l'intera area della finestra, del frame o della tabella. Proprio questo comportamento ha fatto sempre propendere per la scelta di piccole textures in grado di dare la sensazione visiva dell'uniformità. Grazie ai CSS queste limitazioni vengono spazzate via e le possibilità creative, compatibilità permettendo, sono davvero infinite.
Ecco la lista delle proprietà, applicabili, ed è questa la prima grande innovazione dei CSS, a tutti gli elementi:

- background-color
- background-image
- background-repeat
- background-attachment

- background-position

Ciascuna di essa definisce un solo, particolare aspetto relativo allo sfondo di un elemento.

Background-color
Definisce il colore di sfondo di un elemento. Questa proprietà non è ereditata.
Esempio:

```
div { background-color: white; }
```

Background-image
Definisce l'URL di un'immagine da usare come sfondo di un elemento. Questa proprietà non è ereditata.
Esempio:

```
div  { background-image: url(sfondo.gif);
       background-color: white; color: black }
```

Una buona norma e il buon senso vogliono che quando si definisce un'immagine come sfondo si usi sempre anche un colore di sfondo e che questo colore consenta una lettura agevole del testo. Questo perché se le immagini sono disabilitate il browser mostrerebbe il suo colore di sfondo predefinito: se questo è bianco e il nostro testo pure le informazione contenute nella pagina sarebbero inaccessibili.

Background-repeat
Con questa proprietà iniziano le novità. Essa consente di definire la direzione in cui l'immagine di sfondo viene ripetuta sia sull' asse X che sull'asse Y. Proprietà non ereditata.
Esempio:

```
div  { background-image: url(sfondo.gif);
       background-color: white; color: black;
       background-repeat: repeat-x; }
```

Background-attachment
Con questa proprietà si imposta il comportamento dell'immagine di sfondo rispetto all'elemento cui è applicata e all'intera finestra del browser. Si decide, in pratica, se essa deve scorrere insieme al contenuto o se deve rimanere fissa. Proprietà non ereditata.
Esempio:

```
div  { background-image: url(sfondo.gif);
       background-color: white; color: black;
       background-repeat: repeat-x;
       background- attachment: fixed }
```

Background-position
Definisce il punto in cui verrà piazzata un'immagine di sfondo non ripetuta o da dove inizierà la ripetizione di una ripetuta. Si applica solo agli elementi blocco o rimpiazzati. Bisogna però prestare attenzione alla compatibilità e alla resa, non omogenea, tra i vari browser. Proprietà non ereditata.
Esempio:

```
div  { background-image: url(sfondo.gif);
       background-color: white; color: black;
       background-repeat: no-repeat;
       background-position: 50px 50px }
```

2.5.3 Gestione del testo

Se c'è un aspetto essenziale dei CSS questo è il nuovo approccio che hanno introdotto per la gestione del testo. Prima esisteva solo il tag <font>: pagine pesanti e difficili da gestire. Oggi si ha qualcosa che può trasformare una pagina web in un perfetto esempio di stile tipografico e che finalmente consente, almeno in parte, la potenza e la flessibilità di un word-processor.

La proprietà che definiscono il modo in cui il testo appare sullo schermo sono tante. Si è deciso di descrivere solo le più importanti. Iniziamo quindi dalle proprietà di base. Sono quelle che definiscono i seguenti aspetti:

- il font da usare
- la sua dimensione
- la sua consistenza
- l'interlinea tra i paragrafi
- l'allineamento del testo
- la sua decorazione (sottolineature e simili)

Font-family
La proprietà font-family serve a impostare il tipo di carattere di una qualunque porzione di testo. Si applica a tutti gli elementi ed è ereditata.

Gli uomini del W3C hanno creato un meccanismo che consente all'autore di impostare nei CSS non un font singolo e unico, ma un elenco di caratteri alternativi. Il meccanismo funziona così:

```
selettore {font-family: font1, font2, famiglia_generica;}
```

Quando la pagina verrà caricata, il browser tenterà di usare il primo font della lista. Se questo non è disponibile userà il secondo. In mancanza anche di questo verrà utilizzato il font principale della famiglia generica presente sul sistema. La spiegazione di tutto ciò è semplice: ovviare al problema dei diversi font presenti sulle piattaforme software.

Dunque quando si imposta la proprietà font-family si possono usare tutti i font che si vuole, ma non dimenticate mai di inserire alla fine l'indicazione di una famiglia generica.
Esse sono cinque (tra parentesi riportiamo i caratteri predefiniti su ciascuna sui sistemi Windows):

1. serif (Times New Roman)
2. sans-serif (Arial)
3. cursive (Comic Sans)
4. fantasy (Allegro BT)
5. monospace (Courier)

I nomi dei font della lista vanno separati dalla virgola. I caratteri con nomi composti da più parole vanno inseriti tra virgolette.

Font-size
Insieme a font-family è la proprietà considerata essenziale nella definizione dell'aspetto del testo, di cui definisce le dimensioni. Applicabile a tutti gli elementi ed ereditata.

```
selettore {font-size: <valore> ;}
```

Font-weight
Serve a definire la consistenza o "peso" visivo del testo. Si applica a tutti gli elementi ed è ereditata.
Il "peso" visivo di un carattere può essere espresso con una scala numerica o con parole chiave: valori numerici 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 ordinati in senso crescente (dal leggero al pesante)

```
selettore {font-weight: <valore> ;}
```

Font-style
Imposta le caratteristiche del testo in base ad uno di questi tre valori:

- **normal**: il testo mantiene il suo aspetto normale
- **italic**: formatta il testo in corsivo
- **oblique**: praticamente simile a italic

La proprietà si applica a tutti gli elementi ed è ereditata.

```
selettore {font-style: <valore> ;}
```

Line-height
Grazie a line-height è possibile finalmente usare nelle pagine WEB un sistema di interlinea vero e proprio. La proprietà, in realtà, serve a definire l'altezza di una riga di testo all'interno di un elemento blocco. Ma l'effetto ottenuto è appunto quello tanto agognato da tutti i web designer: un modo per impostare uno spazio tra le righe. La proprietà si applica a tutti gli elementi ed è ereditata.

```
selettore {line-height: <valore>;}
```

Text-decoration
Imposta particolari decorazioni e stili per il testo come una linea sopra al testo,sotto,attraverso o anche il lampeggio del testo stesso. Ereditabile e applicabile a tutti gli elementi.

```
selettore {text-align: <valore>; }
```

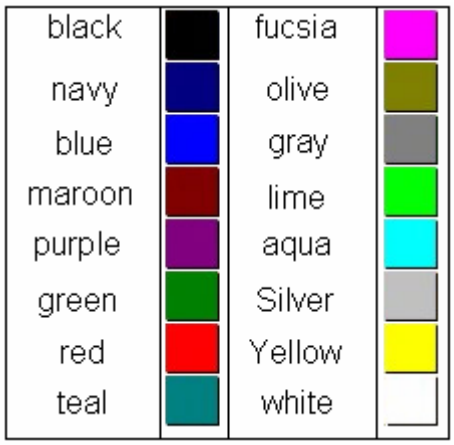
Esistono altre proprietà più avanzate di gestione del testo per intervenire su aspetti complessivamente poco importanti e per alcune di esse il supporto non è affatto garantito.

2.5.4 Gestione del colore

Tra le proprietà di CSS2 la gestione del colore è una di quelle che può a buon diritto dirsi fondamentale. Si esaminerà prima le diverse possibilità di definire i valori dei vari colori, per poi analizzare usi e sintassi della proprietà “color”.

2.5.4.1 Definizione del colore

I 16 colori basilari della palette VGA standard di Windows sono definiti da altrettante parole chiave:



Le semplici 16 parole chiave non esauriscono ovviamente la gamma dei colori visualizzabili su un monitor moderno. Già in HTML era possibile impostare il colore di un elemento servendosi di codici esadecimali. In essi, le prime due lettere (o numeri) corrispondono ai valori per il colore rosso (RED), la seconda coppia fa riferimento al verde (GREEN), l'ultima al blue (BLUE).

Esempio
#CC0000 **Il codice indica una tonalità di rosso.**

Un altro modo per rappresentare i colori è quello di usare per i tre elementi base del sistema RGB una lista di valori in percentuale separati da una virgola.

Esempio
rgb(0%, 0%, 0%) **Nero**
rgb(100%, 100%, 100%) **Bianco**

Un ultima possibilità è data definendo i valori di rosso, verde e blue con tre valori compresi, rispettivamente, tra 0 e 255. Sistema ben noto a chi usa programmi di grafica. Anche qui, i tre valori vanno separati da una virgola.

Esempio
rgb(230, 230, 70) **Il codice indica una tonalità di giallo.**
rgb(100, 50, 255) **Il codice indica una tonalità di blu**.

2.5.4.2 La proprietà color

Per ogni elemento della pagina WEB si possono definire almeno tre colori:

1. il colore di primo piano (foreground color)
2. il colore di sfondo (background color)
3. il colore del bordo (border color)

La proprietà color definisce esclusivamente:
il colore di primo piano, ovvero quello del testo
il colore del bordo di un elemento quando non si imposti esplicitamente questo ultimo con le proprietà border o border-color.
Per il colore di sfondo si usa la proprietà background illustrata in precedenza.
Una buona norma dei CSS vuole comunque che per un elemento di cui si sia definito il colore di primo piano si definisca sempre un colore di sfondo.

2.5.5 Gestione del posizionamento

La parte dei CCS relativa al posizionamento degli elementi è un argomento complesso, ma affascinante. In prospettiva quello più promettente per le conseguenze che può portare. E' con queste regole che si può pensare di realizzare il layout delle pagine solo con i CSS e giungere così alla definitiva e reale separazione tra presentazione e struttura.

La proprietà **position** è fondamentale per la gestione della posizione degli elementi, di cui determina la modalità di presentazione sulla pagina. Si applica a tutti gli elementi e non è ereditata.
La sintassi è:

```
<selettore> { position: <valore>;}
```

I diversi valori che può assumere meritano una spiegazione approfondita da cui emergeranno i concetti di base che governano le regole sul posizionamento.

I. Static
E' il valore di default, quello predefinito per tutti gli elementi non posizionati secondo un altro metodo. Rappresenta la posizione normale che ciascuno di essi occupa nel flusso del documento.

II. Absolute
L'elemento, o meglio, il box dell'elemento viene rimosso dal flusso del documento ed è posizionato in base alle coordinate fornite con le proprietà top, left, right o bottom. Il posizionamento avviene sempre rispetto al box contenitore dell'elemento.

III. Fixed
Usando questo valore il box dell'elemento viene, come per absolute, sottratto al normale flusso del documento. La differenza sta nel fatto che per fixed il box contenitore è la finestra principale del browser, ovvero l'area del contenuto. Altra differenza fondamentale è che un box posizionato con fixed non scorre con il resto del documento.

IV. Relative
L'elemento viene posizionato relativamente al suo box contenitore. In questo caso il box contenitore è il posto che l'elemento avrebbe occupato nel normale flusso del documento. La posizione viene anche qui impostata con le proprietà top, left, bottom, right. Ma qui esse non indicano un punto preciso, ma l'ammontare dello spostamento in senso orizzontale e verticale rispetto al box contenitore.

Passiamo ora all'analisi delle proprietà che concretamente definiscono dove un elemento posizionato va a collocarsi, dal momento che con position viene stabilito solo il metodo.

Con top, bottom, left e right possiamo definire rispettivamente la distanza superiore, inferiore, sinistra e destra.
Le proprietà visibility e z-index influiscono invece sull'aspetto visuale dei box definendone la visibilità e la relazione con gli altri box presenti nella pagina.
Segue una breve descrizione di queste ultime due proprietà:

Visibility
Questa proprietà determina se un elemento debba essere visibile o nascosto. Si applica a tutti gli elementi e non è ereditata.

Z-index
Con questa proprietà si imposta l'ordine di posizionamento dei vari elementi sulla base di una scala di livelli. E' un meccanismo simile a quello dei layer sovrapposti di Photoshop ed è utile nel contesto del posizionamento dinamico. In seguito ad un posizionamento, infatti, è possibile che un elemento si sovrapponga ad un altro rendendolo illeggibile. Impostando lo z-index è possibile modificare l'ordine di tutti gli elementi.

2.6 Tre proprietà speciali: display, float clear

Le tre proprietà speciali display, float e clear possono modificare radicalmente la presentazione del documento. Sono strumenti potenti ma bisogna tenere anche conto di serie implicazioni a livello di rendering tra i diversi browser.

1. Display
Essa ha una sola, semplice funzione: definire il trattamento e la presentazione di un elemento. Fin quando non la si usa ogni elemento segue la sua natura e il suo comportamento standard, ma con display possiamo intervenire su di esso e in certi casi ribaltarlo facendo diventare un elemento blocco inline o viceversa.

I valori più utilizzati per display sono:
- inline. Valore di default. L'elemento assume le caratteristiche degli elementi inline.
- block. L'elemento viene trattato come un elemento blocco.
- list-item. L'elemento si trasforma in un elemento lista.
- none. L'elemento non viene mostrato. O meglio: è come se non fosse nemmeno presente nel documento, in quanto non genera alcun box.

Esempi di applicazione
Per capire bene a cosa serve la proprietà display ecco alcuni esempi di applicazione utilizzati anche nella realizzazione del portale di Ingegneria.
Le immagini, per esempio, sono per loro natura elementi inline, si inseriscono nel testo ed è talvolta complicato gestirne il posizionamento. Se si volesse mostrarle tutte in una riga basterebbe impostare il display su block, così:

```
img {display: block;}
```

Per non parlare dell'utilità del valore none. Se si desidera creare una pagina alternativa fatta solo di testo basterà costruite un CSS alternativo dove imposterete questa regola:

```
img {display: none;}
```

e nella pagina principale inserire un link: "Versione solo testo"

La vera utilità della proprietà display, emerge comunque in linguaggi non strutturali come XML. Uno dei modi per rendere un file XML in un browser è proprio quello di formattarlo con un CSS. Ma gli elementi XML non contengono nulla su presentazione e struttura, a differenza di XHTML quindi si utilizza la proprietà display per farli vedere o meno.

2. Float

Con questa proprietà è possibile rimuovere un elemento dal normale flusso del documento e spostarlo su uno dei lati (destro o sinistro) del suo elemento contenitore. Il contenuto che circonda l'elemento scorrerà intorno ad esso sul lato opposto rispetto a quello indicato come valore di float. La proprietà non è ereditata.

I valori assumibili da float sono:

- left. L'elemento viene spostato sul lato sinistro del box contenitore, il contenuto scorre a destra.
- right. L'elemento viene spostato sul lato destro, il contenuto scorre a sinistra.
- none. Valore iniziale e di default in mancanza di una dichiarazione esplicita. L'elemento mantiene la sua posizione normale.

Se viene usato il float con le immagini non si avranno problemi perchè esse hanno una dimensione intrinseca che il browser riconosce. Ma se viene applicato ad altri elementi si deve esplicitamente impostare una dimensione orizzontale con la proprietà width per non rischiare problemi di layout diverso da browser a browser.

Esempi:

```
div {width: 200px; float:right;}   /*larghezza 200px, testo spostato a destra */
img.foto {float: left;}  /*immagine spostata a sinistra */
```

3.clear

La proprietà clear serve a impedire che al fianco di un elemento compaiano altri elementi con il float. Si applica solo agli elementi blocco e non è ereditata.
L'origine di tale proprietà è dovuta al fatto che serviva una funzione uguale e contraria al float; cioè riallinea il flusso del documento modificato precedentemente dal float.

I valori che clear può assumere sono:

- none. Gli elementi con float possono stare a destra e sinistra dell'elemento.
- left. Si impedisce il posizionamento a sinistra.
- right. Si impedisce il posizionamento a destra.
- both. Si impedisce il posizionamento su entrambi i lati.

Esempi

```
h1 {clear: both;}  /* Riallinea il flusso su entrambi i lati */
```

Capitolo 3: Il multilinguismo

3.1 Metodologie per la gestione di un sito multilingua

Per la maggior parte delle aziende la possibilità di offrire ai clienti la scelta della lingua da utilizzare per interagire con il sito è sicuramente un elemento vantaggioso. Tuttavia, molte di queste ignorano l'opportunità che hanno oppure la risolvono realizzando versioni diverse dello stesso sito.
Amazon.com, ad esempio, fornisce svariate versioni dello stesso sito, come Amazon.de e Amazon.nl.
Se tutte le versioni di un sito potessero essere aggiornate simultaneamente, l'esistenza di più versioni non rappresenterebbe un problema. Tuttavia, nella maggior parte dei casi, ciascuna versione deve essere aggiornata separatamente, determinando spesso delle differenze significative tra l'una e l'altra. La versione inglese di Amazon.com, ad esempio, è molto più ricca di funzionalità rispetto alla versione tedesca o a quella olandese. Tali notevoli differenze possono provocare confusione nel team del progetto e rendere il sito di difficile gestione. D'altro canto però bisogna valutare la possibilità che un azienda non voglia condividere tutte le informazioni o i servizi per le diverse lingue che vuole fornire o che non ne abbia la possibilità.
Infatti il continuo aggiornamento di un sito e di tutte le sue funzioni richiede molto tempo e conoscenza da parte di una figura che si presti intermente, o quasi, a questo lavoro e non tutte le realtà che dispongono di uno spazio WEB hanno questa possibilità.

DI seguito verranno discusse tre diverse metodologie per la creazione di un sito multilingue, ognuna delle quali con i propri vantaggi e svantaggi.

3.1.1 Pagine Duplicate

La duplicazione delle pagine XHTML è il metodo più semplice e diretto per la realizzazione di un sito multilingue.
Per creare una nuova versione di un sito basta fare una duplicazione delle pagine di interesse e cambiare, o eliminare, tutte quelle informazioni che si vogliono offrire nella nuova lingua. Il problema sorge quando si vogliono supportare svariate versioni, ognuna con una propria lingua, di un sito WEB. Questo implica la creazione ripetuta di ogni pagina per ogni lingua e, come è lecito pensare, non è affatto conveniente in termini di spazio occupato su disco. Infatti le pagine aumentano da un numero N a X*N, dove X è il numero delle diverse lingue che si vuole fornire.
Inoltre questo metodo fa aumentare in modo esponenziale la difficoltà nella gestione del portale in quanto le modifiche effettuate su una pagina devono essere eseguite anche sulle altre pagine con le stesse informazioni ma di lingue differenti.
Possiamo pensare quindi che questa metodologia sia utile per siti che non aggiornano frequentemente le pagine WEB e che non offrono il supporto di molte lingue.

3.1.2 Pagine dinamiche con script

Una diversa implementazione per realizzare un sito multilingue è rendere la pagina dinamica.
Questa tecnica è molto utilizzata nei siti di commercio elettronico dove le informazioni sono gli insiemi di proprietà dipendenti dalla lingua e relative ai prodotti, come il nome e la descrizione.
Per gestire le proprietà dipendenti dalla lingua, è sufficiente adattare il codice a ogni proprietà, in modo da identificare la lingua in cui tale proprietà verrà visualizzata.
Il prodotto potrebbe contenere, ad esempio, una proprietà chiamata “descrizione”, che si desidera visualizzare in inglese, francese e olandese. A questo scopo, è sufficiente impostare una variabile per richiamare il file esterno contente tutti i valori di una determinata lingua, da assegnare alla variabili del documento. Tale variabile può essere un parametro immesso dall'utente o una proprietà del profilo utente.
Naturalmente dovrà essere creata un’altra pagina per ogni lingua supportata dove le variabili vengono definite.
Il risultato finale è una pagina XHTML con il testo nella lingua prescelta.

I vantaggi di questa metodologia sono molteplici:

- In caso di modifica del testo o della descrizione delle immagini si dovranno cambiare solo i valori assegnati alle stringhe di ogni lingua e non intaccare il layout della pagina.
- Lo script non aumenta la difficoltà nella comprensione del codice poiché per cancellare o modificare il layout basterà farlo nella pagina principale.
- Per implementare una lingua basterà aggiungere una nuova pagina contente i valori da assegnare alle variabili delle pagine; senza intaccare la pagina principale.
- Il numero delle pagine del portale aumenta da N a N+X, cioè una per ogni lingua ed una principale.

Segue un esempio di codice scritto in PHP, ma è possibile farlo anche in ASP, che svolge proprio la funzionalità appena descritta.

Sono state create 4 pagine:

- Index.html: è la pagina iniziale che permette la scelta della lingua.
- Documento.php: è la pagina principale dove verranno caricate tutte le informazioni tramite le variabili.
- Italian.php: contiene tutti i valori che devono assumere le variabili per ottenere la pagina (documento.php) in lingua italiana.
- English.php: contiene tutti i valori che devono assumere le variabili per ottenere la pagina (documento.php) in lingua inglese.

Index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
      <head>
         <title>Università di Modena e Reggio Emilia</title>
      </head>
      <body>
           <div align="center">
              <p>Scegli la lingua con cui vuoi navigare il portale</p> <br />

                 / * Richiamo la pagina documento.php passando come
                                parametro la scelta effettuata*/
```

```
                <a href="documento.php?newlang=italian">Italiano </a>
                <a href="documento.php?newlang=english">English </a>
            </div>
        </body>
</html>
```

documento.php

```
<?php
$language = "";

        /* controllo che non esistano cookie sul pc , se ci sono vengono
                caricati altrimenti si setta la lingua in Italiano  */

if (!isset($_GET['newlang']) || $_GET['newlang'] == "") {
    if (!isset($_COOKIE['language']) || $_COOKIE['language'] == "") {
        setcookie("language","italian",time()+31536000);
        $language = "italian";
    }
    else {
        $language = $_COOKIE['language'];
    }
}
else {
    setcookie("language",$_GET['newlang'],time()+31536000);
    $language = $_GET['newlang'];
}

        /* includo il file grazie al concatenamento della variabile
                $language con la stringa ".php"  */

include_once("{$language}.php");

        /* Al posto del testo sono inserite delle variabili che assumeranno
            il valore contenuto nel file richiamato alla riga precedente */

echo "<center>"._HOME."</center><br />";
echo "<center><h2>"._WELCOME."</center><br />";
echo "<center><img src=\"2_{$language}.gif\"></center>";
?>
```

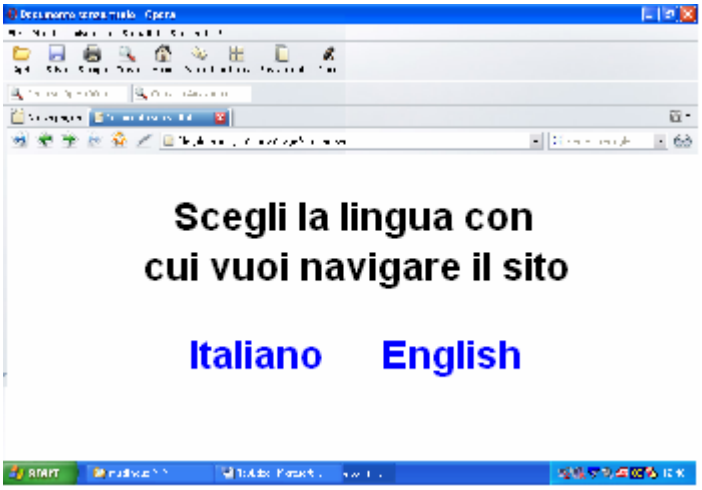
italian.php

```
<?
                /* Assegno i valori alle variabili */
define("_HOME","Università degli studi di Modena e Reggio");
define("_WELCOME","Benvenuto nella pagina di prova per il sito multilingua!");
?>
```

English.php

```
<?
                /* Assegno i valori alle variabili */
define("_HOME","University of the studies of Modena e Reggio");
define("_WELCOME","Welcome in the page of test for the multilanguage web site!");
?>
```

Ed ecco come si presentano all'utente le pagine appena commentate:



Index.html

Documento.php con scelta della lingua in italiano

Documento.php con scelta della lingua in inglese

3.1.3 Pagine gestite attraverso un CMS

L'ultimo metodo che verrà descritto è attraverso l'uso di un CMS.
CMS è l'acronimo di Content Management System (Sistema per la Gestione dei Contenuti).
E' un software per la realizzazione e la gestione in modo semplificato di siti dinamici che possono accrescere e mutare il proprio contenuto continuamente.
Un CMS consente al committente del sito di occuparsi direttamente della sua gestione senza intermediari esterni.
Inoltre mantiene separato il codice di programmazione, la grafica e i contenuti.
Questo significa che su ognuno di questi tre elementi possono intervenire persone diverse con competenze specifiche:

- il grafico web realizza l'interfaccia grafica del sito
- il programmatore informatico configura e personalizza la struttura del sistema integrando il materiale grafico e realizzando eventuali template
- l'azienda o l'ente committente provvede a gestire i contenuti

I CMS sono già predisposti per gestire il multilinguismo, quelli più recenti utilizzano XML per raggruppare tutti i dati all'interno della struttura XML stessa.

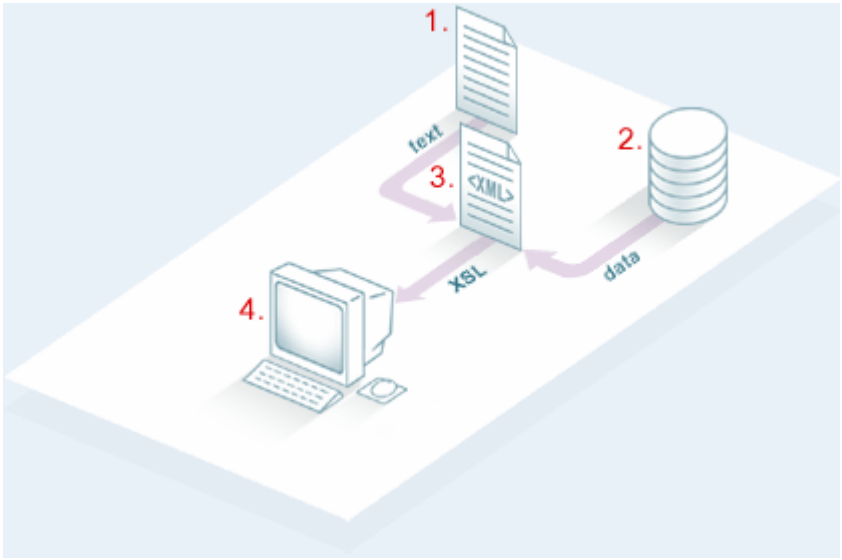


FIGURA 9

Come si può vedere dalla figura 9 il testo fisso viene estrapolato da una tabella di testo multilingue, e i contenuti del database vengono presi da un modulo, che consiste in un file salvato in una determinata directory.
Quindi per inserire una nuova lingua nel sito basterà scaricare il modulo della lingua prescelta, messo a disposizione sul sito web della casa produttrice del CMS, e salvarlo nella directory indicata. Automaticamente il CMS riconoscerà il nuovo modulo e inserirà una nuova voce nella colonna per la scelta rapida della lingua nelle pagine HTML.
La gestione multilingua del sito con un CMS, solitamente laboriosa e complessa, diviene quindi più semplice e rapida.

3.2 Scelta e motivazione del metodo utilizzato

Nello sviluppare il bilinguismo del portale WEB di facoltà si è deciso di applicare il primo metodo descritto in questo capitolo, cioè di creare una copia delle pagine di interesse e di cambiare il testo in lingua inglese. Questo perchè le pagine del portale sono in gran parte costituite da descrizioni discorsive ed inoltre non tutte sono da considerarsi interessanti per gli utenti stranieri. Per quanto riguarda l'utilizzo di un CMS implicherebbe, oltre ad una completa rivoluzione nel metodo di gestione del portale, un costo non trascurabile e il vincolo ad un particolare ambiente software proprietario.

3.3 Progetto delle aree del portale interessate dalla traduzione in lingua inglese

Da un'analisi preventiva per decidere le pagine da inserire nella versione inglese, la scelta è ricaduta su quelle di interesse generale come: il Manifesto degli Studi (che contiene il nome e le informazioni sintetiche sugli insegnamenti), le informazioni relative alla facoltà e quelle dei servizi agli studenti, come le borse di studio o i bandi per gli alloggi.

Il materiale didattico e le pagine relative ai docenti non vengono tradotte in inglese perché l'applicazione WEB del portale permette la gestione individuale delle pagine personali. Quindi ogni insegnante avrebbe dovuto garantire l'inserimento delle propria pagina in inglese, cosa che non è possibile gestire vista la quantità di docenti che lavorano presso l'università e le diverse conoscenze linguistiche che ha ognuno di loro.

Per limitare la maggior parte degli svantaggi che comporta la metodologia della duplicazione delle pagine si è deciso di sfruttare il potente database che la Facoltà vanta per gestire le informazioni più dinamiche, come la presentazione dei Corsi di Studio o le informazioni relative ad ognuno di essi. Per questo motivo è stato esteso il database, fornendo così campi supplementari per la lingua inglese.
Sono state modificate anche le "form" dell'Intranet e gli script SQL per l'inserimento, la modifica, la cancellazione o per il solo reperimento dei record nella lingua prescelta.

Per semplificare la comprensione della struttura al programmatore e per non duplicare tutto il portale, compresi i file per il collegamento al database, si è deciso di mantenere le pagine in inglese nelle stesse cartelle di quelli in italiano con l'aggiunta della stringa "-e" nel nome logico, prima dell'estensione del file stesso; di modo che una futura ricerca e modifica dei file sia semplice e veloce.

In appendice B è riportata la lista delle tabelle modificate e gli script SQL per la creazione di colonne contenenti le informazioni in lingua inglese. In questo modo lanciando gli script è possibile modificare tutto il database senza intervenire manualmente sulle tabelle.
Inoltre sono stati elencati alcuni script creati ed utilizzati nelle prime fasi di test del portale bilingue che inseriscono dei valori nelle colonne adibite alle informazioni in inglese.
Questi valori vengono letti e copiati dalle relative colonne in italiano, già presenti nella base dati del portale, e inserite nelle colonne in inglese concatenando loro la stringa "--inglese--" per una rapida verifica delle funzionalità delle pagine bilingue.

Capitolo 4: Il progetto realizzato

In questo capitolo sono descritte le modalità con cui si è realizzato il nuovo layout accessibile del portale WEB della Facoltà di Ingegneria di Modena.
Vengono inoltre elencati alcuni problemi riscontrati durante la programmazione e spiegati i metodi utilizzati per la loro risoluzione.

4.1 Analisi del portale

Il primo passo da compiere per creare un portale accessibile è avere una buona suddivisone logica delle pagine, in modo tale da sfruttare tutti i vantaggi offerti dai CSS.
Durante il lavoro di stage presso l'università è stata fatta un analisi integrale del portale WEB, mirata a decidere la suddivisione da applicare alle pagine per la rielaborazione del codice in formato XHTML.
Il "vecchio" layout utilizza tabelle trasparenti annidate tra loro per il posizionamento degli elementi o comunque introduce elementi non validi per lo standard XHTML Strict. Come spiegato nel capitolo 2, questo non è affatto corretto per la realizzazione di un portale accessibile in quanto le tabelle devono essere usate per dati tabulari e non per facilitare l'opera di web-design da parte del programmatore.

Si è quindi optato per una suddivisone che non modifica l'impaginazione finale del portale ma che viene costruita in modo concettualmente diverso, presentando sempre una testata, un menù di navigazione a sinistra, un corpo della pagina e un menù a piè di pagina, come è possibile vedere in figura 11.

FIGURA 10 (Layout senza CSS)

FIGURA 11 (Layout con CSS)

4.2 Ricodifica del codice

Terminata l'analisi del portale si può passare alla parte di programmazione vera e propria. Per facilitare il lavoro sono stati creati prima i CSS, ognuno dei quali racchiude regole per un determinato scopo, e poi si sono modificate le pagine HTML trasformandole in XHTML 1.0 Strict.

I file CSS creati sono:

- imposta.css
- classici.css
- tabelle.css
- print.css
- stili1.css

Questi file meritano una spiegazione approfondita per avere maggior chiarezza sulle regole contenuti in essi.

imposta.css

Il CSS racchiude tutti le regole adibite alla formattazione della pagina, come il posizionamento della testata, del menù di navigazione, del corpo del documento e del menù situato a piè di pagina.
Sono inoltre contenute tutte le regole per la corretta impostazione grafica delle liste di elementi utilizzate per i menù di navigazione.
Oltre a queste 3 liste ne è stata creata un'altra non visibile sullo schermo che consente agli screen-reader di elencare agli utenti non vedenti i tasti di scelta rapida per i salti di pagina.

classici.css

Questo foglio di stile contiene le classi fondamentali utilizzate per il posizionamento del testo e la sua formattazione, come il grassetto, il corsivo e il font.
Il CSS racchiude anche le regole per gestire in modo differente i link visitati da quelli non visitati.
Infine contiene le impostazioni per la dimensione da applicare agli elementi <h1>…<h6> dal momento che quest'ultimi devono essere utilizzati per creare blocchi strutturali nel documento e non per il dimensionamento del testo.

FIGURA 12 (Layout senza CSS)

FIGURA 13 (Layout con CSS)

tabelle.css

Nel file sono state create classi che vanno a modificare impostazione all'interno delle tabelle, come la larghezza delle celle, il loro sfondo, il font e il posizionamento del testo.

FIGURA 14 (Layout senza CSS)

FIGURA 15 (Layout con CSS)

print.css

Questo foglio di stile contiene quattro selettori ID che tramite la proprietà display eliminano durante la fase di stampa il menù in alto, a sinistra e in basso della pagina, lasciando quindi solo il corpo del documento.
Con questa tecnica vengono eliminate tutte le pagine del portale a cui si accedeva tramite il link “versione stampabile”, utilizzando così meno spazio su disco e agevolando il lavoro dei webmaster.

stili1.css

Nel CSS sono impostati tutti gli stili particolari che non si presentano in modo ripetitivo nelle pagine del portale e tutte le regole che caricano immagini a scopo puramente estetico.
I font dei caratteri non possono essere inseriti nel codice XHTML perché lo stesso standard non lo permette, mentre le immagini a scopo estetico non possono essere inserite perché sono solo una veste grafica della pagina e non un veicolo di informazione (vedi tesi di laurea di Caterina Barbieri).

FIGURA 16 (Layout senza CSS)

FIGURA 17 (Layout con CSS)

Dopo aver creato i CSS si può passare alla reale migrazione dallo standard HTML 4.0 a XHTML 1.0 Strict in tutte le pagine.

Essendo la mole di lavoro enorme (le pagine da modificare sono circa 90) si è deciso di creare un template accessibile.
Questo documento contiene al suo interno il codice per la formattazione standard richiamato in tutte le pagine. In questo modo si è semplificato il lavoro di reengineering del

portale perché le variazioni vanno effettuate solo per il corpo del documento e non per tutta la pagina, diminuendo i tempi di programmazione.

Ecco il codice del template creata:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml" lang="it" >
        <head>
            <meta http-equiv="content-type" content="text/html; charset=iso-8859-15" />
             <title>Template</title>
                  <link rel="stylesheet" type="text/css" href="./CSS/imposta.css" />
                  <link rel="stylesheet" type="text/css" href="./CSS/classici.css" />
        </head>
        <body class="container">
                <!-- Meccanismi per saltare parti della pagina e andare direttamente ai contenuti-->
                <!--#include file="./include/ing/jump.html" -->
                <!--Fine meccanismo per saltare parti della pagina e andare direttamente ai contenuti-->

                <div id="header">
                        <!-- inizio parte alta  -->
                        <!--#include file="./include/ing/header.html" -->
                        <!-- fine parte alta  -->
                </div>

                <div id="visite">
                        <%'Stampo a Video il Numero di Accessi alla Pagina
                             prendendolo dalla Tabella del Logging nel DB Statistiche

        Response.Write(AccessiParametri(Request.ServerVariables("SCRIPT_NAME"),Request.QueryString))
                        %>
                        <br />

                        <!-- inizio procedura che visualizza la data dell'ultimo
                             aggiornamento per la pagina corrente -->
                        <!--#include file="./include/ing/ultimo_agg.asp" -->
                        <%ultimo_agg("fac.asp")'il parametro è il nome della pagina corrente%>
                        <!-- fine procedura che visualizza la data dell'ultimo aggiornamento -->
                </div>

                <!-- inizio procedura per autentificazione -->
                <!-- #INCLUDE FILE="./CampusOne/Utility/Utility.asp"-->
                <!-- fine procedura per autentificazione -->

                <div id="barra">
                        <!--#include file="./include/ing/bar.html" -->
                </div>

                <div id="content">
                                **...Qui scrivo il corpo del documento...**
                </div>

                <div class="footer">
                        <!-- inizio footer -->
                        <!--#include file="./include/ing/footer.html" -->
                        <!-- fine footer -->
                </div>
        </body>
</html>
```

4.3 Validazione

Al termine dell'opera di reengineering e ottenuto il layout grafico desiderato si può passare alla validazione delle pagine scritte in XHTML.

Il controllo della validità del codice XHTML può essere effettuato automaticamente, ricorrendo ad appositi software, come il "MarkUp Validation Service" presente sul sito W3C (http://validator.w3.org/) che convalida il codice XHTML rispetto alla DTD scelta.
Questo però non basta ancora perché un documento sia pienamente accessibile. Si possono, infatti, costruire pagine perfettamente aderenti alla sintassi di XHTML e CSS, piene tuttavia di contenuti inaccessibili. Occorre quindi, per valutare il grado di accessibilità raggiunto, ricorrere a degli strumenti specifici come il validatore fornito dalla Watchfire Corporation, all'indirizzo "http://webxact.watchfire.com/".
Nel correggere gli errori di codice, il validatore Webxact fornisce un discreto aiuto: indica allo sviluppatore la riga del listato della pagina dove è presente l'errore e descrive il tipo di errore. Quest'ultimo ausilio è utile però solo se si conosce già a sufficienza la sintassi di XHTML, tanto da saper identificare e risolvere il problema.

La maggior parte degli errori riscontrati dal validatore riguardano:

- Elementi aperti e non chiusi o viceversa
- Elementi incastrati invece che annidati (... , invece di ...)
- Uso di elementi e attributi non consentiti dalla DTD usata (problema che si verifica tipicamente quando si inseriscono nel codice elementi e attributi di presentazione dopo aver dichiarato la DTD rigorosa, che non li prevede)
- Uso del carattere '&' in una stringa di query (va sostituito con l'entità carattere '&')
- Uso di valori di attributo non consentiti
- DTD non dichiarata

Una volta riscontrata la conformità del codice alla DTD dichiarata ad inizio pagina, il passo successivo è convalidare il codice CSS usato nei fogli di stile interni o esterni associati alla pagina. Anche per questo riscontro, esiste un apposito software sviluppato dal W3C: CSS Validator ("http://jigsaw.w3.org/css-validator/").

La validazione dei fogli di stile può essere effettuata in tre modi:

1. fornendo l'URI del documento da validare (il file CSS o anche il file XHTML a cui il CSS è associato)
2. inserendo il codice CSS all'interno di un apposito campo modulo
3. spedendo sul sito W3C il file da analizzare.

Quale che sia il metodo scelto, il funzionamento del programma è analogo a quello del validatore di codice XHTML: se vengono riscontrati errori, viene fornito un elenco contenente la riga di codice e il tipo di errore.
Gli errori più comuni a cui occorre porre rimedio sono i seguenti:

- mancanza del punto e virgola finale che chiude la dichiarazione di una proprietà
- mancanza della parentesi graffa che chiude un elenco di proprietà
- un colore dichiarato in valori esadecimali non preceduti dal simbolo '#'

- ‘sans-serif' (famiglia generica di caratteri) scritto senza il trattino separatore o senza le doppie virgolette
- nomi di classe e id non validi
- un commento (/* ... */) aperto e non chiuso, o viceversa

A differenza del validatore per HTML e XHTML, il validatore CSS fornisce, oltre la lista degli errori, anche una lista di avvertimenti ("warnings", in inglese): si tratta di suggerimenti che aiutano a migliorare l'accessibilità dei documenti. L'ideale è ottenere dei CSS che al termine dell'analisi del validatore W3C risultino privi sia di errori sia di avvertimenti.

Durante la validazione delle pagine del portale si sono incontrati problemi non dati da un errata scrittura del codice ma dovuti ad agenti “esterni”.
Di seguito vengono elencati i più importanti:

- Nelle pagine di Intranet viene utilizzato un editor di testo che permette la formattazione delle informazioni inserite dai docenti. Questo editor crea codice html non valido per eseguire le operazioni richieste e quindi anche se il codice delle pagine XHTML è valido, le pagine visualizzate non lo sono perché le stringhe di testo passate come parametro contengono tag deprecati dalla DTD dichiarata.

- La pagina “index.asp” non è stata resa conforme allo standard XHTML 1.0 Strict perché è utilizzato il tag <iframe> per visualizzare le news. Essendo le news prese da una fonte dati XML si è provato ad utilizzare il tag <object> per rendere la pagina valida in XHTML 1.0 Strict. Purtroppo questo escamotage non è ben supportato dai browser, anche da quelli di ultima generazione.

Conclusioni

Un sito web accessibile è un sito di qualità che dimostra l'attenzione da parte del team di sviluppo per le esigenze degli utenti, compresi gli utenti disabili. Esistono diversi motivi che dovrebbero spingere a progettare e costruire siti web accessibili. Il primo fra tutti è che gli utenti disabili hanno gli stessi diritti di chiunque altro e non è giusto escluderli dai servizi presenti in rete. In secondo luogo gli utenti disabili si dimostrano clienti affezionati con chi cura le loro esigenze. Visto e considerato che si parla di milioni di utenti, rendere accessibile il proprio sito porta vantaggi anche a livello economico. Come ultima motivazione (ma per questo non meno importante) per le amministrazioni pubbliche è obbligatorio il rispetto di alcune norme. Negli Stati Uniti i siti federali devono rispettare le regole imposte dalla Section 508 e in Italia il 9 gennaio 2004 è entrata in vigore una legge, denominata legge Stanca, contenente criteri e strumenti per migliorare l'accessibilità dei siti web e delle applicazioni informatiche da parte di persone disabili.

Obiettivo di questo progetto è stato il reengineering del portale web della Facoltà di Ingegneria di Modena per renderlo conforme alla legge Stanca.
Sulla base del concetto di accessibilità, grazie agli standard di XHTML e CSS, si è divisa la parte strutturale della pagina da quella di presentazione.
In questo modo i documenti scritti in XHTML sono validi secondo la DTD più rigorosa e le pagine sono diventate più gestibili da browser alternativi come quelli vocali (screen-reader) o testuali.
L'accessibilità del portale è aumentata in modo notevole anche grazie al fatto che sono state inserite funzionalità come il testo alternativo per le immagini e la descrizione delle tabelle.

Per questi motivi, il lavoro svolto nel corso di questa tesi, può rappresentare un valido inizio per rendere (e mantenere) ancora più accessibile il portale web della facoltà di Ingegneria di Modena.

Possibili sviluppi futuri

Internet è una realtà in continua evoluzione, di conseguenza, le tecnologie e le soluzioni messe a disposizione dalla rete sono in costante crescita. Per questo motivo, sarà necessario aggiornare il portale se la tecnologia lo permetterà, magari fra un anno quando sarà pubblicato lo standard CSS3 e il WCAG 2.0 (Web Content Accessibility Guidelines).

Essendo anche il portale stesso in continua evoluzione, bisognerà mantenere i criteri di accessibilità adottati fino ad ora per ogni modifica che verrà effettuata perché anche solo l'inserimento sbagliato di un elemento può compromettere la validità di una pagina. Quindi ogni volta che verrà effettuata una modifica, sarà necessario controllare le pagine con uno o più validatori online, sia per quanto riguarda la sintassi XHTML che per l'accessibilità.

Come spiegato nel paragrafo 4.3 l'editor di testo utilizzato nell'Intranet è stato eliminato perché genera codice non valido ai fini dell'accessibilità.
Per poterlo riutilizzare sarà necessario effettuare delle modifiche al software (scritto in Javascript) al fine che inserisca elementi XHTML validi nella pagina e che comunque non permetta sintassi che pregiudichino la validità della pagina stessa.

Per le varie possibilità di sviluppo future non bisogna dimenticare che è già stata svolta la parte di analisi e sviluppo per il supporto del bilinguismo.
Sono state create le pagine in lingua inglese, che però non sono accessibili perché il lavoro è stato svolto prima di rendere accessibile il portale.
Bisognerà quindi inserire gli script ASP già creati nelle pagine in lingua inglese in una copia delle pagine accessibili in lingua italiana. In seguito si dovrà eseguire gli script SQL (elencati nell'appendice B) per la modifica del database e inserire nelle nuove colonne i valori in inglese relativi a docenti, corsi di studio, insegnamenti, etc…

Bibliografia

Documentazione online

[1] Sito per guide XHTML e CSS:
http://www.html.it

[2] Sito per documentazione ASP:
http://www.aspitalia.com

[3] World Wide Web Consortium:
http://www.w3.org

[4] Sito con vari riferimenti per l'accessibilità:
http://www.diodati.org

[5] Sito governativo per l'informazione sull'accessibilità:
http://www.pubbliaccesso.gov.it/

Testi di riferimento

1. Titolo: "Cascading Style Sheets – La guida completa", 2a ed.
   Autore: Eric Meyer
   Casa editrice: O'REILLY, 2004

2. Titolo: ASP
   Autore: Elijah Lovejoy
   Casa editrice: Tecniche Nuove

Appendice A

Lista degli elementi blocco XHTML

La lista elenca tutti gli elementi blocco definiti nelle tre DTD XHTML 1.0. Di ciascuno viene fornita una breve descrizione e vengono specificate, per i più importanti, le regole del corretto annidamento (quali elementi possono contenere e quali no).

La terza colonna specifica il supporto delle tre DTD:

S = Strict; T = Transitional; F = Frameset

Elemento	Descrizione	Supporto DTD
<address>	Definisce un blocco di testo destinato a indirizzi, firme, indicazioni sull'autore. Non può contenere altri elementi blocco.	**STF**
<blockquote>	Usato per riportare citazioni da altri documenti. Il testo inserito viene indentato. Può contenere tutti gli elementi blocco.	**STF**
** **	Centra il testo che racchiude. Sconsigliato in HTML 4.0.	**TF**
<dir>	Crea una lista di tipo directory. Sconsigliato in HTML 4.0	**TF**
** **	Definisce un blocco di contenuto generico o una sezione del documento. Può contenere tutti gli elementi blocco.	**STF**
<dl>	Crea una lista di definizione. Può contenere solo gli elementi <dt> e <dd>.	**STF**
<fieldset>	Usato per raggruppare campi di un form.	**STF**
<form>	Definisce un form. Può contenere i classici elementi dei form ma anche elementi blocco.	**STF**
<h1>..<h6>	Definiscono titoli e sottotitoli. Non possono contenere altri elementi blocco.	**STF**
<hr>	Inserisce una linea divisoria orizzontale. E' un elemento vuoto.	**STF**
<isindex>	Inserisce un elemento simile alle caselle di testo. Sconsigliato in HTML 4.0	**TF**
<menu>	Definisce una lista di tipo menu. Sconsigliato in HTML 4.0	**TF**
<noframes>	Inserisce contenuto alternativo per i browser che non supportano i frames.	**STF**

<noscript>	Inserisce contenuto alternativo per i browser che non supportano gli script.	**STF**
** **	Lista ordinata. può contenere solo l'elemento	**STF**
<p>	Definisce un paragrafo. Non può contenere altri elementi blocco, ma solo testo o elementi inline.	**STF**
<pre>	Definisce testo preformattato che mantiene le impostazioni dello spazio bianco.	**STF**
<table>	Definisce una tabella per l'inserimento di dati tabulari.	**STF**
** **	Lista non ordinata. Può contenere solo elementi	**STF**

Altri elementi molto che possono essere considerati di tipo blocco, in quanto possono contenerne altri dello stesso tipo, sono:

Elemento	Descrizione	Supporto DTD
<dd>	Descrizione di un termine in una lista di definizione.	**STF**
<dt>	Definizione di un termine in una lista di definizione.	**STF**
<frameset>	Definisce un frameset.	**F**
** **	Elemento di una lista ordinata o non ordinata.	**STF**
<tbody>	Definisce il corpo di una tabella. Con <thead> e <tfoot> serve a raggruppare le righe di una tabella.	**STF**
<td>	Cella di tabella.	**STF**
<tfoot>	Definisce il "piede" di una tabella.	**STF**
<th>	Intestazione di cella.	**STF**
<thead>	Definisce la testata di una tabella.	**STF**
<tr>	Riga di tabella.	**STF**

Lista degli elementi inline XHTML

La lista elenca tutti gli elementi inline definiti nelle tre DTD XHTML 1.0. Di ciascuno viene fornita una breve descrizione.

La terza colonna specifica il supporto delle tre DTD:

S = Strict; T = Transitional; F = Frameset

Elemento	Descrizione	Supporto DTD
<a>	Definisce un'ancora o un collegamento (con l'attributo href).	**STF**
<abbr>	Usato per le abbreviazioni	**STF**
<acronym>	Specifica che il testo è un acronimo	**STF**
<b>	Formatta il testo in grassetto	**STF**
<basefont>	Definisce un font di base	**TF**
<bdo>	Definisce la direzione del testo da mostrare	**STF**
<big>	Ingrandisce il testo	**STF**
** **	Inserisce un'interruzione di riga	**STF**
<cite>	Usato per testi citati	**STF**
<code>	Formatta il testo come codice di computer	**STF**
<dfn>	Usato per i termini di definizioni	**STF**
<em>	Stile di testo simile al corsivo	**STF**
<font>	Formatta dimensioni, colori e tipi dei caratteri	**TF**
<i>	Testo in corsivo	**STF**
<img>	Inserisce un'immagine	**STF**
<input>	Inserisce elementi di un form. L'attributo type ne definisce la tipologia	**STF**
<label>	Definisce l'etichetta per l'elemento di un form	**STF**
<q>	Specifica una breve citazione all'interno di un paragrafo	**STF**
<s>	Testo cancellato. Sconsigliato in HTML 4.0	**TF**
<samp>	Simile a code	**STF**
<select>	Inserisce un menu a tendina in un form	**STF**
<small>	Rimpicciolisce una porzione di testo	**STF**
<span>	Definisce una sezione di testo inline cui applicare stili	**STF**
<strike>	Test cancellato	**TF**
<strong>	Definisce un testo rafforzato, simile al grassetto	**STF**
<sub>	Test sottoscritto	**STF**

<sup>	Testo soprascritto	**STF**
<textarea>	Inserisce un'area di testo modificabile	**STF**
<tt>	Testo con carattere monospazio	**STF**
<u>	Testo sottolineato	**TF**
<var>	Definisce una variabile	**STF**

Altri elementi possono essere di tipo inline ma anche blocco:

Elemento	Descrizione	Supporto DTD
<applet>	Inserisce un applet Java	**TF**
<button>	Inserisce un pulsante	**STF**
<del>	Testo cancellato	**STF**
<iframe>	Inserisce un iframe.	**TF**
<ins>	Specifica che il testo è stato inserito	**STF**
<map>	Mappa immagine cliccabile	**STF**
<object>	Inserisce un oggetto	**STF**
<script>	Definisce script di programmazione	**STF**

Appendice B

Variazioni apportate alle tabelle del database Campusone

Di seguito sono elencati i nuovi campi inseriti nelle tabelle per il corretto funzionamento della versione inglese del portale. e i relativi script SQL per modificare il database.

Nome tabella: TBL_FACOLTA_AA

NOME CAMPO	TIPO	DIMENSIONE
PresentazioneEng	VARCHAR	4000

Script SQL:

```
ALTER TABLE [dbo].[TBL_FACOLTA_AA] ADD
	[PresentazioneEng] [varchar] (4000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Nome tabella: TBL_RUOLO

NOME CAMPO	TIPO	DIMENSIONE
RouloEng	VARCHAR	100

Script SQL:

```
ALTER TABLE [dbo].[TBL_RUOLO] ADD
	[RuoloEng] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Nome tabella: TBL_COMMISSIONI_INTRANET

NOME CAMPO	TIPO	DIMENSIONE
DenominazioneEng	CHAR	256

Script SQL:

```
ALTER TABLE [dbo].[TBL_Commissioni_Intranet] ADD
	[DenominazioneEng] [char] (256) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Nome tabella: TBL_ORIENTAMENTI

NOME CAMPO	TIPO	DIMENSIONE
DenominazioneEng	VARCHAR	100
NoteAnnoEng1	VARCHAR	2000
NoteAnnoEng2	VARCHAR	2000

NoteAnnoEng3	VARCHAR	2000
ModalitaLaureaEng	VARCHAR	2000
NoteEng	VARCHAR	5000

Script SQL:

```
ALTER TABLE [dbo].[TBL_ORIENTAMENTI] ADD
        [DenominazioneEng] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [NoteAnnoEng1] [varchar] (2000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [NoteAnnoEng2] [varchar] (2000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [NoteAnnoEng3] [varchar] (2000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [ModalitaLaureaEng] [varchar] (2000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [NoteEng] [varchar] (5000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Nome tabella: TBL_CORSI_CATEGORIE

NOME CAMPO	TIPO	DIMENSIONE
DenominazioneEng	VARCHAR	100

Script SQL:

```
alter TABLE [dbo].[TBL_CORSI_CATEGORIE] add
        [DenominazioneEng] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Nome tabella: TBL_CORSI_STUDIO_REGOLAMENTO

NOME CAMPO	TIPO	DIMENSIONE
PresentazioneEng	VARCHAR	2500
DenominazioneEng	VARCHAR	100

Script SQL:

```
ALTER TABLE [dbo].[TBL_CORSO_STUDIO_REGOLAMENTO] ADD
        [DenominazioneEng] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [PresentazioneEng] [varchar] (2500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Nome tabella: TBL_INSEGNAMENTI

NOME CAMPO	TIPO	DIMENSIONE
DenominazioneEng	VARCHAR	200
ObiettiviEng	VARCHAR	2000
SvolgimentoEng	VARCHAR	2000
AccertamentoEng	VARCHAR	2000
ParoleChiaveEng	VARCHAR	500

NOTA: Il campo “EngName” viene eliminato perché sostituito da “DenominazioneEng”

Script SQL:

```
ALTER TABLE [dbo].[TBL_INSEGNAMENTI] ADD
	[DenominazioneEng] [varchar] (200) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[ObiettiviEng] [varchar] (2000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[SvolgimentoEng] [varchar] (2000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[AccertamentoEng] [varchar] (2000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[ParoleChiaveEng] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL

ALTER TABLE [dbo].[TBL_INSEGNAMENTI] DROP
	[EngName]
```

Nome tabella: TBL_INSEGNAMENTI_AA

NOME CAMPO	TIPO	DIMENSIONE
ProgrammaEng	VARCHAR	8000
TestiConsigliatiEng	VARCHAR	2000
NoteEng	VARCHAR	2000

Script SQL:

```
ALTER TABLE [dbo].[TBL_INSEGNAMENTI_AA] ADD
	[ProgrammaEng] [varchar] (8000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[TestiConsigliatiEng] [varchar] (2000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[NoteEng] [varchar] (2000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Nome tabella: TBL_TIPO_ACCESSO

NOME CAMPO	TIPO	DIMENSIONE
DescrizioneEng	VARCHAR	100

Script SQL:

```
ALTER TABLE [dbo].[TBL_TIPO_ACCESSO] ADD
	[DescrizioneEng] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Nome tabella: TBL_GIORNI

NOME CAMPO	TIPO	DIMENSIONE
GiorniEng	VARCHAR	50

Script SQL:

```
ALTER TABLE [dbo].[TBL_GIORNI] ADD
	[GiornoEng] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Nome tabella: TBL_PERIODI

NOME CAMPO	TIPO	DIMENSIONE
DenominazioneEng	VARCHAR	100
NoteEng	VARCHAR	500

NOTA: I campio “DenominazioneIng” e “NoteIng” vengono eliminati perché sostituiti rispettivamente da “DenominazioneEng” e “NoteEng”

Script SQL:

```
ALTER TABLE [dbo].[TBL_PERIODI] ADD
	[DenominazioneEng] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[NoteEng] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,

ALTER TABLE [dbo].[TBL_PERIODI] DROP
	[DenominazioneIng] ,
	[NoteIng]
```

Nome tabella: TBL_AULE

NOME CAMPO	TIPO	DIMENSIONE
PianoEng	CHAR	10
UbicazioneEng	CHAR	500

Script SQL:

```
ALTER TABLE [dbo].[TBL_AULE] ADD
	[PianoEng] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[UbicazioneEng] [char] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
```

Select e viste introdotte nel database Campusone

Di seguito sono elencate le select e le viste introdotte per il funzionamento di alcune pagine dinamiche in lingua inglese. Queste pagine sono molto complesse e per reperire il flusso di dati che visualizzano si sono dovute creare viste o select ad hoc.

1. CorsoStudio-e.asp

E' stata creata una stored procedure chiamata "SELECT_LAUREA_ENG" per il corretto funzionamento della pagina CorsoStudio-e.asp.

Script SQL:

```
CREATE PROCEDURE SELECT_LAUREA_ENG


@IDCsr int,
@IDCategoria int

AS

SELECT     TA.DescrizioneEng AS Descrizione, C.DenominazioneEng AS NomeCategoria, CAA.*,
FAA.IDFacoltaAA AS Expr2, F.IndirizzoFacolta AS Expr3, F.Web AS Expr4,
                  F.UltimoAggiornamento AS Expr5, F.Denominazione AS DenominazioneFacolta, CSR.*,
AA.Denominazione AS DenominazioneAnno,
                  TBL1.Nome AS Nome1, TBL1.Cognome AS Cognome1, TBL1.Telefono AS Telefono1,
TBL1.Fax AS Fax1, TBL1.Email AS Email1, TBL2.Nome AS Nome2,
                  TBL2.Cognome AS Cognome2, TBL2.Telefono AS Telefono2, TBL2.Fax AS Fax2, TBL2.Email
AS Email2, TBL3.Nome AS Nome3,
                  TBL3.Cognome AS Cognome3, TBL3.Telefono AS Telefono3, TBL3.Fax AS Fax3, TBL3.Email
AS Email3, CC.ClasseMIUR AS ClasseMIUR,
                  CC.DenominazioneEng AS NomeClasse
FROM         TBL_CORSO_STUDIO_REGOLAMENTO CSR LEFT OUTER JOIN
                  TBL_FACOLTA F ON CSR.IDFacolta = F.Id LEFT OUTER JOIN
                  TBL_FACOLTA_AA FAA ON F.Id = FAA.IDFacolta LEFT OUTER JOIN
                  TBL_CORSI_AA CAA ON CSR.Id = CAA.IdCorsoStudioRegolamento LEFT OUTER JOIN
                  TBL_DOCENTI TBL1 ON CAA.IdReferente = TBL1.Id LEFT OUTER JOIN
                  TBL_DOCENTI TBL2 ON CAA.IdInformazioni = TBL2.Id LEFT OUTER JOIN
                  TBL_DOCENTI TBL3 ON CAA.IdDelegato = TBL3.Id LEFT OUTER JOIN
                  TBL_ANNI_ACCADEMICI AA ON CSR.IdAnnoDiRiferimento = AA.Id LEFT OUTER JOIN
                  TBL_TIPO_ACCESSO TA ON CSR.IDAccesso = TA.ID LEFT OUTER JOIN
                  TBL_CORSI_CATEGORIE C ON CSR.IdCategoria = C.Id LEFT OUTER JOIN
                  TBL_CORSI_CLASSI CC ON CC.Id = CSR.IdClasse
 WHERE CSR.Id = @IDCsr
  AND CSR.DELETED = 0
  AND CSR.IDCategoria = @IDCategoria
  AND  FAA.IDAnnoRiferimento = CSR.IDAnnoDiRiferimento
  AND CSR.CompilazioneUltimata = 1
  AND FAA.PubblicazioneWEB = 1
GO
```

2. Ricerca_ins-e.asp

E' stata creata una vista chiamata "VIEW_INSEGNAMENTI_CON_MUTUAZIONI_ENG" per il corretto funzionamento della pagina Ricerca_ins-e.asp.

Script SQL:

```
CREATE VIEW dbo.VIEW_INSEGNAMENTI_CON_MUTUAZIONI_ENG
AS
SELECT     I.Id AS IdInsegnamento, I.Id AS IdPadre, I.Denominazione AS NomeInsegnamento,
I.IdCorsoStudioRegolamento AS IdCSR, I.AnnoCorso, I.Crediti,
                  I.Deleted, I.Disattivato, I.DenominazioneEng  AS NomeInsegnamentoEng
FROM         dbo.TBL_INSEGNAMENTI I INNER JOIN
                  dbo.TBL_INSEGNAMENTI_AA IAA ON I.Id = IAA.IdInsegnamento
WHERE     I.Id NOT IN
                      (SELECT     IDInsegnamentoMutuato
                       FROM         TBL_MUTUA_DA) AND I.Deleted = 0
UNION
SELECT     I2.Id AS IdInsegnamento, M.IdInsegnamentoAA AS IdPadre, I2.Denominazione AS
NomeInsegnamento, I2.IdCorsoStudioRegolamento AS IdCSR,
                  I2.AnnoCorso, I2.Crediti, I2.Deleted, I2.Disattivato, I2.DenominazioneEng AS
NomeInsegnamentoEng
FROM         dbo.TBL_INSEGNAMENTI I, TBL_MUTUA_DA M, dbo.TBL_INSEGNAMENTI_AA IAA,
TBL_INSEGNAMENTI I2
WHERE     I.Id = M.IdInsegnamentoAA AND I.Id = IAA.IdInsegnamento AND M.IdInsegnamentoMutuato =
I2.Id AND I.Deleted = 0 AND I2.Deleted = 0
```

3. VisualizzaOrario-e.asp

Sono state create due viste chiamate "VIEW_INS_ORARI_ENG" e "VIEW_INSEGNAMENTI_PERIODO4_ENG" per il corretto funzionamento della pagina VisualizzaOrario-e.asp.

Script SQL VIEW_INS_ORARI_ENG**:**

```
CREATE VIEW dbo.VIEW_INS_ORARI_ENG
AS
SELECT     dbo.VIEW_INSEGNAMENTI_PERIODO4_ENG.IdInsegnamento,
dbo.VIEW_INSEGNAMENTI_PERIODO4_ENG.IdPadre,
dbo.VIEW_INSEGNAMENTI_PERIODO4_ENG.NomeInsegnamento,
                  dbo.VIEW_INSEGNAMENTI_PERIODO4_ENG.AnnoCorso,
dbo.VIEW_INSEGNAMENTI_PERIODO4_ENG.IDOrientamento, dbo.TBL_INS_ORARIO.Id_Aula,
                  dbo.TBL_AULE.Descrizione AS Aula, dbo.TBL_INS_ORARIO.Id_Anno,
dbo.TBL_INS_ORARIO.Id_Periodo, dbo.TBL_GIORNI.Id AS IdGiorno,
                  dbo.TBL_GIORNI.GiornoEng, dbo.TBL_ORE.Id AS IdOra, dbo.TBL_ORE.Ora,
dbo.TBL_INS_ORARIO.Id_CDS
FROM         dbo.VIEW_INSEGNAMENTI_PERIODO4_ENG INNER JOIN
                  dbo.TBL_INS_ORARIO ON dbo.VIEW_INSEGNAMENTI_PERIODO4_ENG.IdPadre
= dbo.TBL_INS_ORARIO.Id_Ins LEFT OUTER JOIN
```

```
                dbo.TBL_AULE ON dbo.TBL_INS_ORARIO.Id_Aula = dbo.TBL_AULE.Id RIGHT
OUTER JOIN
                dbo.TBL_ORE ON dbo.TBL_INS_ORARIO.Id_Ora = dbo.TBL_ORE.Id RIGHT
OUTER JOIN
                dbo.TBL_GIORNI ON dbo.TBL_INS_ORARIO.Id_Giorno = dbo.TBL_GIORNI.Id
```

Script SQL VIEW_INSEGNAMENTI_PERIODO4_ENG **:**

```
CREATE VIEW dbo.VIEW_INSEGNAMENTI_PERIODO4_ENG
AS
SELECT     I.Id AS IdInsegnamento, I.Id AS IdPadre, I.DenominazioneEng  AS
NomeInsegnamento, I.AnnoCorso, OIAA.AScelta, I.Crediti, I.Asterisco,
                O.Id AS IDOrientamento, O.Denominazione AS NomeOrientamento,
O.IdCorsoStudioRegolamento AS IdCsr, P.Id AS IDPeriodo,
                P.Denominazione AS NomePeriodo, P.DataFine, P.DataInizio, O.Note, I.Deleted,
I.Disattivato
FROM        dbo.TBL_INSEGNAMENTI I INNER JOIN
                dbo.TBL_INSEGNAMENTI_AA IAA ON I.Id = IAA.IdInsegnamento INNER JOIN
                dbo.TBL_INSEGNAMENTI_AA_PERIODI IAAP ON IAA.Id =
IAAP.IdInsegnamentoAA INNER JOIN
                dbo.TBL_PERIODI P ON IAAP.IdPeriodo = P.Id INNER JOIN
                dbo.TBL_ORIENTAMENTI_INSEGNAMENTI_AA OIAA ON I.Id =
OIAA.IdInsegnamento INNER JOIN
                dbo.TBL_ORIENTAMENTI O ON OIAA.IdOrientamento = O.Id AND I.Id NOT IN
                   (SELECT     IDInsegnamentoMutuato
                     FROM          TBL_MUTUA_DA)
WHERE     I.Deleted = 0
UNION
SELECT     I2.Id AS IdInsegnamento, M.IdInsegnamentoAA AS IdPadre, I2.DenominazioneEng
AS NomeInsegnamento, I2.AnnoCorso, OIAA2.AScelta, I2.Crediti,
                I2.Asterisco, O2.Id AS IDOrientamento, O2.Denominazione AS NomeOrientamento,
O2.IdCorsoStudioRegolamento AS IdCsr, P.Id AS IDPeriodo,
                P.Denominazione AS NomePeriodo, P.DataFine, P.DataInizio, O2.Note, I.Deleted,
I.Disattivato
FROM        dbo.TBL_INSEGNAMENTI I, TBL_MUTUA_DA M, dbo.TBL_INSEGNAMENTI_AA
IAA, dbo.TBL_INSEGNAMENTI_AA_PERIODI IAAP, dbo.TBL_PERIODI P,
                dbo.TBL_ORIENTAMENTI_INSEGNAMENTI_AA OIAA, dbo.TBL_ORIENTAMENTI
O, TBL_INSEGNAMENTI I2, dbo.TBL_INSEGNAMENTI_AA IAA2,
                dbo.TBL_ORIENTAMENTI_INSEGNAMENTI_AA OIAA2,
dbo.TBL_ORIENTAMENTI O2
WHERE     I2.Id = IAA2.IdInsegnamento AND IAA.Id = M.IdInsegnamentoAA AND I.Id =
IAA.IdInsegnamento AND IAA.Id = IAAP.IdInsegnamentoAA AND
                IAAP.IdPeriodo = P.Id AND I.Id = OIAA.IdInsegnamento AND OIAA.IdOrientamento
= O.Id AND M.IdInsegnamentoMutuato = I2.Id AND
                I2.Id = OIAA2.IdInsegnamento AND OIAA2.IdOrientamento = O2.Id AND I.Deleted =
0 AND I2.Deleted = 0
```

4. Manifesto-e.asp

E’ stata creata una vista “VIEW_INSEGNAMENTI_PERIODI2_ENG” per il corretto funzionamento della pagina Manifesto-e.asp.

Script SQL:

```
CREATE VIEW dbo.VIEW_Insegnamenti_Periodi2_Eng
AS
SELECT     I.Id AS IdInsegnamento, I.DenominazioneEng  AS NomeInsegnamento, I.AnnoCorso,
OIAA.AScelta, I.Crediti, O.Id AS IDOrientamento,
                  O.DenominazioneEng AS NomeOrientamento, O.IdCorsoStudioRegolamento AS
IdCsr, P.Id AS IDPeriodo, P.DenominazioneEng AS NomePeriodo, P.DataFine,
                  P.DataInizio, O.NoteEng, O.ModalitaLaureaEng, I.Deleted, I.ParoleChiaveEng,
IAA.MaterialeDidattico, I.Asterisco, I.Disattivato
FROM         dbo.TBL_INSEGNAMENTI I INNER JOIN
                  dbo.TBL_INSEGNAMENTI_AA IAA ON I.Id = IAA.IdInsegnamento INNER JOIN
                  dbo.TBL_INSEGNAMENTI_AA_PERIODI IAAP ON IAA.Id =
IAAP.IdInsegnamentoAA INNER JOIN
                  dbo.TBL_PERIODI P ON IAAP.IdPeriodo = P.Id INNER JOIN
                  dbo.TBL_ORIENTAMENTI_INSEGNAMENTI_AA OIAA ON I.Id =
OIAA.IdInsegnamento INNER JOIN
                  dbo.TBL_ORIENTAMENTI O ON OIAA.IdOrientamento = O.Id AND IAA.Id NOT IN
                     (SELECT     IDInsegnamentoMutuato
                       FROM          TBL_MUTUA_DA)
WHERE     I.Deleted = 0
UNION
SELECT     I2.Id AS IdInsegnamento, I2.DenominazioneEng AS NomeInsegnamento,
I2.AnnoCorso, OIAA2.AScelta, I2.Crediti, O2.Id AS IDOrientamento,
                  O2.DenominazioneEng AS NomeOrientamento, O2.IdCorsoStudioRegolamento AS
IdCsr, P.Id AS IDPeriodo, P.DenominazioneEng AS NomePeriodo, P.DataFine,
                   P.DataInizio, O2.NoteEng, O2.ModalitaLaureaEng, I.Deleted, I2.ParoleChiaveEng,
IAA.MaterialeDidattico, I2.Asterisco, I2.Disattivato
FROM         dbo.TBL_INSEGNAMENTI I, TBL_MUTUA_DA M, dbo.TBL_INSEGNAMENTI_AA
IAA, dbo.TBL_INSEGNAMENTI_AA_PERIODI IAAP, dbo.TBL_PERIODI P,
                  dbo.TBL_ORIENTAMENTI_INSEGNAMENTI_AA OIAA, dbo.TBL_ORIENTAMENTI
O, TBL_INSEGNAMENTI I2,
                  dbo.TBL_ORIENTAMENTI_INSEGNAMENTI_AA OIAA2,
dbo.TBL_ORIENTAMENTI O2, dbo.TBL_INSEGNAMENTI_AA IAA2
WHERE     IAA.Id = M.IdInsegnamentoAA AND I.Id = IAA.IdInsegnamento AND IAA.Id =
IAAP.IdInsegnamentoAA AND IAAP.IdPeriodo = P.Id AND
                  I.Id = OIAA.IdInsegnamento AND OIAA.IdOrientamento = O.Id AND
M.IdInsegnamentoMutuato = IAA2.Id AND I2.Id = OIAA2.IdInsegnamento AND
                  OIAA2.IdOrientamento = O2.Id AND I.Deleted = 0 AND I2.Deleted = 0 AND I2.Id =
IAA2.IdInsegnamento
```

Come ultima nota ricordiamo che diverse tabelle non hanno la form per l'inserimento dei dati via Web. Questo perché la frequenza di aggiornamento è molto bassa, anche una volta ogni 2 anni. I dati contenuti in tali tabelle dovranno quindi essere modificati manualmente tramite il software per la gestione del database, Enterprise Manager.
Le tabelle sono:

- TBL_AULE
- TBL_COMMISSIONI_INTRANET
- TBL_CORSI_CATEGORIE
- TBL_RUOLO
- TBL_TIPO_ACCESSO
- TBL_CONFIGURAZIONE_VIEW
- TBL_CORSI_CLASSI