

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SEDE DI CESENA

SECONDA FACOLTÀ DI INGEGNERIA CON SEDE A CESENA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

HTML 5

Evoluzione del markup language per il Web

Elaborato in

Sistemi Distribuiti L-A

Relatore

Andrea Omicini

Presentata da

Simone Dozza

Sessione numero: 3

Anno Accademico: 2009/2010

1. INTRODUZIONE

HTML5 sta per fare il suo ingresso nel mondo del web andando a sostituire quello che sarà fra circa due anni il suo predecessore, ovvero HTML4.

Ma come cambierà la progettazione e il markup delle web pages con il suo avvento?

Innanzitutto, dalle prime righe, si può facilmente dedurre che il seguente elaborato andrà poi riveduto in futuro visto che si basa su una versione ancora non definitiva del linguaggio in esame.

Già a questo punto dello sviluppo di HTML 5 però si possono trarre importanti osservazioni su quali saranno le principali novità e miglioramenti che esso porterà rispetto al suo predecessore.

In questo elaborato introdurrò il nuovo HTML, ne descriverò le caratteristiche e mi soffermerò sulle novità presenti fra queste, nonché sulla loro importanza.

1.1 *Compatibilità*

La domanda che sorge spontanea è sicuramente come sia possibile analizzare un linguaggio che ancora, non essendo definitivo, non può di certo godere dell'opportuno livello di compatibilità per quanto riguarda i maggiori browser web sul mercato quali Firefox, Chrome ed Explorer. Principalmente questo è possibile in quanto il nuovo HTML non va considerato come un'unica grande "cosa" indivisibile ma come un insieme di features e in quanto tale non si può ricercare una compatibilità riferita ad esso in modo esteso bensì bisognerà analizzare per ogni diversa feature se questa è compatibile o no a seconda del browser. Già da questo preciso istante è quindi possibile testare il nuovo HTML in quanto per ognuna di esse (ad es. canvas, video o geolocation...) è possibile trovare un supporto compatibile.

In generale possiamo dire che HTML5, alla luce anche di quanto detto sopra, è già ora ben supportato dai principali Browser Web.

Rimanendo sull'argomento compatibilità ma spostando il nostro punto di vista dalla parte opposta, va assolutamente specificato che le applicazioni web che funzionano ora sfruttando HTML4 continueranno a funzionare in futuro con l'avvento di HTML5. Il programmatore non dovrà imparare nuovamente cose che già conosce o buttare via il suo vecchio markup, ma dovrà impegnarsi nell'imparare come sfruttare le nuove opzioni a sua disposizione. HTML5 sarà totalmente retro compatibile con

HTML4 e si limiterà al massimo a deprecare qualche soluzione che invece nel suo predecessore veniva utilizzata senza problemi o non aveva valide alternative che permettessero di evitarne l'utilizzo.

1.2 Primi accenni alle novità

Le prime cose alle quali si pensa quando si parla di HTML sono sicuramente i tags e le parentesi angolari. Sempre analizzando il tutto a grandi linee, prima di entrare in tali questioni più approfonditamente in seguito, queste ricopriranno sicuramente anche in HTML 5 una parte di fondamentale importanza, con la differenza che in questa nuova versione si definisce più accuratamente il modo con il quale esse interagiscono insieme a JavaScript, attraverso un documento denominato DOM (Document Object Model), già presente in HTML4 ma notevolmente esteso. Proseguendo con la lettura vi accorgerete come JavaScript ricopra un ruolo molto importante fra le diverse novità introdotte dalla W3C.

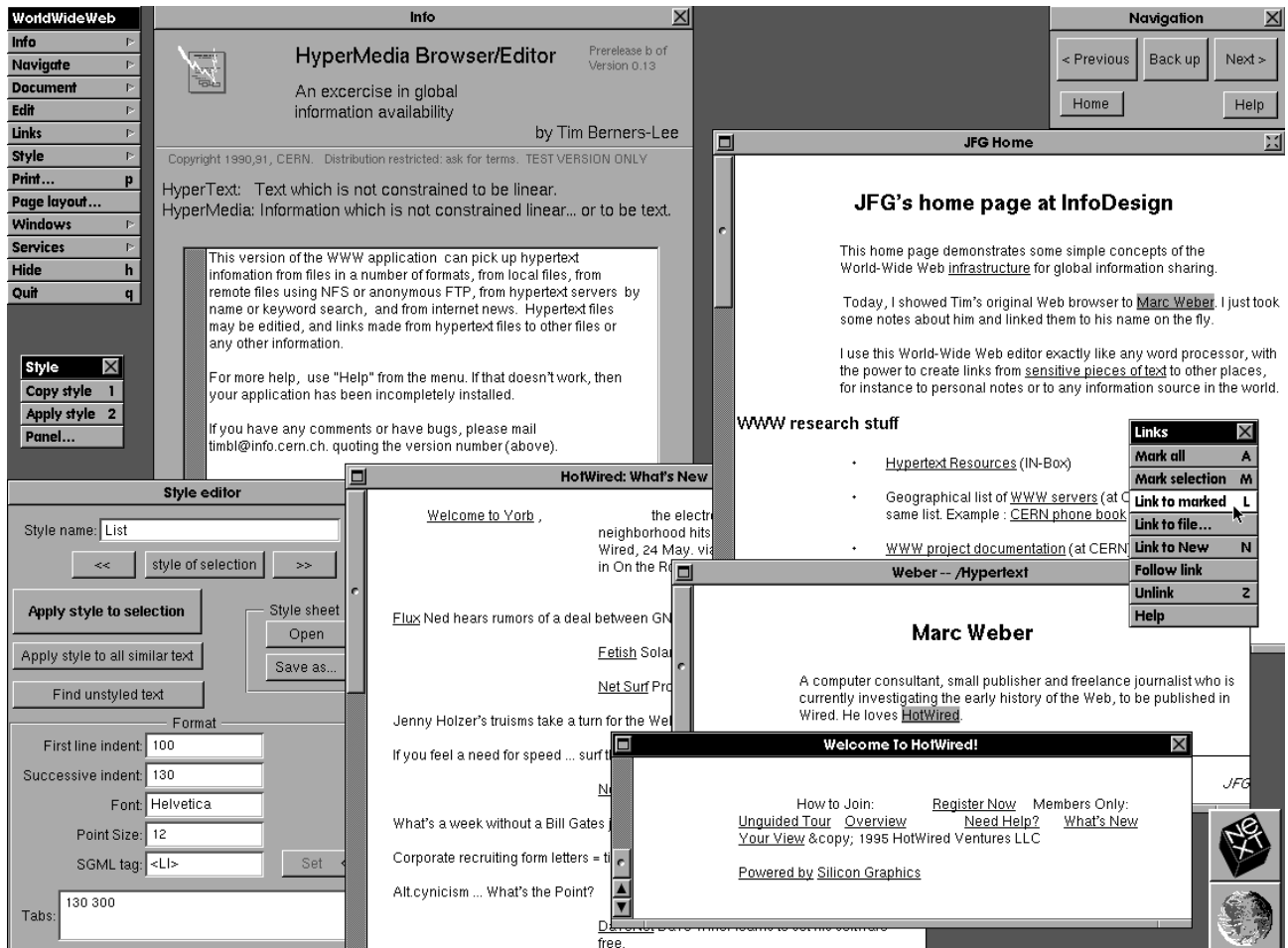
Portando un esempio alla vostra attenzione consideriamo il tag `<video>`: esso non viene solo definito in quanto tag ma ad esso viene fatto corrispondere anche un DOM API per oggetti video all'interno del DOM. Si può ad esempio sfruttare tale API per rilevare quali format video sono supportati, per dare i vari comandi play, pausa, audio muto, ma anche per tracciare a che punto è un determinato download del video in esame, insomma qualsiasi cosa possa risultare utile per fornire un'esperienza completa e soddisfacente per quanto concerne questo tipo di tag.

Proprio avendo un occhio di riguardo rispetto ai tags ci accorgeremo come l'aggiunta di molti di essi rispecchi le nuove esigenze venutesi a creare con l'evoluzione del mondo del web negli ultimi anni.

1.3 Cenni storici

Presentando gli accadimenti che ci hanno portato a questo punto non possiamo evitare di parlare di Tim Berners-Lee, un uomo che non solo è stato il principale artefice della nascita di HTML (HyperText Markup Language) ma anche di quella del WWW (World Wide Web), HTTP (HyperText Transfer Protocol) e URLs (Universal Resource Locators), il tutto sviluppato nel lasso di tempo compreso fra il 1989 e il 1991.

In questo periodo la sede centrale delle operazioni era il CERN di Ginevra. Nello screenshot di seguito possiamo ammirare uno dei primi Browser Editor sviluppati da Berners-Lee per lo sviluppo di codice inerente la prima versione di HTML, ufficialmente rilasciato nel 1993.



Girava su un computer NeXT ed era implementato in linguaggio Objective-C. Di seguito possiamo vedere proprio il modello utilizzato da Berners-Lee, famoso per essere stato il primo web server della storia.



Con lo scopo di guidare lo sviluppo del Web e definirne gli standard, Tim Berners-Lee fondò nel 1994 il W3C, ovvero il World Wide Web Consortium.

Quell'anno nacque HTML 2, la prima versione a raggiungere un certo livello di notorietà. Fra le principali novità vi è l'introduzione dei forms.

Nel 1995 il W3C fu costretto a rivedere la versione 3.0, sviluppata implementando moltissime nuove features (come le tabelle) senza neanche aver avuto un pieno consenso, o non prendendo neanche in considerazione altre caratteristiche quali il supporto per la matematica. I browser non avrebbero mai supportato pienamente HTML 3 così si decise di svilupparne una revisione utilizzando un approccio più pragmatico, concentrandosi sulla generazione di un suo sottoinsieme capace però di avere un pieno consenso e andare incontro anche alle esigenze dei produttori.

Nel 1997 uscì quindi HTML 3.2: includeva tabelle, applet, script e altre notevoli migliorie, ma non i frame.

Seguì HTML 4 nel 1998, poi leggermente corretto e rivisitato con l'uscita l'anno seguente di HTML 4.01.

L'HTML 4 estende le possibilità dell'HTML con meccanismi per i fogli di stile, lo scripting, i frame, gli oggetti incorporati, con un miglior supporto per la scrittura da destra verso sinistra e per quella a direzione mista, con tabelle più ricche e miglioramenti ai moduli. Il suo sviluppo si è svolto inoltre con un notevole occhio di riguardo rispetto alle esigenze delle persone affette da disabilità, alle quali si offre una maggiore accessibilità.

HTML 4 è anche sinonimo di una nuova internazionalizzazione grazie all'incorporazione del "Internationalization of the HyperText Markup Language" che rende possibile la scrittura dei documenti in qualsiasi lingua e ne permette così una piena trasportabilità in tutto il mondo.

L'HTML offre ora un maggior supporto per le diverse lingue naturali usate all'interno di un documento. Ciò consente una più efficace indicizzazione dei documenti da parte dei motori di ricerca, una qualità tipografica superiore, una migliore conversione da testo a parlato e molto altro ancora.

Da qui in poi la nascita di HTML 5 non è immediata. W3C dichiara infatti nel 1999 che non intende proseguire con lo sviluppo di HTML ma vuole dedicarsi interamente alla creazione di un suo equivalente basato su XML. Nascerà così nel 2000 XHTML 1.0. Nel frattempo parte delle API di HTML create dai produttori di Browser vengono pubblicate sotto il nome di DOM, documento organizzato su diversi livelli.

La scintilla che riaprirà i discorsi relativi ad una nuova versione di HTML è la pubblicazione di XForms, una tecnologia percepita da tutti come il futuro dei Web Forms, insieme alla realizzazione che XML come tecnologia Web non potrà mai rimpiazzare HTML bensì potrà solo limitare il suo raggio d'azione verso una migliore gestione delle nuove tecnologie come RSS prima e Atom poi. Inoltre ben presto ci si accorge che la maggior parte delle novità introdotte da XForms si possono fornire anche estendendo i forms di HTML 4, con tutti i vantaggi del caso per quanto riguarda soprattutto i browser e relativa compatibilità con le nuove features.

Nonostante tutto, nel 2004 l'idea di partire con lo sviluppo di HTML5 viene rifiutata dalla W3C, così i lavori continuano sotto l'egida di Apple, Mozilla ed Opera che fondano il WHATWG.

Punti fermi del nuovo HTML saranno individuati nei principi di retro compatibilità, combaciamento fra specifiche e implementazioni, livello di dettaglio delle specifiche tale da garantire il raggiungimento dell'interoperabilità fra le diverse implementazioni possibile senza la necessità di una decompilazione.

W3C nel 2007 forma un working group e decide di unirsi a WHATWG nello sviluppo di HTML5, quindi ottiene di pubblicare le specifiche di questa nuova versione sotto il proprio copyright.

WHATWG mantiene ovviamente una versione delle specifiche di HTML sul suo sito, inizialmente leggermente diverso in quanto comprendente un numero maggiore di caratteristiche.

2. LE SPECIFICHE

In questo capitolo riassumerò le caratteristiche principali di HTML 5, cercando di dare un'idea di base riguardo alle sue features e concentrandomi sulla loro spiegazione e sulle possibilità a noi fornite dal loro utilizzo.

Non mi soffermerò solo su quelle che sono le novità ma prenderò in considerazione HTML nella sua interezza prima di procedere nel prossimo capitolo ad approfondire quelle che sono o potranno diventare interessanti aggiunte rispetto a quello che già avevamo in mano nel precedente HTML 4.

Aggiungo anche che mi concentrerò maggiormente sul linguaggio vero e proprio pur trattando, in un primo paragrafo, il DOM HTML, il quale non poteva certo essere tralasciato completamente vista la sua importanza nella gestione dei documenti HTML.

2.1 DOM

2.1.1 Introduzione

DOM, ovvero Document Object Model, è una rappresentazione dei documenti la cui struttura è orientata agli oggetti in essi presenti.

Può anche essere definito come un API, in quanto in esso sono raccolte un insieme di funzioni, metodi e proprietà che i programmatori di pagine Web e non solo possono richiamare per accedere e manipolare tutti gli elementi delle pagine stesse.

Il DOM risulta indipendente dalla piattaforma sulla quale viene richiamato e graficamente si può immaginare come un albero i cui nodi sono rappresentati da praticamente ogni cosa presente nel testo in esame. I nodi possono a loro volta contenerne altri e in questo caso parliamo di essi come elementi della pagina.

Per esempio, prendendo in esame una semplice porzione di pagina web:

```
<html>
  <body>
    <a href="pagina.html">Home</a>
  </body>
</html>
```

Analizzandola arriveremo ad un albero del tipo:

HTML ----- BODY ----- A ----- "HOME"

Nel DOM vi è sempre un elemento “document” che contiene tutti gli altri e, di conseguenza, anche questo elemento corrisponde ad un tag.

Questo tag ovviamente sarà, all’interno dell’albero, nodo padre per tutti gli altri ed identificherà la pagina Web.

2.1.2 Metodi di document

Mi occuperò solo dei principali, coerentemente con quanto detto all’inizio di questo capitolo.

- getElementById()

restituisce il riferimento all’elemento che all’interno del documento è caratterizzato da un valore dell’attributo ID uguale a quello che passiamo come parametro alla funzione:

document.getElementById (ID_elemento)

- getElementsByTagName()

in questo caso otteniamo un array con l’insieme degli elementi caratterizzati dal tag il cui nome specifico come parametro alla funzione. Gli elementi dell’array sono poi facilmente riferibili tramite la solita notazione *array[indice]*.

document.getElementsByTagName (nome_tag)

- createElement()

creiamo un qualunque tipo di elemento di cui la funzione ci ritorna un riferimento.

Il nome del nuovo elemento viene passato come parametro:

nuovo_elemento = document.createElement (nome_tag)

- createTextNode()

con questa funzione creiamo un nodo di testo. Come forse si sarà già notato dal precedente paragrafo di introduzione, i testi sono particolari nodi che non possono contenerne altri e, graficamente, considerando il DOM come un albero, rappresenterebbero le sue foglie.

Questa funzione ritorna un riferimento al nuovo nodo di testo e richiede in ingresso la stringa che in esso vogliamo inserire:

nuovo_testo = document.createTextNode (testo_da_inserire)

2.1.3 Metodi per gli elementi

Per elemento si intendono tutti i tag in generale, ovvero quei nodi nell'albero del DOM che contengono altri nodi.

Come detto prima il nodo principale di un documento è il tag “document” ma ci sono anche altri tags il cui ordine all'interno dell'albero DOM viene standardizzato dando a loro un significato ben preciso:

- `<html>`: al suo interno si definisce il documento HTML
- `<head>`: le informazioni di intestazione
- `<title>`: il titolo in cima alla finestra di browser
- `<body>`: il corpo del documento HTML

Per completezza ricordo che non è necessario che questi tags ci siano tutti, questo voleva essere solo un accenno ai principali elementi che in ordine compongono le parti fondamentali di un documento HTML.

Tornando ai metodi relativi agli elementi, questi per lo più permettono di gestirli e modificarne le caratteristiche e comprendono anche funzioni relative ai loro attributi.

- `getElementsByTagName`

È lo stesso metodo visto per document solo che in questo caso viene applicato ai sotto elementi del tag in esame. Il suo utilizzo risulta quindi possibile ad ogni livello dell'albero DOM per ricavare l'array degli elementi contenuti dal tag specificato come parametro di ingresso.

- `setAttribute`

Con questo metodo si associa un nuovo attributo all'elemento specificato o si sovrascrive il valore di quello vecchio se l'attributo che si vuole aggiungere è già presente nel tag a cui ci riferiamo.

element.setAttribute (nome_attributo, valore_attributo)

I parametri di ingresso sono entrambi delle stringhe e specificano il nome ed il valore dell'attributo da aggiungere.

- `getAttribute`

Con questo metodo ricaviamo dall'elemento il valore dell'attributo il cui nome specifichiamo in ingresso:

element.getAttribute (nome_attributo)

- `removeAttribute`

Con questo metodo si rimuove l'attributo passato come parametro; se l'attributo ha un valore di default, l'elemento assumerà tale valore:

`element.removeAttribute (nome_attributo)`

In generale sia per quanto riguarda il “document” sia per quanto riguarda gli “elements”, ci sono metodi per ricavare il valore o i valori rispettivamente degli elementi che lo compongono e dell'elemento stesso.

Ad esempio, riprendendo i tag sopra definiti, possiamo avere:

- `document.body`: ritorna l'elemento `body` del documento se presente oppure `NULL` in caso contrario;
- `document.html`: come sopra solo che riferito al tag “html”
- etc.

Oppure per quanto riguarda il singolo elemento possiamo avere:

- `element.tagName`: ritorna il nome del tag dell'elemento associato.

2.1.4 Metodi per i nodi

Un nodo si può intendere come qualsiasi cosa presente all'interno del documento in quanto, mentre per gli elementi si richiedeva che questi potessero avere dei sotto elementi, un nodo è anche quell'elemento, come una stringa di testo o un commento, che rappresenta l'oggetto di base e più basso nella gerarchia di ogni albero DOM.

Con i seguenti metodi potremo sia muoverci che modificare il DOM tree, quindi rappresentano le armi fondamentali per poter agire direttamente sulla struttura di base di ogni documento a nostra disposizione.

- `childNodes`

Ritorna un array contenente tutti i nodi figli al nodo a cui è applicato. Se quest'ultimo non ne ha ritornerà un array vuoto.

`node.childNodes`

Una volta ricavato l'array possiamo agire su di esso come al solito, utilizzando la sua proprietà “length” per ottenerne la lunghezza oppure muovendoci al suo interno tramite il tipico indice fra parentesi quadre.

- `firstChild` e `lastChild`

con questi due metodi prendiamo rispettivamente primo e ultimo figlio del nodo a cui vengono applicati che equivale a dire il primo e l'ultimo nodo presenti nell'array ottenuto con il precedente metodo:

`node.firstChild` $\rightarrow$ `node.childNodes[0]`

`node.lastChild` $\rightarrow$ `node.childNodes[length - 1]`

In entrambi i casi, se non sono presenti figli il risultato sarà NULL.

- `nextSibling` e `previousSibling`

Restituiscono rispettivamente nodo successivo e precedente rispetto a quello su cui viene applicato questo metodo. Questa parte vi sarà più chiara guardando l'immagine alla fine di questa disamina. La sintassi è uguale a quella dei precedenti metodi, ovvero “node.” seguito dal nome del metodo. Nel caso non vi siano altri nodi “fratelli” al nostro, il risultato sarà NULL.

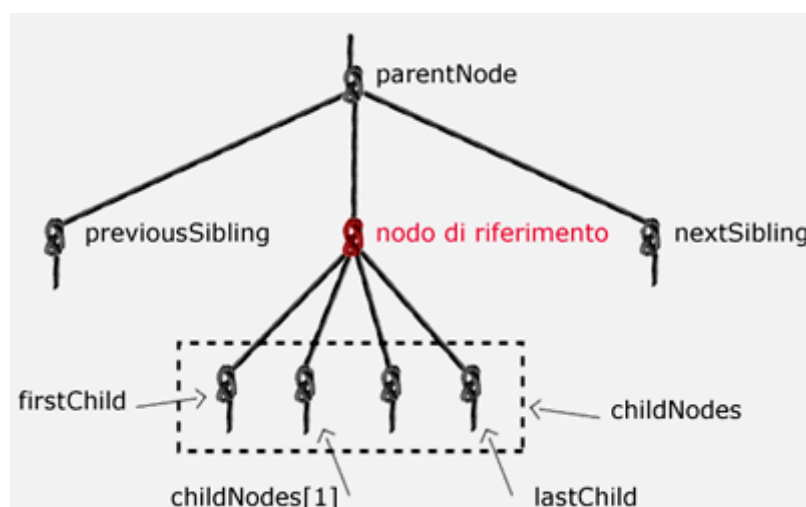
- `parentNode`

Ritorna il nodo che contiene quello a cui stiamo applicando il metodo. È interessante notare che oltre a “document” tutti gli altri nodi inseriti nella struttura della pagina hanno sicuramente un padre che al massimo sarà “document” stesso.

Ho specificato non a caso che ci stiamo riferendo a nodi inseriti nella struttura del documento in quanto se questo non fosse vero essi saranno considerati nodi a parte, senza figli e senza padre.

Nel caso il metodo ritorna NULL.

Di seguito un immagine che renderà tutti questi metodi molto più chiari:



- `nodeValue`

Restituisce il valore del nodo. Ritorna NULL nel caso venga applicato ad un tag, ritorna il testo se applicato ad un nodo di testo. In quest'ultimo caso permette anche di rimpiazzare il testo del nodo una volta ricavato.

Per ora abbiamo trattato i principali metodi di navigazione all'interno dell'albero di un documento, vediamo ora come agire su di esso:

- `appendChild`

Inserisce un nodo figlio al termine della lista dei figli del nodo a cui è applicato

`node.appendChild(nodo)`

- `insertBefore` o `replaceChild`

Con questi due metodi si decide rispettivamente di inserire un figlio prima di un determinato altro figlio del nodo a cui il metodo viene applicato oppure di inserirlo al posto di quel determinato figlio.

La sintassi risulta:

`node.insertBefore (nodo_da_inserire, nodo_figlio_esistente)`

`node.replaceChild (nuovo_nodo, vecchio_nodo)`

- `removeChild`

Elimina uno specificato nodo figlio dalla lista del nodo a cui il metodo viene applicato:

`node.removeChild (nodo)`

2.1.5 Applicazioni

Alla luce delle proprietà del DOM e dei principali metodi che ho nei precedenti paragrafi elencato e descritto è facile capire perché è stato importante procedere con la disamina del cosiddetto Document Object Model.

Tramite il DOM e i suoi metodi infatti è possibile agire facilmente su un documento (anche non necessariamente scritto con linguaggio HTML) e sulla sua struttura, procedendo in modo chiaro e semplice ad operazioni che altrimenti richiederebbero molto più tempo e risulterebbero molto più confusionarie.

Si può ad esempio decidere di modificare tutti i tags di un certo tipo, applicare un determinato attributo o toglierne uno, aggiungere o rimuovere interi elementi nel

documento arrivando a cambiarlo radicalmente con l'utilizzo di semplici metodi in sequenza.

In conclusione, l'applicazione di questi metodi, soprattutto nel caso di documenti complessi, semplifica di molto la vita di ogni programmatore permettendogli di velocizzare sensibilmente la modifica o il processo di analisi del documento, permettendogli di agire contemporaneamente su più tags dello stesso tipo nonché di evitare una complessa rivisitazione pezzo per pezzo del documento in questione allo scopo di ottenere chissà quale informazione o applicare un qualche update, il che spesso permette di ridurre sensibilmente anche la possibilità che si presentino errori.

Terminata questa utile e interessante introduzione alle caratteristiche del DOM e a ciò che ci permette di fare, possiamo addentrarci finalmente nella disamina del linguaggio HTML 5 vero e proprio.

Nei prossimi paragrafi andrò ad analizzare le sue principali features e, infine, nel seguente capitolo, mi concentrerò su quali sono effettivamente le novità introdotte da questa nuova versione, come e perché è interessante applicarle nonché quali saranno effettivamente i vantaggi che potremo avere da un loro utilizzo.

2.2 Sintassi

La sintassi su cui ci si dovrà basare secondo le specifiche di HTML 5 risulterà, come vi sarà presto chiaro, all'insegna della facilità di implementazione e dell'immediatezza. Tuttavia, nonostante l'enorme numero di possibilità diverse concessaci sintatticamente parlando da questa nuova versione di HTML non dovrà essere usata a sproposito e si consiglia utilizzare sempre il buon senso al fine di mantenere una buona leggibilità del documento che si andrà a creare.

Innanzitutto appare chiaro fin da subito come si ponga un occhio di riguardo alla semantica associata da HTML agli elementi.

Agli autori di pagine web che utilizzeranno questo standard si ricorda che a ogni tag è associato un significato e, per questo, non vanno usati a sproposito.

Usare tag tipici di una tabella (<tr>, <td>) per impaginare un documento che chiaramente al suo interno non fa uso di tabelle è non conforme secondo le nuove specifiche, anche se la pagina è sintatticamente corretta. In questo caso è preferibile utilizzare tag tipici dei paragrafi come <p> e <blockquote> ad esempio.

Concentrandoci ora sulla sintassi vera e propria analizziamo brevemente le direttive della W3C.

2.2.1 Tag

Ci possono essere diversi tipi di tags in base al modo nel quale vengono implementati:

```
tag 'classico'
<nome_tag> text </nome_tag>

tag 'vuoto'
<nome_tag attribute1=... attribute2=... (...) attributen=...>

tag 'autochiudente'
<nome_tag attribute1=... attribute2=... (...) attributen=... />
```

In base al contenuto che possono racchiudere gli elementi associati ai relativi tag si possono invece dividere in:

- Elementi normali: possono racchiudere contenuti sotto forma di testo, commenti HTML o altri elementi HTML...etc. Fra questi possiamo quindi riscontrare i paragrafi <p>, le liste , i titoli <h1>...etc. Di solito vengono implementati secondo la forma del tag 'classico'.
- Elementi vuoti: questi elementi non possono contenere alcun contenuto. Vengono implementati attraverso un tag 'vuoto'. Fra questi possiamo trovare *area* (definisce un area all'interno di una image-map), *base* (specifica un indirizzo di default al quale vengono indirizzati tutti i links di una pagina), *br* (inserisce una linea vuota), etc. La chiusura del tag non è obbligatoria in questa versione di HTML.
- Elementi provenienti da altri namespaces: vengono utilizzati tag 'autochiudenti'. Questi elementi sono quelli che poi vengono utilizzati da specifiche esterne come MathML ad esempio.

Nelle nuove specifiche vengono concesse diverse libertà all'insegna di quella immediatezza e semplicità di cui parlavo sopra.

I tag possono essere definiti sia attraverso lettere maiuscole che minuscole.

In alcuni casi, implementando un tag 'classico' si può omettere la particella di chiusura. Ovviamente perché questo non alteri il significato del codice HTML,

l'applicazione di questa possibilità è limitata ai casi nel quale il browser può determinare autonomamente i limiti di operatività del tag di apertura.

Omettendo il tag `<html>` questo viene considerato come dichiarato implicitamente e inserito a runtime dallo user-agent.

2.2.2 *Attributi*

Anche per quanto riguarda la definizione degli attributi W3C ha apportato qualche modifica.

Non è più necessario infatti racchiudere il loro valore fra virgolette. D'ora in poi si potranno quindi implementare nei seguenti modi:

- Attributi vuoti: non è necessario definire un valore per l'attributo, basta il nome e questo verrà ricavato dalla stringa vuota. Facciamo l'esempio di un tag `input` che vogliamo sia preselezionato al caricamento della pagina web; i seguenti modi di definire questa situazione sono equivalenti:

```
<input checked="checked" />
```

```
<input checked>
```

- Attributi senza virgolette: i valori degli attributi possono essere definiti in diversi modi, i seguenti sono equivalenti:

```
<input checked=checked />
```

```
<input checked='checked' />
```

```
<input checked="checked" />
```

Nelle specifiche viene semplificata anche l'implementazione di alcuni tag che prima richiedevano obbligatoriamente la definizione di alcuni attributi per essere considerati conformi. Ora, prendendo in considerazione il tag `<style>`, si può per esempio evitare di specificare l'attributo `type`.

2.3 *Attributi Globali*

Questo tipo di attributi è applicabile a qualsiasi tag indistintamente. Non sono una novità assoluta ma sicuramente nelle specifiche di HTML 4 tali attributi non avevano subito uno sviluppo importante come in queste.

Il più famoso fra i Global Attributes fino ad ora è stato sicuramente “id”, nella nuova versione di HTML se ne sono aggiunti diversi e alcuni molto interessanti.

Concentriamoci su una selezione dei più importanti fra essi:

- *id = value*: specifica un identificatore univoco per il tag al quale è applicato. Il valore deve essere diverso da quello di tutti gli altri applicati eventualmente nel cosiddetto home subtree, ovvero il sotto albero relativo al primo antenato del nodo al quale ci stiamo riferendo, e ovviamente deve contenere almeno un carattere. È utile nel caso di selettori CSS o per poi ricavare il tag attraverso il metodo `getElementById()`.
- *title = value*: il valore deve essere una stringa di testo e rappresenta una sorta di informazione aggiuntiva riguardante il contenuto del tag al quale viene associato. Per un immagine potrebbe essere una sua descrizione o un modo per citare chi è l'autore di essa, oppure per esempio nel caso di una citazione potrebbe contenere maggiori informazioni riguardo la fonte. L'omissione di tale attributo significa che, anche per il tag in questione, diventa rilevante il 'title' eventualmente associato all'antenato più prossimo. Alcuni tag attribuiscono a questo attributo un ulteriore valore semantico.
- *lang = value*: specifica il linguaggio principale per il contenuto del tag.
- *dir = value*: specifica la direzione con la quale si deve leggere il contenuto del tag. Il valore di questo attributo può assumere i seguenti stati:
 - *ltr* (left-to-right)
 - *rtl* (right-to-left)
 - *auto* (la direzionalità viene determinata analizzando i nodi parenti)
- *class = value*: il valore di questo attributo è un insieme di parole separate da spazi ognuna delle quali rappresenta una diversa classe alla quale il relativo tag appartiene. Molto utili soprattutto per una gestione più veloce degli stylesheet, ma, portando l'analisi a livelli più complessi, utile anche per selettori CSS o

metodi del DOM. Qui di seguito possiamo vedere un esempio nel quale prima definiamo due classi dal tipo di formato diverso, quindi in seguito le richiamiamo per applicare tale stile ai tags che ci interessano senza dover ridefinire al loro interno tutti gli attributi ogni volta.

```
<html>
<head>
<style type="text/css">
h1.intro {color:blue;}
p.important {color:green;}
</style>
</head>

<body>
<h1 class="intro">Header 1</h1>
<p>A paragraph.</p>
<p class="important">Note that this is an important paragraph.</p>
</body>
</html>
```

- *style* = "*property1:value ; property2:value ; etc.*": questo attributo si riferisce a quello definito nel linguaggio CSS. Permette di dare al tag uno stile secondo la sintassi sopra descritta, elencando tutte le proprietà come colore, sfondo e via scorrendo separate da punti e virgola. Ad esempio il contenuto di un tag così definito:

```
<DIV style="color:red; font-style:italic">I'm some red-hot italic text!</DIV>
```

Apparirà come di seguito...

I'm some red-hot italic text!

- *Contenteditable* = *true*: con questo attributo il contenuto dell'elemento a cui è applicato e tutti gli elementi sotto di esso diventano editabili da web. In poche parole si va oltre la textarea per raggiungere un livello di modificabilità del documento al quale solo tramite alcune librerie si era cercato di arrivare in passato con HTML 4. L'unico limite a tali libertà è quello che dopo la modifica il testo deve comunque rimanere conforme alle specifiche di HTML 5 per questo i browser potranno interpretare attraverso diverse politiche quello che digita la persona sul di essi. Per fare un esempio, al comando di invio qualcuno deciderà di inserire il tag `<br>`, altri potrebbe decidere di aprire un nuovo paragrafo mandando a capo e inserendo un tag `<p>`. A disposizione dell'utente ovviamente ci sono già numerose API che lo possono aiutare a modificare zone di testo da lui selezionate attraverso l'attivazione di alcuni

comandi dall'esterno. Possiamo infatti creare una toolbar e permettere che premendo un bottone su di essa la selezione corrente diventi in grassetto. In questo caso dovremmo invocare la seguente funzione:

```
document.execCommand('bold')<br><br>
```

- *spellecheck=true*: in generale, ogni browser contiene un motore di verifica della sintassi grammaticale, ma nel caso noi volessimo applicare quello definito dalle specifiche HTML5 proprio su quelle porzioni che come visto precedentemente possiamo dichiarare editabili dall'utente, allora possiamo associare questo attributo al tag in questione settando il suo valore a true.
- *contextmenu="value"*: con questo attributo potremo associare ad un oggetto un menu contestuale, ovvero un menù che apparirà una volta che qualcuno cliccherà con il tasto destro su di esso. Per adesso vediamo un esempio ma tale attributo non è stato ancora implementato in maniera funzionante da nessuno per cui, ad ora, rimane relegato al concetto di semplice specifica.

```
<form name="npc">
  <label>Character name: <input name="char" type="text"
contextmenu="namemenu" required></label>
  <menu type="context" id="namemenu">
    <command label="Pick random name"
onclick="document.forms.npc.elements.char.value = getRandomName()">
    <command label="Prefill other fields based on name"
onclick="prefillFields(document.forms.npc.elements.char.value)">
  </menu>
</form>
```

In questo form abbiamo piazzato una label dove l'utente dovrebbe inserire un nome. Ad essa però abbiamo associato un menu contestuale con due possibili opzioni che appariranno premendo il tasto destro del mouse sulla label stessa. Con una di esse nel campo di testo faremmo digitare un nome casuale, con l'altra riempiamo il campo basandoci sul nome dell'utente.

- *data-*=value*: con questo suffisso HTML ci dà la possibilità di definire per ogni elemento una serie di attributi il cui nome lo definiamo noi. L'unica cosa che deve rimanere invariata in ognuno di essi è il suffisso definito sopra (*data-*), ad esempio:

```

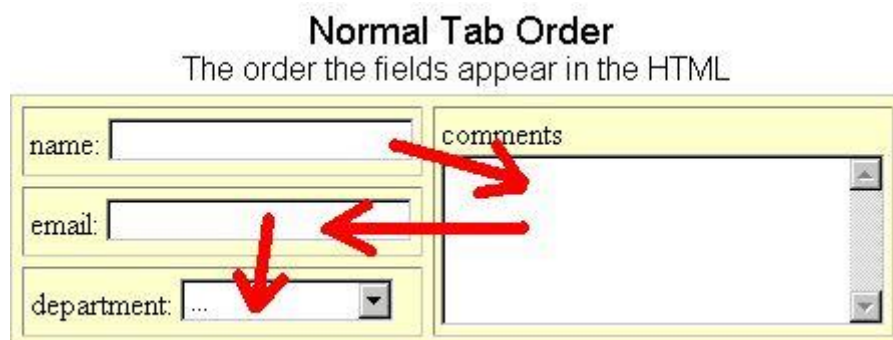
```

In questo caso mi sono portato ad esempio definendo i miei attributi principali all'interno del tag `<img>` che visualizza la mia fototessera.

- *hidden*: sostituisce ' `style="display:none"` ', quindi ogni tag al quale viene associato non verrà considerato rilevante e non verrà visualizzato sullo schermo.
- *tabindex="integer_value"*: siamo in un tipico contesto di registrazione ad un sito web. Siamo di fronte ad un form il quale è composto da molti campi di testo da compilare o da selezionare in modo appropriato. Spesso in questi casi, per velocizzare l'operazione, ci si affida al tasto tab per passare in modo "ordinato" da un campo all'altro, evitando la scomodità di abbandonare con le mani la tastiera per utilizzare il mouse. Con questo attributo possiamo dare un ordine ai campi che con il tasto tab verranno visitati. Ogni tag dovrà avere tale attributo settato ad un valore intero che ne indica la posizione nell'ideale procedimento di visualizzazioni che stiamo andando a creare. Per rendere meglio l'idea vediamo un esempio e cosa cambia applicandolo ad un form generico:

Normal Tab Order
The order the fields appear in the HTML

name:	comments
email:	
department:	



```

<FORM ACTION="../cgi-bin/mycgi.pl" METHOD=POST>
<TABLE BORDER CELLPADDING=3 CELLSPACING=5 BGCOLOR="#FFFFCC">
<TR>
  <TD>name: <INPUT NAME="realname" TABINDEX=1></TD>
  <TD ROWSPAN=3>comments<BR>
  <TEXTAREA COLS=25 ROWS=5 TABINDEX=4></TEXTAREA></TD></TR>
<TR> <TD>email: <INPUT NAME="email" TABINDEX=2></TD></TR>
<TR> <TD>department: <SELECT NAME="dep" TABINDEX=3>
  <OPTION VALUE="">...
  <OPTION VALUE="mkt">Marketing
  <OPTION VALUE="fin">Finance
  <OPTION VALUE="dev">Development
  <OPTION VALUE="prd">Production</SELECT></TD></TR>
</TABLE>
</FORM>

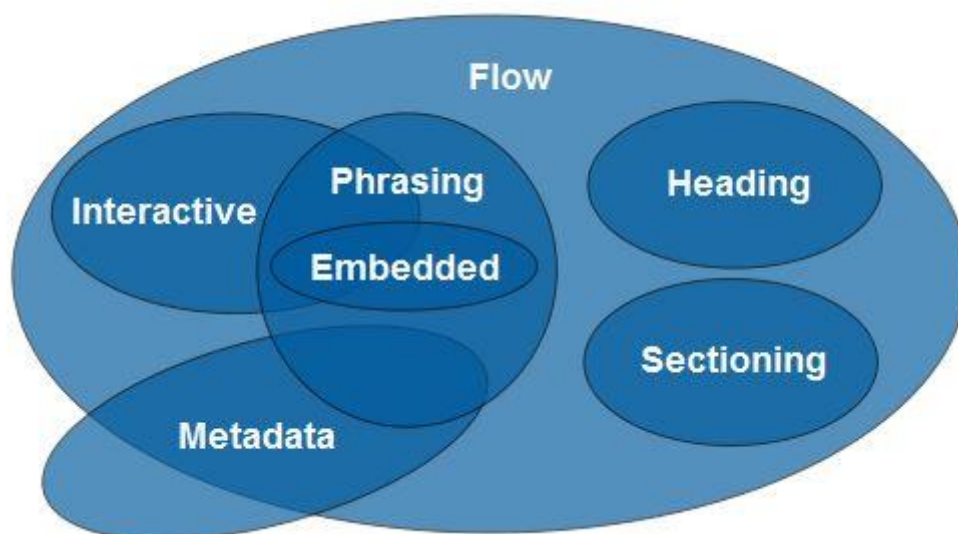
```

With TABINDEX
tab order is anything you like

2.4 Content Models

Questo capitolo è figlio soprattutto dell'attenzione che si pone sulla semantica in queste specifiche. Ogni elemento è associato infatti nelle specifiche al suo content model, ovvero ad una descrizione del contenuto che ci si aspetta vi sia al suo interno. Per essere conforme alle nuove specifiche di HTML5 bisognerà usare i giusti tags in modo che il loro contenuto corrisponda a quello richiesto dal relativo content model dell'elemento.

Dividiamo quindi i possibili contenuti che dovremmo trattare e, per ognuno di essi, gli elementi che è possibile utilizzare per farlo. Ogni elemento potrà ricadere in 0 o più categorie di contenuti il che ci permetterà anche di notare, fra le altre cose, quali analogie legano i diversi elementi, com'è chiaro dalle intersezioni nella seguente immagine che rappresenta le relazioni fra i diversi contenuti.



2.4.1 Metadata Content

Contenuti dai quali deriva la presentazione o il comportamento del documento.

Tag: base, command, link, meta, noscript, script, style, title.

2.4.2 Flow Content

Nelle specifiche si parla degli elementi categorizzati sotto tale model content come la maggior parte di quelli utilizzati nel body del documento e delle applicazioni. In effetti i tags, se li citassimo nella loro interezza, sarebbero moltissimi, per cui mi limito a considerare i principali.

Tag: a, area, address, b, blockquote, button, canvas, footer, form, h*, object, script, section... etc.

2.4.3 Sectioning Content

I tag di questo model content definiscono lo spazio di applicazione di headers e footers. Di queste due entità parleremo nei prossimi paragrafi, per rendere più chiaro il tutto diciamo che in questa categoria ci sono quegli elementi studiati per contenere al loro interno contenuti atomici e semanticamente ordinati. Di solito, al loro interno si trova un heading, rappresentante un titolo per la sezione, e un outline, considerabile come il corpo della sezione, composto eventualmente da una serie di altre sezioni innestate.

Tag: article, aside, nav, section.

2.4.4 Heading Content

Definiscono l'header, il quale può essere non solo la testata, bensì qualsiasi cosa funga da elemento introduttivo o ausilio alla navigazione per la sezione nella quale viene posizionato l'elemento.

Tag: h*, hgroup

2.4.5 Phrasing Content

Rappresenta il testo del documento nonché tutti gli elementi di markup del testo all'interno dei diversi paragrafi.

Tag: a (solo se contiene phrasing content a sua volta), abbr, area (se figlio di un elemento map), audio, b, bdi, bdo, br, button, canvas, cite, code, command, datalist, del (solo se contiene phrasing content a sua volta), dfn, em, embed, i, iframe, img, input, ins (solo se contiene phrasing content a sua volta), kbd, keygen, label, map (solo se contiene phrasing content a sua volta), mark, math, meter, noscript, object, output, progress, q, ruby, s, samp, script, select, small, span, strong, sub, sup, svg, textarea, time, var, video, wbr.

2.4.6 Embedded Content

In questa categoria sono inseriti tutti gli elementi utili a incorporare nella pagina oggetti esterni.

Tag: audio, canvas, embed, iframe, img, math, object, svg, video.

2.4.7 Interactive Content

Tale model content comprende tutti gli elementi che permettono un'interazione con l'utente.

Tag: a, audio (con l'attributo controls presente), button, details, embed, iframe, img (con l'attributo usemap presente), input (con l'attributo type diverso da hidden), keygen, label, menu (con l'attributo type="toolbar"), object (con l'attributo usemap presente), select, textarea, video (i con l'attributo controls presente).

2.5 Elementi

In questo paragrafo tratteremo gli elementi principali che di solito caratterizzano la struttura di un tipico documento HTML.

Pur essendo un elemento anch'esso, questo paragrafo non tratterà dei forms, in quanto vista la loro importanza e particolarità ho deciso di dedicargli una sezione a parte.

Sono altresì lasciati ad un altro paragrafo gli elementi di scripting in quanto tratterò questo argomento a parte, così da poterlo solo introdurre e specificare come possa essere implementato all'interno di un documento HTML senza entrare nel dettaglio a spiegare specifiche relative ad altri linguaggi come javascript.

Gli elementi di seguito descritti saranno ordinati e divisi prendendo spunto dal contenuto che in essi sarà possibile trovare, insomma dal loro content model, ma soprattutto dall'ordine con il quale vengono utilizzati per strutturare il documento.

2.5.1 Root Element: html

L'elemento che funge da radice nell'albero di ogni documento HTML è sempre "html". Anche se le specifiche di questa versione ci danno la possibilità di ometterlo, come detto in precedenza il tag verrà inserito dallo user agent a run time. Al suo interno ci si aspetta la presenza di un head e di un body (tratteremo questi elementi nei seguenti paragrafi).

Oltre agli attributi globali "standard", per questo particolare elemento viene dichiarato un nuovo attributo chiamato manifest che regola la gestione di una memoria cache locale chiamata application cache al fine di utilizzarla nel momento in cui il browser vada momentaneamente offline. Il valore di questo attributo deve essere un URL ma può anche essere un file appositamente chiamato cache manifest, al cui interno compaiono tutti gli URL delle pagine ma anche dei singoli oggetti che il progettista ritiene siano utili ad una navigazione nell'applicazione creata in assenza di rete. Il comportamento del browser nel caso tale attributo sia presente è molto semplice: appena caricata la pagina esso fa il download nella application cache degli elementi specificati nel manifest e, in assenza di rete anche momentanea, gli utilizza per mantenere attiva l'applicazione. Al ritorno della connessione la cache sarà aggiornata.

2.5.2 Document Metadata

2.5.2.1 <head>

Il primo elemento contenuto sotto il tag “html” è head, il quale rappresenta una collezione di metadata per il documento. Il tag title è spesso indispensabile al suo interno ma in alcuni casi può essere omesso, ad esempio quando il titolo viene provvisto da un protocollo di livello superiore; questo può accadere ad esempio nel momento in cui stessimo utilizzando html per gestire una casella di posta. In questo caso infatti il titolo delle varie e-mail sarebbe dato dal mittente o dall’oggetto da lui specificato all’invio.

2.5.2.2 <title>

Rappresenta il titolo del documento, viene specificato all’interno di un tag head dove ve ne può essere al massimo uno.

2.5.2.3 <base>

Specifica l’URL di base del documento, viene specificato all’interno di un tag head dove ve ne può essere al massimo uno, in caso contrario tutti quelli successivi al primo saranno ignorati.

Oltre agli attributi globali, quando si definisce questo elemento è necessario associare ad esso almeno uno fra i seguenti attributi:

- *href*: il suo valore è un URL.
- *target*: il suo valore è una stringa contenente il nome di un browsing context valido. Il browsing context è l’ambiente nel quale gli oggetti del documento sono presentati all’utente.

2.5.2.4 <link>

Come si può dedurre dal nome di questo elemento, esso viene utilizzato per richiamare link esterni alla pagina corrente. Richiede la presenza di due attributi:

- *href*: come già spiegato sopra, contiene l’URL al quale punta questo tag
- *rel*: con questo attributo specifichiamo la relazione che intercorre fra la pagina attuale e quella cui punta il link. A questo attributo possiamo associare un insieme di valori enumerati. Di seguito ne specifico alcuni:

- *alternate*: rappresentazione alternativa del documento attuale
- *author*: link all'autore del sito
- *help*: link ad una pagina con aiuti
- *index*: link all'indice
- *next*: link al documento successivo
- *stylesheet*: link allo stylesheet

2.5.2.5 <meta>

Con questo elemento possiamo specificare quei metadata che è impossibile esprimere con gli altri tag di questo paragrafo.

Il suo significato è caratterizzato principalmente da tre attributi, almeno uno dei quali va sempre specificato:

- *name*: identifica un nome relativo alla proprietà del metadata specificato. Il suo valore è un enumerato. Va sempre accompagnato con l'attributo *content* che invece specifica il valore del metadata. Ad esempio se in *name* scelgo di inserire la stringa "author", in *content* specificherò il nome di uno degli autori della pagina.
- *http-equiv*: identifica delle direttive pratiche per il browser al fine di indirizzare l'interpretazione ma non solo che esso darà alle pagine. Anche questo è un attributo enumerato che è necessario far seguire dall'attributo *content*. Ad esempio posso scegliere "refresh" come valore di *http-equiv* e in *content* dichiarare ad esempio il numero intero 3: in questo modo il browser ricaricherà la pagina ogni 3 secondi.
- *charset*: di tags <meta> con questo attributo nel documento ce ne può essere al massimo uno. Il valore di questo attributo specifica il character encoding usato dal documento.

2.5.2.6 <style>

Con questo tag possiamo inserire nel documento informazioni circa il suo stile. Ci possono essere diversi attributi specificabili, come 'type' attraverso il quale possiamo definire il linguaggio utilizzato per scrivere il documento (se omissso di default sarà "text/css"), o 'media' per definire i media al quale lo stile sarà applicato.

In generale, per fare chiarezza sul suo utilizzo, possiamo portare ad esempio il seguente estratto, con il quale si specifica lo stile da applicare ai campi ‘body’ e ‘em’:

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <title>My favorite book</title>
    <style>
      body { color: black; background: white; }
      em { font-style: normal; color: red; }
    </style>
  </head>
  <body>
    <p>My <em>favorite</em> book of all time has <em>got</em> to be
    <cite>A Cat's Life</cite>. It is a book by P. Rahmel that talks
    about the <i lang="la">Felis Catus</i> in modern human society.</p>
  </body>
</html>
```

2.5.3 Sezioni

In questo paragrafo tratteremo gli elementi principali che strutturano le diverse sezioni del documento, i quali sono quelli che hanno subito il maggior numero di cambiamenti rispetto alla precedente versione.

2.5.3.1 <body>

Attraverso l'utilizzo di questo elemento si definisce il corpo principale del documento. Per essere conforme con le specifiche correnti, ogni documento deve avere un solo “body”. Oltre agli attributi globali, questo elemento può implementare diversi attributi relativi allo script della pagina e utili per attivare o comunque utilizzare gli event handler all'interno del documento. Parleremo sia di event handler che di script nella sezione dedicata all'implementazione di javascript all'interno di documenti HTML.

Di seguito un esempio nel quale creiamo un indicatore per segnalare se l'utente è online o offline:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Online or offline?</title>
    <script>
      function update(online) {
        document.getElementById('status').textContent =
          online ? 'Online' : 'Offline';
      }
    </script>
```

```

</head>
<body ononline="update(true) "
      onoffline="update(false) "
      onload="update(navigator.onLine) ">
  <p>You are: <span id="status">(Unknown)</span></p>
</body>
</html>

```

In grassetto ho evidenziato gli aspetti di rilievo dell'esempio.

Nel tag di script è facile notare, anche se non ho ancora trattato tale elemento, che si dichiara una funzione 'update' alla quale si passa un parametro 'online'. Questa funzione ricava il testo contenuto nell'elemento avente ID 'status', ovvero riprende la stringa "You are: ", e concatena ad essa la stringa "Online" o "Offline" a seconda che il parametro in ingresso 'online' abbia valore rispettivamente true o false.

Per aggiornare il valore di tale parametro si usano gli attributi presenti in body che gestiscono i relativi eventi. L'attributo 'ononline' setta il valore di update a true in risposta all'evento "l'utente è connesso" e il contrario succede per l'attributo 'onoffline'. Durante il caricamento della pagina, ovvero l'evento associato ad 'onload', il valore di update viene ricavato dall'API 'navigator.onLine' che ha la proprietà di mantenere un valore true nel caso il browser sia online per poi aggiornarla con un valore false ogni volta che il browser sarà "non in linea".

2.5.3.2 <section>

Semplicemente l'elemento utilizzato per dividere la struttura del documento in diverse sezioni. Al suo interno ci si aspetta l'utilizzo di un heading e di un outline. Per riprendere meglio l'approccio più semantico utilizzato per dare vita a queste specifiche, il tag <section> viene utilizzato per dare vita a raggruppamenti tematici di contenuti.

2.5.3.3 <nav>

L'elemento <nav> viene utilizzato per creare sezioni nel documento contenenti link ad altre pagine o a parti interne del documento stesso. Al suo interno è possibile, utilizzando tag come o , inserire una lista di link creando così un blocco di navigazione.

```

<nav>
  <ul>
    <li>Questo è un link</li>
    <li>Questo è un link</li>
  </ul>
</nav>

```

```

    <li>Questo è un link</li>
    <li>Questo è un link</li>
    [...]
</ul>
</nav>

```

Se all'interno della sezione che si vuole creare l'utilizzo dei collegamenti è presente ma non in maniera estesa è preferibile continuare ad usare tags come <section>.

2.5.3.4 <article>

La sezione generata usando l'elemento <article> rappresenta un'entità autonoma che è possibile potenzialmente ridistribuire o riutilizzare in parte o interamente in altre pagine sul web.

Semanticamente è chiaro che rappresenta il classico articolo di giornale, post di un blog, etc. In quanto tale, oltre agli attributi globali questo elemento supporta altri attributi tipici di post e articoli in generale, come `pubdate` per specificare la data di pubblicazione, o `author` per specificare l'autore.

La presenza di così tanti tags che definiscono, seppur in modo differente, delle sezioni all'interno del documento potrebbe rendere difficile la loro giusta implementazione e potrebbe dare adito a diversi dubbi che ora cercherò di chiarire.

L'elemento chiave per creare una sezione generica all'interno di una pagina è ovviamente <section>, ma non bisogna assolutamente pensare che il suo utilizzo precluda quello di un tag come <article>. Possono esserci sia <section> con all'interno elementi <article> che viceversa.

Un tag <article> al suo interno può contenere a sua volta altri elementi dello stesso tipo, ad esempio per impaginare i commenti relativi ad un post su un blog.

Nonostante la sintassi rimanga corretta, quello che ho detto prima non va interpretato erroneamente. Gli elementi utili alla dichiarazione di nuove sezioni non sono fra loro equivalenti: se una sezione è possibile che in futuro venga ripubblicata o riutilizzata va preferito l'utilizzo di un elemento <article> e se al suo interno l'uso dei links è massiccio, o comunque assume un ruolo da protagonista, va preferito l'utilizzo di un tag <nav>.

In presenza di script va valutata l'effettiva convenienza nell'utilizzare questo tipo di tag rispetto all'elemento <div>. In seguito questa parte risulterà più comprensibile.

```

<article>
  <header>
    <h1>Titolo articolo</h1>
    <time pubdate datetime="2011-10-09T14:28-08:00"></time></p>
  </header>
  <p>Contenuto dell'articolo</p>
  <footer>

```

```
    <p>Informazioni riguardo l'autore</p>
  </footer>
</article>
```

2.5.3.5 <aside>

Nelle specifiche, <aside> viene descritto come una sezione costituita da informazioni correlate al contenuto del suo elemento padre ma legate ad esso solo marginalmente, in modo tale da poter essere anche considerate distintamente rispetto a tale contenuto. Questa spiegazione non è del tutto immediata, ma se vogliamo tradurla ponendo una maggiore attenzione agli aspetti più pratici possiamo pensare questo elemento come una sezione di approfondimento rispetto all'articolo o <section> generica che lo contiene.

```
<aside>
  <h1>Questi sono dei contenuti di approfondimento marginali rispetto al
contenuto principale</h1>
  <nav>
    <h2>Link a pagine correlate al contenuto</h2>
    <ul>
      <li>Informazione correlata al contenuto</li>
      <li>Informazione correlata al contenuto</li>
      <li>Informazione correlata al contenuto</li>
    </ul>
  </nav>
  <section>
    <h2>Pubblicità</h2>
    [immagini pubblicitarie]
  <section>
</aside>
```

2.5.3.6 <h*>

L'asterisco rappresenta un numero intero compreso fra 1 e 6. Questo elemento definisce i titoli veri e propri delle varie sezioni del documento. In base al numero del tag si definisce la gerarchia del titolo in esame rispetto agli altri presenti nella sezione. In questo modo <h2> sarà sottotitolo di <h1> ad esempio. Tuttavia, senza l'elemento del prossimo paragrafo, il browser tratterebbe tutti gli <h*> indistintamente come titoli della sezione.

2.5.3.7 <hgroup>

L'elemento <hgroup> viene utilizzato per raggruppare gli elementi <h*> nelle diverse sezioni, rispetto alle quali quindi rappresenta l'intestazione.

L'outline della sezione in esame, utilizzando tale elemento, restituirà come suo titolo unicamente l'elemento `<h*>` più alto e non più tutti i titoli e sottotitoli in modo indistinto.

```
<article datetime="2010-11-22" pubdate >
  <header>
    <strong> <hgroup>
      <h2>Le prospettive per il futuro del web</h2>
      <h1>L'HTML 5 rivoluzionerà il mondo del web</h1>
    </hgroup></strong>
  </header>
  <p>Presto il web sarà pieno di siti e applicazioni che saranno in grado di
mettere in crisi le più grandi case di desktop application...</p>
  <footer>
    <p>Pinco pallino</p>
  </footer>
</article>
```

2.5.3.8 `<header>`

L'elemento `<header>` serve a contenere un insieme di ausili introduttivi o di navigazione. La sua funzione non è da confondere con quella degli elementi `<h*>` in quanto, come detto sopra, non funge solo da titolo per la sezione.

Oltre ad un elemento `<hgroup>`, ad esempio, al suo interno potremo inserire link alla homepage del sito o alla pagina dell'autore dell'articolo visualizzato, insomma link che forniscono un valido supporto alla navigazione nel nostro sito web.

```
<header>
  <hgroup>
    <h1>We5! Il blog della guida HTML5</h1>
    <h2>Approfittiamo fin da oggi dei vantaggi delle specifiche HTML5!</h2>
  </hgroup>
  <nav>
    <h1>Navigazione:</h1>
    <ul>
      <li><a href="/home">Home</a></li>
      <li><a href="/about">Gli autori</a></li>
      <li><a href="/refactoring">Il progetto four2five</a></li>
      <li><a href="/archives">Archivio</a></li>
    </ul>
  </nav>
</header>
```

2.5.3.9 `<footer>`

L'elemento `<footer>` contiene le cosiddette informazioni d'appendice relative alla sezione in cui è contenuto. Per ogni sezione ce ne può essere anche più di uno e il suo posizionamento non è vincolato ad essere a piè di pagina.

Tipiche informazioni che possono riguardare questo tag sono quelle relative all'autore, al copyright, ma, seppur non genera una nuova sezione vera e propria, può contenere anche link ad esempio relativi a documenti correlati.

Al suo interno possiamo dichiarare intere sezioni.

2.5.3.10 <address>

<address> fornisce informazioni sui contatti relativi al documento a cui è applicato. Le istruzioni relative al suo utilizzo risultano spesso ambigue ma, facendo degli esempi, al suo interno possiamo specificare i nomi dei manutentori della pagina web, collegamenti alle loro pagine personali, etc.

Abbiamo terminato questa panoramica approfondita sulle specifiche relative alle sezioni, paragrafo volutamente più esaustivo di altri in quanto, come ho detto anche nel corso dell'introduzione al paragrafo stesso, è quello che ha subito i maggiori stravolgimenti.

In seguito andremo a trattare, invece, una parte delle specifiche nettamente meno ritoccata rispetto alle altre e, di conseguenza, cercherò di filtrare solo gli aspetti più rilevanti e innovativi.

2.5.4 Grouping Content

- <p>: rappresenta un nuovo paragrafo;
- <hr>: utilizzato per segnalare un cambiamento tematico a livello di paragrafi (ad esempio un cambio di scena in una storia)
- <blockquote>: rappresenta una citazione. Il browser visualizzerà il suo contenuto in quanto tale secondo le sue politiche di formattazione. Possiamo inserire l'URL della fonte attraverso l'attributo 'cite = "URL_fonte"';
- : rappresenta una lista di entità ordinate, il cui ordinamento è determinante per una loro giusta interpretazione. Ogni entità è contenuta nell'elemento come nodo figlio di ;

- `<ul>`: lista di entità il cui ordine non è importante, anche in questo caso ognuna di esse sarà contenuta nel nodo figlio `<li>`;
- `<dl>`: lista di 0 o più gruppi composti da un nome e un insieme di valori per tale nome. Il nome viene contenuto in `<dt>`, i valori ad esso associati in tags `<dd>`.

```

<dl>
  <dt> Authors
  <dd> John
  <dd> Luke
  <dt> Editor
  <dd> Frank
</dl>

```
- `<pre>`: utilizzato per inserire porzioni di testo pre-formatto. Tutto ciò che si trova fra questi tag viene visualizzato nella pagina esattamente con la formattazione con la quale è stato inserito nel codice HTML: stesse andate a capo, stesse spaziature, etc.
- `<figure>`: raggruppa elementi generici, utilizzato per marcare sezioni di codice, di foto, di illustrazioni. Con il tag `<figcaption>` al suo interno possiamo specificare una sorta di “didascalia” per il contenuto che raccoglie;
- `<div>`: non ha un significato particolare, divide il documento in diverse sezioni generiche. Nelle specifiche, il programmatore viene incoraggiato a vedere questo elemento come ultima fra le possibilità e a farne uso solo nel momento in cui nessun altro tag risponda alle sue necessità.

2.5.5 Elementi relativi a contenuti text-level

Gli elementi che è possibile utilizzare a questo livello di dettaglio sono molteplici. Anche in questo caso cercherò di riassumere molto velocemente i principali per evitare elenchi troppo lunghi e noiosi.

- `<a>`: è il principale elemento per definire un collegamento. A seconda degli attributi in esso specificati assume significati differenti:
 - `<a>...</a>` » Origine e destinazione di un collegamento ipertestuale.
 - `<a href="url">...</a>` » Collegamento ipertestuale.
 - `<a href="#name">...</a>` » Collegamento ad un'ancora nel documento stesso.

- `<a href="URL#name">...</a>` » Collegamento ad un'ancora in un altro documento.
 - `<a name="name">...</a>` » Ancora in un documento.
 - `<a href="mailto:e-mail">...</a>` » Collegamento ad una e-mail.
- `<em>`: usato per enfatizzare il suo contenuto (corsivo o grassetto);
 - `<strong>`: usato per dare più importanza al contenuto (corsivo o grassetto);
 - `<small>`: riduce la dimensione del testo in esso contenuto;
 - `<cite>`: rappresenta un generico titolo. Per capire meglio il suo uso farò un esempio:


```
<p>My favorite book is <cite>The Reality Dysfunction</cite> by
Peter F. Hamilton. My favorite comic is <cite>Pearls Before
Swine</cite> by Stephan Pastis. My favorite track is <cite>Jive
Samba</cite> by the Cannonball Adderley Sextet.</p>
```
 - `<q>`: rappresenta il “blockquote” a livello testuale, quindi una citazione;
 - `<time>`: definisce un al suo interno un orario o una data, o entrambe le cose. Supporta l'attributo ‘datetime’;
 - `<code>`: contiene al suo interno frammenti di computer code;
 - `<var>`: al suo interno contiene il nome di una variabile;
 - `<sub>` e `<sup>`: semplicemente il loro contenuto viene interpretato come pedice o apice rispettivamente;
 - `<i>` e `<b>`: formattano il testo al loro interno il primo dandogli uno stile italico, il secondo facendolo apparire in grassetto. Utilizzati al posto di `<strong>` e `<em>` per formattare in modo diverso il testo al loro interno senza per questo dargli una maggiore importanza;
 - `<mark>`: evidenzia il suo contenuto o, comunque, lo segnala all'utente come importante.

2.5.6 Tags di modifica

Gli elementi trattati in questo paragrafo non servono a modificare i paragrafi ma a segnalare le modifiche effettuate alle diverse sezioni. Non possono essere inclusi nel precedente paragrafo in quanto posso anche essere utilizzati a livello di tags di raggruppamento per contenere intere sezioni che potrebbero essere state aggiunte o cancellate.

`<ins>`: al suo interno poniamo il contenuto che vogliamo marcare come nuovo rispetto ad una precedente versione del documento al quale ci stiamo riferendo.

`<del>`: al suo interno poniamo il contenuto che vogliamo marcare come eliminato rispetto alle precedenti versioni del documento di riferimento.

Utilizzando contemporaneamente questi elementi possiamo ad esempio dare vita al seguente estratto, nel quale segnaliamo che il numero massimo di studenti in una stessa aula è stato modificato dal precedente valore di 100 al nuovo di 120:

```
<P>  
  Il massimo numero di studenti che possono occupare un'aula nello stesso  
  momento è <DEL>100</DEL><INS>120</INS>.  
</P>
```

2.5.7 Elementi per l'incorporamento di contenuti

2.5.7.1 `<img>`

L'elemento `<img>` viene utilizzato per incorporare nel documento un'immagine. Per la sintassi di questo elemento si utilizza un tag 'autochiudente'.

Va sempre specificato l'attributo `'src = "URL_immagine"'` che definisce la sorgente da cui ricavare l'immagine. Eventualmente si può specificare l'attributo `'alt = "fallback_content"'`. Il valore di alt è una stringa che verrà visualizzata nel caso ci siano problemi a caricare l'immagine dalla sorgente indicata.

Molto utile anche l'attributo `'usemap'` che, se presente, indica la presenza di un'immagine associata all'immagine.

2.5.7.2 `<iframe>`

"iframe" significa "inline frame" ed è una vera e propria finestra che si apre su un URL specificato nel tag stesso. La sintassi è la seguente:

```
<iframe src="URL_pagina,sorgente" width="300" height="300">
```

Contenuto alternativo per i browser che non leggono gli iframe o descrizione del contenuto del frame interno...

```
</iframe>
```

Ovviamente, tutti i browser attuali leggono questi frame ma ho comunque lasciato il testo compreso nel tag perché potrebbe essere utile per quanto riguarda il posizionamento del documento nei motori di ricerca.

È indispensabile definire l'attributo 'src', sono facoltativi quelli relativi alle dimensioni della finestra.

Porto ad esempio un estratto ricavato dal sito '*xhtml.html.it*' in quanto molto interessante. Prima citerò il codice sorgente HTML della pagina, quindi come questa viene visualizzata da un browser Mozilla Firefox.

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>HTML.it</title>
</head>
<body>
  <br><br><br><br><br>
  <div align="center">Ecco un <b>&lt;iframe&gt;</b> che si apre su
  pro.html.it:
    <br><br>
    <iframe src="http://pro.html.it" width="600" height="300">
      <p>Su <a href="http://pro.html.it">PRO.HTML.it</a> -
      Approfondimenti sul Web Publishing e articoli per
      webmaster</p>
    </iframe>
  </div>
</body>
</html>
```

Ecco un `<iframe>` che si apre su `pro.html.it`:



2.5.7.3 <embed>

<embed> rappresenta un punto di integrazione all'interno del documento HTML per un'applicazione esterna o per contenuti interattivi (per lo più multimediali, tipicamente flash, quick time...).

Come al solito è necessario l'attributo src con l'URL del contenuto da incorporare.

In risposta al fatto che questo elemento è spesso usato per trattare contenuti multimediali, si può anche inserire in esso l'attributo *'quality'* il cui valore è una stringa rappresentante la qualità con la quale si dovrà riprodurre l'elemento (high per esempio).

2.5.7.4 <object>

Nella precedente versione di HTML questo elemento aveva completamente sostituito il tag <embed> e per questo il suo utilizzo è quasi in tutto e per tutto analogo ad esso. Anche in questo caso trattiamo per lo più l'inserimento di elementi multimediali, vediamo un po' di attributi che è possibile specificare:

- data: URL dell'oggetto da inserire
- type: tipo dell'oggetto da inserire
- archive: lista di URL separati da virgole contenenti risorse relative all'oggetto inserito
- width e height: larghezza e altezza del riquadro con il quale si visualizzerà l'oggetto

Almeno un attributo fra data e type è obbligatorio che sia inserito nell'elemento.

Per questi ultimi due tag è giusto porre l'attenzione su una tipica proprietà del principale contenuto che essi trattano al loro interno, ovvero la dimensione dell'oggetto multimediale che incorporano nel documento. L'aumento diffuso della banda a disposizione dell'utente medio concede più libertà al programmatore web rispetto al passato ma va anche considerato il fatto che negli ultimi tempi vi è stata un'enorme diffusione di video in alta definizione, per cui il peso dei file è molto maggiore ora. Per questo bisogna comunque non esagerare nel momento in cui si decide di includere contenuti multimediali nel documento, perché potremo causare all'utente disagi legati al tempo di caricamento della pagina.

2.5.7.5 <param>

<param> viene utilizzato per specificare le impostazioni di esecuzione di un oggetto inserito tramite <object>. Per questo nelle specifiche viene indicato che questo elemento, di per sé, non ha alcun significato, ma lo acquista solamente se innestato all'interno dell'elemento <object>.

I suoi attributi 'name' e 'value' attribuiscono al parametro un nome ed un valore ed è obbligatorio che entrambi siano definiti.

Ogni oggetto puntato dal tag <object> accetta parametri diversi.

Nel seguente esempio richiamo un oggetto orologio, quindi dichiaro tre parametri con i quali chiedo che sia analogico, con il quadrante di colore bianco e le lancette e i dettagli di colore blue navy.

```
<OBJECT CLASSID="java:Clock.class" CODETYPE="application/java" WIDTH=100  
HEIGHT=100 TITLE="A real live clock!" STANDBY="Do you know what time it is?">  
<PARAM NAME=TYPE VALUE=ANALOG>  
<PARAM NAME=BGCOLOR VALUE=WHITE>  
<PARAM NAME=FGCOLOR VALUE=NAVY>  
</OBJECT>
```

2.5.7.6 <video>

Questo elemento viene utilizzato per inserire esplicitamente un video all'interno della pagina. Grazie a questo elemento, se supportato dal browser dell'utente, la maggior parte dei video può essere riprodotta senza l'ausilio di player esterni, principalmente di un flash player ovviamente.

La sintassi è molto simile a quella del tag . È obbligatorio specificare attributi 'src', 'width' e 'height'. Possiamo anche specificare un testo visualizzato nel caso il video non possa essere caricato o per dare maggiore visibilità nel motore di ricerca alla pagina. Altri attributi utili:

- controls: attributo senza valore, se presente visualizza i controlli del video;
- preload: se presente il video viene caricato subito all'apertura della pagina;
- autoplay: se presente il video parte automaticamente all'apertura della pagina;
- poster: ha come valore un URL che punta ad un immagine da visualizzare nel frattempo che il video non sia ancora stato caricato.

2.5.7.7 <audio>

Questo elemento incorpora esplicitamente un contenuto audio nel documento. Il suo comportamento è del tutto analogo a quello del tag <video>. Non è presente

l'attributo 'poster' ma è molto utile quello 'loop' con il quale si può specificare il numero di volte che la musica verrà riprodotta una volta che l'utente deciderà di farla partire. Specificando l'attributo 'autoplay' potremo dar vita ad una musica di sottofondo per il nostro sito web.

2.5.7.8 <track>

Questo elemento permette all'autore del sito di specificare esplicitamente una serie di frasi o tracce testuali visualizzate secondo una determinata scaletta temporale. <track> non ha nessun significato se non è associato ad un elemento richiamante un contenuto multimediale. L'attributo enumerato 'kind' viene utilizzato per definire il tipo delle tracce testuali da visualizzare:

- subtitles: sottotitoli, visualizzati sopra il video;
- captions: possono essere sottotitoli per il dialogo, ma anche per effetti sonori o in generale informazioni rilevanti rispetto alla traccia audio, visualizzati sopra il video;
- chapter: titolo dei capitoli, visualizzati come un indice interattivo e utilizzati per navigare nella risorsa multimediale;
- metadata: tracce utilizzate dagli scripts, non visualizzate.

L'attributo 'label' viene utilizzato per dare un titolo alla traccia.

Nel caso generale di elementi multimediali, in presenza di un errore, si può ricavare il valore dell'errore occorso attraverso il metodo 'media.error'.

2.5.7.9 <canvas>

L'elemento <canvas> è una delle novità di HTML 5, per questo potrebbe ancora avere qualche problema di compatibilità con alcuni browser. Questo tag viene utilizzato per operare con elementi grafici in generale o per, semplicemente, disegnare qualcosa nella pagina.

<canvas>, perché tutte le sue potenzialità siano sfruttate, richiede il più delle volte un linguaggio di scripting di supporto.

La sua sintassi è analoga a quella di <video>, <audio> e degli altri contenitori di oggetti multimediali. Se le dimensioni del riquadro non vengono specificate con gli attributi width e height gli verrà data dal browser una grandezza standard di 300x150. Come nei precedenti elementi, il testo compreso fra i tags <canvas> e </canvas> verrà visualizzato nel caso l'elemento grafico non possa essere caricato dal browser.

Oltre agli attributi di dimensionamento del riquadro, questo elemento, a differenza degli altri sopracitati, non supporta altri attributi che non siano quelli globali.

Nelle specifiche si puntualizza sul fatto che gli autori non dovranno usare canvas a sproposito solo per la sua comodità nel rendere graficamente degli oggetti. Se vogliamo, ad esempio, inserire un titolo graficamente accattivante lo dovremmo inserire in un elemento di tipo `<h*>` per poi andarlo a ritoccare attraverso l'integrazione di codice CSS.

Vediamo ora un esempio del suo utilizzo, spiegando il principale modo con il quale può essere implementato. Per operare in maggior dettaglio su di esso si richiede che il programmatore sviluppi conoscenze di linguaggio javascript.

```
<canvas id="bandierina" width="180" height="100">Non Supportato</canvas>
<script type="text/javascript">
var canvas = document.getElementById('bandierina');
var c = canvas.getContext('2d');
c.fillStyle='#FF0000';
c.fillRect(0,0,180,100);
c.fillStyle='#FFFFFF';
c.fillRect(0,0,120,100);
c.fillStyle='#00FF00';
c.fillRect(0,0,60,100);
</script>
```

Del tag script parleremo in seguito nel paragrafo dedicato all'introduzione a javascript, per ora spiegherò le operazioni di base per operare con `<canvas>`.

Definiamo l'elemento con un ID. All'interno di un tag script nel documento creiamo una variabile canvas richiamando l'elemento definito prima tramite il suo ID con il metodo `getElementById`. Quindi definiamo una variabile c prelevando e specificando il contesto sul quale vogliamo operare (in questo caso 2d).

Di seguito elencherò un po' di metodi di script utili ad operare su canvas:

- `obj.beginPath(); (...); obj.closePath();` sono metodi utilizzati per lo più nella creazione di lavori più complessi al fine di racchiudere fra di essi una pluralità di istruzioni
- `obj.fillRect (X, Y, L, A);` creiamo un rettangolo dove:
 - o X = distanza dal bordo sinistro del canvas
 - o Y = distanza dal bordo superiore del canvas
 - o L = base del rettangolo
 - o A = altezza del rettangolo

Sostituendo nella dichiarazione fill con stroke disegneremo solo il contorno, con fill si tende a riempire anche il suo interno.

- `obj.arc (X, Y, raggio, angoloIniziale, angoloFinale, sensoAntiorario)`; creiamo un cerchio dove:
 - `X` = distanza del centro dal limite sinistro del canvas
 - `Y` = distanza del centro dal limite superiore del canvas
 - `angoloIniziale` = angolo iniziale dell'arco
 - `angoloFinale` = angolo finale dell'arco
 - `sensoAntiorario` = `true` se vogliamo che il senso di percorrenza per passare da angolo iniziale a finale sia antiorario
- `obj.fillStyle = "rgba (R, G, B, T)";` modifichiamo lo stile di riempimento:
 - `R, G, B` = livelli dei colori rosso, verde e blue, ciascuno variabile fra 0 e 255
 - `T` = livello di trasparenza in percentuale (valori fra 0 e 1)

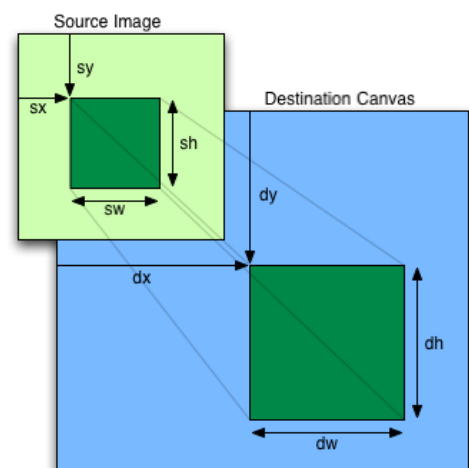
Ora trattiamo le animazioni, caratterizzate mediante l'esecuzione periodica di script ad intervalli di tempo regolari.

- `setInterval (nomeFunzione, intervalloTempo);`
- `setTimeout (nomeFunzione, intervalloTempo);`

Con questi metodi possiamo gestire gli intervalli di tempo fra le esecuzioni della funzione che specifichiamo.

Caricamento di un immagine e metodi per modificarla:

- `obj.drawImage (image, sx, sy, sw, sh, dx, dy, dw, dh)`; carichiamo nel canvas un'immagine che rappresenteremo scalandola rispetto all'originale. Evitando di definire gli attributi di scalabilità l'immagine verrà presentata identica all'originale (immagine a fianco).
- `obj.rotate (angoloRotazione)`; ruota di un angolo, definito in modo percentuale con un valore da 0 a 1.



2.5.7.10 <map>

Il tag <map> è utile nel caso si voglia che solo una parte di un'immagine sia collegata ad un determinato link. Per implementare questa feature dobbiamo prima specificare all'interno del tag relativo all'immagine di interesse che questa è associata ad una mappa. Lo facciamo attraverso l'attributo 'usemap':

```

```

A questo punto creiamo la mappa di immagini utilizzando l'elemento <map> e associandogli, attraverso l'attributo obbligatorio 'name', l'immagine su cui ci interessa agire:

```
<map name="nomeMappa">
```

```
...
```

```
</map>
```

Al suo interno utilizziamo il tag <area> che approfondiremo di seguito per definire le zone dell'immagine da far corrispondere ai links.

2.5.7.11 <area>

L'elemento <area> rappresenta principalmente un link associato ad una determinata area all'interno di una mappa di immagini.

Viene definito attraverso i seguenti attributi, fra i quali 'href' e 'alt' è obbligatorio siano presenti:

- *href* = "URL": indica il link di destinazione dell'area cliccabile;
- *alt* = "nome_link": imposta un testo alternativo all'area di immagine, utile per descrivere il link e per indicizzarlo nei motori di ricerca;
- *shape* = "enum_forma": definisce il tipo di forma che sarà visualizzata. È un attributo enumerato e può assumere i valori:
 - *rect*: crea un rettangolo o un quadrato a seconda dei lati definiti;
 - *circle*: cerchio o ovale;
 - *polygon*: forma astratta;
 - *default*: non genera alcuna area e può essere omissso;
- *coords* = "x, y, z, w": imposta le coordinate del collegamento al link sull'immagine. Definire questo attributo ad occhio è molto complicato ma esistono editor apposta per definire tali coordinate in modo più semplice e preciso. Inoltre le coordinate variano giustamente con il variare dell'attributo 'shape' che, a seconda della forma che definisce, richiede più o meno coordinate per specificarne tutte le caratteristiche (per il rettangolo base e altezza, per il cerchio raggio etc.).

2.5.8 Elementi per la gestione di tabelle

Per trattare questo argomento partiremo da un esempio di implementazione degli elementi principali per poi scoprirli meglio nel dettaglio:

```
<table>
  <caption>
    <strong>Characteristics with positive and negative sides.</strong>
    <details>
      <summary>Help</summary>
      <p>Characteristics are given in the second column, with the
        negative side in the left column and the positive side in the right
        column.</p>
    </details>
  </caption>
  <thead>
    <tr>
      <th id="n"> Negative
      <th> Characteristic
      <th> Positive
    </tr>
  </thead>
  <tbody>
    <tr>
      <td headers="n r1"> Sad
      <th id="r1"> Mood
      <td> Happy
    </tr>
    <tr>
      <td headers="n r2"> Failing
      <th id="r2"> Grade
      <td> Passing
    </tr>
  </tbody>
</table>
```

- `<table>`: tag usato per contenere i dati da visualizzare sotto forma di tabella;
- `<caption>`: definisce il titolo della tabella;
- `<colgroup>`: rappresenta un insieme di colonne `<col>`;
- `<col>`: rappresenta una o più colonne, per avere senso deve essere nodo figlio di un `<colgroup>`;
- `<tbody>`: rappresenta un insieme di righe della tabella, più precisamente l'insieme di righe che compongono il corpo della tabella;
- `<thead>`: definisce un blocco di righe rappresentanti il titolo delle colonne della tabella;
- `<tfoot>`: rappresenta un blocco di righe che fungono da `<footer>` delle colonne della tabella. Nelle specifiche viene incentivato il suo utilizzo dopo l'elemento `<thead>` e prima di quello `<tbody>`;
- `<tr>`: nodo figlio di `<thead>`, `<tfoot>` e `<tbody>`, rappresenta una riga composta da più celle;
- `<td>`: nodo figlio di `<tr>`, rappresenta una singola cella;

- `<th>`: nodo figlio di `<tr>`, rappresenta la cella header della tabella;

Per quanto riguarda gli attributi, oltre a quelli globali, i tag `<td>` e `<th>` ne hanno in comune diversi molto utili per la formattazione della nostra tabella:

- `colspan = "n_colonne"`: attributo per specificare quante colonne la cella comprende;
- `rowspan = "n_righe"`: come `'colspan'` ma riferito alle righe;
- `headers = "id_cellaTitolo"`: indica la cella che funge da titolo per quella che stiamo creando. Il valore deve essere uguale a quello dell'attributo `'id'` associato alla cella header;
- `scope = "enum"`: attributo enumerato per definire il significato della cella:
 - o `col`: titolo di una colonna;
 - o `row`: titolo di una riga;
 - o `colgroup`: titolo di un insieme di colonne;
 - o `rowgroup`: titolo di un insieme di righe;

2.5.9 Elementi interattivi

In questo paragrafo vengono trattati elementi che cambiano le loro caratteristiche a seconda di quello che succede nel documento. Ovviamente questa interattività va creata mediante comandi di scripting, che tratteremo a grandi linee nell'ultimo paragrafo di questo capitolo.

2.5.9.1 `<detail>`

Questo elemento specifica dei dettagli aggiuntivi o dei controlli che è possibile rendere visibili su richiesta.

Perché il contenuto del tag sia visibile bisogna che in esso sia presente l'attributo `'open="open"'`.

Usando al suo interno l'elemento `<summary>` possiamo definirne l'header.

2.5.9.2 `<summary>`

Rappresenta l'header dell'elemento `<detail>`.

2.5.9.3 `<command>`

L'elemento `<command>` definisce un comando button, radio button o checkbox e viene visualizzato solo se tag figlio del tag `<menu>` che vedremo nel prossimo sotto-paragrafo. Se non si trova all'interno di `<menu>` non viene visualizzato ma può comunque essere utilizzato per specificare un keyboard shortcut, ovvero per associare ad un tasto della tastiera un collegamento o un operazione.

Fra gli attributi possiamo avere *'checked'*, *'disabled'*, *'icon'* (contenente l'URL di un'icona per il bottone), *'label'* (nome del comando), *'type'*, *'radiogroup'* (se il tasto è un radio button allora indica il nome del gruppo di bottoni radio al quale appartiene).

Nel seguente esempio, quando il tasto viene cliccato viene visualizzato il messaggio Hello World. Il tutto viene ottenuto sfruttando uno dei generatori di eventi di HTML5 che vedremo nella sezione di scripting, ovvero *onclick*, che genera un evento nel momento in cui viene premuto l'elemento al quale è associato.

```
<menu>
<command onclick="alert('Hello World')">
Click Me!</command>
</menu>
```

2.5.9.4 `<menu>`

Viene utilizzato per definire una lista di comandi o campi di input, più in generale per creare menù contestuali, toolbars, raggruppamenti di controlli per forms o comandi.

```
<menu>
<li><input type="checkbox" />Red</li>
<li><input type="checkbox" />blue</li>
</menu>
```

2.6 I Forms

Ho deciso di dividere questo argomento dal precedente perchè, viste le numerose aggiunte e rivisitazioni relative a questo elemento, mi sembrava riduttivo dedicargli un sotto-paragrafo.

Desidero fin da subito però precisare che i 'forms' rimangono pur sempre degli elementi nel documento HTML.

Nei prossimi paragrafi mi concentrerò su tutti gli aspetti per implementarli, ricordando che mi addentrerò sull'analisi delle vere e proprie novità solo nel prossimo capitolo.

2.6.1 Introduzione

Un form è descrivibile come un modulo e principalmente viene utilizzato per gestire le interazioni con l'utente della nostra pagina web.

La creazione di un form è frutto di un percorso composto da diversi passi, il più delle volte strutturati nel seguente modo:

- Creazione dell'interfaccia utente del form: definiamo la struttura e la formattazione dell'interfaccia utente del nostro modulo utilizzando gli elementi e gli attributi messi a nostra disposizione da HTML5.
- Configurazione delle politiche di comunicazione del form con il server: fondamentalmente possiamo scegliere come il form comunica con il server, specificando quale fra i due metodi principali HTTP GET e HTTP POST preferiamo sia utilizzato. Dobbiamo inoltre specificare il tipo di encodifica per i dati e l'URL del service che li gestirà.
- Configurazione delle politiche di validazione dei dati immessi dall'utente: questo step si occupa dell'implementazione lato client di quella serie di controlli atti a validare i dati ricavati dall'interazione con l'utente. Nelle precedenti versioni questo significava ricorrere in modo esteso all'utilizzo del linguaggio javascript ma ora, grazie alle aggiunte delle nuove specifiche, ridurremo questa parte del nostro compito se utilizzeremo in modo appropriato tutti i nuovi elementi e i nuovi attributi a nostra disposizione. Capiremo meglio quanto appena detto leggendo il prossimo capitolo sulle novità.

2.6.2 Categorie di elementi del form

Oltre alle categorie definite grazie ai content models, gli elementi utilizzabili nei form, a seconda delle loro proprietà, possono dividersi nelle seguenti classi:

- *Listed elements*: tutti quegli elementi che, se presenti, possono essere ritornati dalle APIs 'form.elements' e 'fieldset.elements' e, per questo, vengono considerati come parte delle loro liste (la prima, applicata al form, si occupa di tutti i suoi elementi; la seconda solo di quelli a cui l'utente ha associato un valore);

Tag: button, fieldset, input, keygen, object, output, select, textarea.

- *Elementi etichettabili*: tutti gli elementi a cui è associabile un'etichetta tramite il tag `<label>`;
Tag: button, input (se il suo attributo 'type' non ha valore "hidden"), keygen, meter, output, progress, select, textarea.
- *Submittable elements*: denota gli elementi usati per dar vita al set di dati del form;
Tag: button, input, keygen, object, select, textarea.
- *Elementi resettabili*: elementi che potrebbero subire modifiche a seguito di un reset dell'elemento `<form>`;
Tag: input, keygen, output, select, textarea.

2.6.3 Elementi strutturali del form

2.6.3.1 `<form>`

L'elemento padre di ogni form, da esso tutto ha inizio.

Rappresenta una collezione di elementi associati alla creazione del form, alcuni dei quali modificabili dall'utente o, comunque, utili all'interazione con esso.

Oltre agli attributi globali è possibile associare ad esso molti altri attributi. Di seguito ne citerò alcuni specificando, nel caso, quali di essi sono obbligatori:

- *action* = "URL": obbligatorio, specifica l'URL dove inviare i dati del form quando questi vengono confermati dall'utente, ad esempio attraverso la pressione di un `<button>`;
- *method* = "enum": specifica il metodo con il quale inviare i dati al server. Può assumere i valori:
 - *GET*: accoda i dati all'indirizzo URL della pagina richiesta, facendo seguire il nome della pagina da un punto interrogativo e dalle coppie nome=valore separate, se sono più di una, da un '&'. La stringa dopo il punto interrogativo è detta "query string";
 - *POST*: i dati vengono inviati in modo che non siano direttamente visibili all'utente attraverso la richiesta HTTP che il browser invia al server.

In entrambi i casi i dati inviati vengono memorizzati in un array superglobale, rispettivamente denominato `$_GET` e `$_POST`.

- `name = "nome_form"`: specifica il nome del form.
- `autocomplete = "enum"`: specifica che l'elemento al quale è associato può essere completato in modo automatico dal browser (*"on"*) oppure no (*"off"*). Nuovo attributo utile quando si tratta con campi delicati come quelli dedicati all'inserimento del numero di una carta di credito per esempio.
- `novalidate = "boolean"`: assume valori true o false a seconda che vogliamo o non vogliamo che vengano saltati tutti i meccanismi di validazione dei tag che discendono da `<form>`. Se per esempio un elemento figlio di `<form>` avesse dichiarato l'attributo `required`, specificando questo attributo come true non verrà controllato che tale elemento sia stato digitato o meno dall'utente.

2.6.3.2 `<fieldset>`

Elemento usato per raggruppare logicamente una serie di elementi all'interno di un form. Questo tag disegna un box attorno agli elementi che contiene al suo interno e il suo titolo può essere specificato mediante il tag `<legend>`.

Specificando l'attributo *'disabled'* il `fieldset` viene disabilitato e diventa inutilizzabile da parte dell'utente. Utile per evitare che venga compilato prima che non sia verificata un'altra condizione. Ovviamente per questo utilizzo è necessaria l'integrazione di codice javascript.

2.6.3.3 `<legend>`

Titolo del `<fieldset>`. Vediamo un esempio di implementazione per rendere più chiari gli ultimi due paragrafi. Il seguente codice...

```
<form>
  <fieldset>
    <legend>Personalia:</legend>
    Name: <input type="text" size="30" /><br />
    Email: <input type="text" size="30" /><br />
    Date of birth: <input type="text" size="10" />
  </fieldset>
</form>
```

... genera...

Personalia:

Name:	
Email:	
Date of birth:	

2.6.3.4 `<label>`

Definisce un titolo o letteralmente un etichetta da associare ad un determinato elemento attraverso l'utilizzo di due attributi principalmente:

- *for* = "*ID_elemento*": specifica a quale elemento del form deve essere applicata;
- *form* = "*ID_form*": specifica a quale form va applicata.

2.6.4 Elemento `<input>`

L'elemento `<input>` rappresenta un generico campo nel quale l'utente può inserire dei dati. A causa del suo significato così ampio, per specializzare il suo comportamento sono stati definiti un elevato numero di attributi e sono anche stati aggiunti molti possibili valori per l'attributo `'type'`, quello che principalmente definisce il tipo di uso che l'utente potrà fare del tag.

Partiamo con l'analizzare l'enorme mole di valori assumibili da quest'ultimo attributo.

2.6.4.1 Attributo `'type = "enum_value"'`

Di seguito ecco elencati i possibili valori:

- *button*: definisce un pulsante cliccabile;
- *checkbox*: definisce un checkbox;
- *color*: definisce un color picker, ovvero una paletta di colori dalla quale è possibile sceglierne uno;
- *date*: definisce un campo dove inserire una data, con controllo del calendario;
- *datetime*: definisce una data e un ora, controllando la validità di entrambi;

- *month, week, time*: definisce un campo per rispettivamente mese, settimana e orario con controllo dei dati inseriti;
- *email*: campo utilizzato per l'inserimento di un indirizzo e-mail;
- *file*: definisce un campo e un pulsante 'Browse' per permettere l'upload di file;
- *hidden*: definisce un campo nascosto;
- *image*: definisce un immagine da usare come pulsante di conferma;
- *number*: definisce un campo per inserire un numero con controllo sulla sua precisione;
- *range*: definisce un campo per inserire un numero controllato affinché sia contenuto in un determinato range;
- *password*: definisce un campo per l'inserimento di password. I caratteri in esso inseriti saranno mascherati;
- *radio*: definisce un pulsante radio;
- *reset*: definisce un pulsante per resettare tutti i valori degli elementi del form riportandoli al loro valore iniziale;
- *search*: campo di testo per effettuare ricerche;
- *submit*: un pulsante che se premuto invia tutti i dati del form al server;
- *tel*: campo di testo per numeri telefonici;
- *text*: campo di testo per default di lunghezza pari a 20 caratteri;
- *url*: campo di testo per l'inserimento di un URL.

La maggior parte di questi valori è stata aggiunta nelle specifiche di HTML 5 e possono risultare utili anche per quanto riguarda l'implementazione dei controlli di validazione del form.

2.6.4.2 Altri attributi

Di seguito elenco gli altri attributi principali del tag `<input>`. Molti di essi sono relativi a solo alcuni tipi di elementi `<input>`; in questo caso sarà segnalato il valore dell'attributo 'type' al quale si può associare l'attributo stesso:

- *alt* = "*testo_alternativo*": nel caso `type=image`, rappresenta il testo da visualizzare nel caso ci siano problemi con il caricamento dell'immagine;
- *autocomplete* = "*on*" o "*off*": dichiara se il browser può o non può autocompletare il campo di input;

- *autofocus* = “*boolean_value*”: se “true”, questo attributo imposta il focus sullo specifico elemento del form sul quale è applicato. Appena la pagina viene caricata, il form si focalizzerà sull’elemento. Ovviamente solo un elemento del form dovrà avere questo attributo settato a true e questo elemento non dovrà avere type=hidden. Tipico esempio il campo di ricerca nella homepage di google;
- *checked* = “*boolean*”: per type=radio o checkbox. Preseleziona il campo al caricamento della pagina;
- *disabled* = “*boolean*”: disabilita l’utilizzo dell’input. Da non utilizzare se il campo ha type=hidden;
- *form* = “*formname*”: specifica il nome del o dei forms al quale l’input appartiene;
- *formaction*, *formmethod*: definiti rispettivamente come gli attributi ‘action’ e ‘method’ dell’elemento form, vengono utilizzati per sovrascrivere questi ultimi. Non si possono utilizzare in type=hidden o submit;
- *height*, *width*: per type=image definiscono le dimensioni dell’immagine;
- *list* = “*ID_elemento_datalist*”: specifica che nell’elemento <datalist> individuato dall’ID inserito come suo valore sono presenti opzioni predefinite riguardo al campo di input;
- *max* o *min* = “*numero o data*”: utilizzati per definire un range di valori validi per il dato inserito;
- *maxlength* = “*intero*”: per un campo di testo specifica il massimo numero di caratteri che è possibile immettere in esso;
- *multiple* = “*boolean*”: l’utente può inserire più valori;
- *placeholder* = “*testo*”: specifica una stringa che sarà visualizzata nel campo finché questo non sarà modificato dall’utente. Utile per descrivere che tipo di dato ci si aspetta inserisca;
- *readonly* = “*boolean*”: il valore nel campo non può essere modificato;
- *required* = “*boolean*”: il campo deve essere obbligatoriamente compilato;
- *src* = “*URL_immagine*”: solo per type=image rappresenta la sorgente da dove caricare l’immagine;
- *step* = “*numero*”: specifica che intervallo di valori deve esserci fra un numero valido e l’altro all’interno di un range. Se ad esempio il suo valore fosse 5, i numeri validi che sarebbe possibile inserire nel campo sarebbero {..., -5, 0, 5, ...}

- *value* = “*valore*”: non può essere utilizzato per input di tipo “file”; cambia significato a seconda del tipo di input:
 - *buttons*: specifica il testo sul pulsante;
 - *pulsanti immagine*: il risultato simbolico restituito agli script;
 - *checkboxes e radio buttons*: per questo tipo di input è obbligatorio l'attributo *value*. Specifica il risultato dell'elemento quando viene cliccato e il valore inviato all'URL nell'attributo ‘*action*’ del form quando viene cliccato il submit button;
 - *hidden, password e text fields*: valore di default dell'elemento.

Anche questi attributi hanno subito notevoli variazioni rispetto al passato che per lo più hanno aumentato la possibilità di validare i dati del modulo senza ricorrere in maniera massiccia agli script.

2.6.5 Principali elementi di interazione con l'utente

2.6.5.1 <button>

Questo elemento definisce un pulsante come quello che possiamo creare tramite <input>, con la differenza che al suo interno possiamo inserire contenuti come testo o immagini.

Conviene specificare sempre l'attributo enumerato ‘*type*’, scegliendo fra i seguenti valori:

- *button*;
- *reset*;
- *submit*.

In caso contrario il tipo del bottone sarà scelto dal browser.

Possiamo associare ad esso molti attributi già visti in <input> come ‘*autofocus*’, ‘*disabled*’, ‘*form*’, ‘*formaction*’, ‘*formmethod*’, ‘*value*’.

Si può specificare il nome del bottone con l'attributo ‘*name*’.

2.6.5.2 <select>, <optgroup> e <option>

Il tag <select> crea un menù a discesa. Le varie voci di questo menù vengono dichiarate dal tag <option> come mostrato di seguito:

```
<select>
  <option value ="volvo">Volvo</option>
  <option value ="saab">Saab</option>
  <option value ="opel">Opel</option>
  <option value ="audi">Audi</option>
</select>
```

I tag `<option>` possono essere raggruppati logicamente sotto l'elemento padre `<optgroup>`. Associando ad esso un attributo `label = "titolo"` assoceremo al gruppo così creato un etichetta.

Attributi utili per il tag `<select>` sono (oltre ad autofocus e form):

- *name*: per definire il nome del menù;
- *size*: per definire il numero di opzioni visualizzate nel menù a discesa;
- *multiple*: se presente dichiara che possiamo selezionare più opzioni contemporaneamente.

Per quanto riguarda il tag `<option>` abbiamo:

- *value*: per dichiarare il nome dell'opzione;
- *selected*: se presente l'opzione sarà preselezionata.

Per entrambi possiamo usare disabled.

2.6.5.3 `<datalist>`

Questo elemento definisce un insieme di opzioni. Se usato insieme all'elemento `<input>` questa lista di opzioni rappresenterà i valori che esso potrà assumere.

Con il seguente codice il dato input potrà assumere solo i valori BMW, Ford e Volvo:

```
<input list="cars" />
<datalist id="cars">
  <option value="BMW">
  <option value="Ford">
  <option value="Volvo">
</datalist>
```

2.6.5.4 `<textarea>`

Definisce un'area di testo avente più di una linea e nessun limite per quanto riguarda il numero di caratteri che è possibile inserire al suo interno.

Con diversi attributi possiamo specificare le dimensioni dell'area (`'cols'` e `'rows'`), il numero massimo di caratteri inseribili (`'maxlength'`), l'autofocus, il suo nome (`'name'`).

2.6.5.5 `<output>`

Il contenuto di questo elemento è il risultato di un calcolo, spesso eseguito tramite degli script.

Nel seguente esempio facciamo inserire due numeri in due campi di input; una volta che sono stati inseriti entrambi ricaviamo il loro valore tramite il metodo `valueAsNumber` e li sommiamo. Sarà visualizzato il risultato di questa somma:

```
<input name="numa" type="number"> +  
<input name="numb" type="number"> =  
<output name="result"  
onforminput="value=numa.valueAsNumber + numb.valueAsNumber"></output>
```

2.6.6 I nuovi elementi del form

2.6.6.1 `<keygen>`

Rappresenta un generatore di chiavi numeriche. Quando il form contenente il tag `<keygen>` viene inviato al server, la chiave privata rimane memorizzata nel keystore locale e la chiave pubblica viene inviata al server.

Utile per implementare metodi sicuri di navigazione.

2.6.6.2 `<progress>`

Rappresenta lo stato di completamento di un compito e può essere di due tipi:

- indeterminato: se non possiamo risalire alla percentuale di completamento;
- determinato: è possibile quantificare un range da zero a un valore massimo entro il quale definire a che punto siamo e quanto ci rimane per completare l'operazione in esecuzione.

Attualmente questo elemento non è supportato da nessun browser e quindi il suo utilizzo è fortemente sconsigliato.

Possiamo inserire con l'attributo `max` la quantità di lavoro complessivo richiesto dall'operazione e con `value` la quantità già completata. Ovviamente questi valori vanno aggiornati mediante script ad intervalli di tempo costanti per tutta la durata dell'operazione.

2.6.6.3 `<meter>`

Equivalente all'elemento `<progress>` solo che associato ad oggetti il cui completamento non è progressivo.

Per dare un significato a questo elemento utilizziamo i seguenti attributi:

- *min, max*: indicano valore limite minimo e massimo disponibili;
- *value*: indica il valore dell'elemento;
- *low, high*: indicano un valore considerevole basso e alto rispettivamente;
- *optimum*: indica un valore considerato ottimale. Se superiore all'attributo *high* significa che il massimo valore è il migliore, se inferiore all'attributo *low* il contrario.

2.7 Scripting

In questo paragrafo non mi addentrerò troppo nel codice javascript ma mi concentrerò sul come vengono implementati gli script in un documento HTML, sulla descrizione di cosa sono e delle loro principali caratteristiche.

Fin da ora posso affermare che la nozione di scripting rappresenta sicuramente la base per rendere interattive le nostre pagine, come d'altro canto i fogli di stile CSS sono la base per dare quel qualcosa in più al nostro documento per quanto riguarda l'aspetto grafico.

HTML, con queste nuove specifiche, ha fatto passi in avanti enormi riguardo allo scripting, permettendo al programmatore di evitare in molte occasioni dove prima era impensabile l'uso di linguaggi esterni come Javascript. Introduciamo nel dettaglio tutto questo.

2.7.1 Introduzione

Le specifiche fin da subito chiariscono che è preferibile utilizzare nel documento espedienti dichiarativi alternativi allo scripting dove questo è possibile, in quanto così si facilita la manutenzione dei meccanismi della pagina e si evitano problemi nel caso in cui l'utente l'abbia disattivato. In quest'ultimo caso, W3C chiede agli autori di fare in modo che la pagina mantenga un aspetto apprezzabile nel caso lo scripting risulti disattivato o non supportato.

Le istruzioni di scripting vanno inserite nel documento mediante l'elemento `<script>`. Queste istruzioni vengono elaborate in diversi momenti, a seconda delle scelte fatte dal programmatore.

Programmando con gli script l'autore deve concepire il documento come un albero di oggetti gerarchizzati in relazione l'uno con l'altro, ognuno dei quali ha delle proprietà che è possibile richiamare tramite gli script. Di seguito vediamo come HTML supporta tutto questo.

2.7.2 `<script>`

All'interno di questo elemento viene definito uno script. Può contenere sia le dichiarazioni di scripting vere e proprie o puntare ad un file sorgente che le contiene. Questo elemento, come anche accennato nell'introduzione, viene soprattutto utilizzato per programmare cambiamenti dinamici del contenuto, per implementare politiche di validazione per i forms o anche per manipolare delle immagini.

Di seguito ecco una descrizione degli attributi associabili a questo elemento:

- `async = "async"`: se presente, lo script sarà eseguito in maniera asincrona;
- `type = "enum_value"`: specifica il MIME type dello script. Se non specificato il suo valore di default sarà `"text/javascript"`, il più utilizzato fino ad ora ma anche il più obsoleto. Quello che adesso è il nuovo standard è il valore `"application/javascript"` ma viene mantenuto quell'altro in quanto internet Explorer non lo supporta;
- `charset = "charset"`: definisce il tipo di encodifica utilizzato per lo script;
- `defer = "defer"`: se presente, indica che lo script non sta generando nessun contenuto per la pagina e che il browser, quindi, può continuare il caricamento senza aspettare il suo completamento;
- `src = "URL"`: indica l'URL del file che contiene lo script da eseguire.

A seconda delle combinazioni ottenibili dichiarando gli attributi *'async'* e *'defer'* il comportamento dell'elemento cambia sensibilmente:

- *solo 'async' presente*: lo script verrà eseguito in modo asincrono rispetto alla pagina, il che significa che verrà eseguito mentre il resto della pagina continuerà ad essere caricata dal browser;
- *solo 'defer' presente*: lo script sarà eseguito soltanto quando l'intera pagina sarà stata processata dal browser;
- *entrambi assenti*: lo script verrà eseguito immediatamente, appena il browser arriverà a processare la sua dichiarazione. Il browser aspetterà che termini la sua esecuzione prima di continuare il parsing della pagina.

2.7.3 Attivazione degli script

Il codice eseguibile inserito dal programmatore nel documento non viene eseguito solo quando il parser del browser si trova a processare l'elemento `<script>` e il relativo contenuto. Esistono diversi modi con i quali l'autore può decidere quando o in base a cosa eseguire tali script.

Di seguito elenco i principali momenti nei quali l'esecuzione di uno script può iniziare:

- nel momento in cui il parser processa l'elemento `<script>`.
- nel momento in cui accade un particolare evento. Per ricavare il momento nel quale accade possiamo registrare un `EventListener` nel DOM usando il metodo `addEventListener()`, utilizzare gli event handler IDL attributes o content attributes etc. Molti attributi globali per la gestione degli eventi sono stati aggiunti nelle specifiche di HTML5, utilissimi per semplificarci le cose.
- nel momento in cui si processano tecnologie XBL (XML binding language, utilizzato per associare gli elementi di un documento con script, event handlers, CSS etc.) o SVG (linguaggio derivato dall'XML per la gestione di immagini, siano esse animate o statiche) che possiedono le proprie features per lo scripting.

2.7.4 Gli oggetti

Vediamo di seguito quanti oggetti possiamo trovare in un comune documento HTML e come possiamo navigare in essi:

- *navigator*: l'oggetto che rappresenta il browser stesso;
- *window*: rappresenta il padre per tutti gli oggetti solitamente, è la finestra che contiene tutto;
- *window.frames*: eventuali frames, notare come sono già gerarchicamente sotto l'oggetto window;
- *document*: è il documento vero e proprio;
- *document.forms["nomeForm"]*: l'eventuale form;
- *document.images["nomeImmagine"]*: le eventuali immagini;
- *etc.*

Vediamo come possiamo richiamare i loro metodi con un esempio. Ricavo la lunghezza dell'attributo value di un input presente in un form:

```
<html>
<head>
  <title>Esempio</title>
</head>
<body>
  <FORM NAME="mioForm">
    <input type="text" name="testoProva" value="paradiclorobenzoro">
  </FORM>
</body>
</html>
```

```
document.mioForm.testoProva.value.length
```

2.7.5 Event handlers

Questa sezione era già presente in HTML4 ma in questa versione è stata migliorata enormemente, facilitando di molto la gestione degli eventi e la creazione di pagine interattive in generale.

Prima di vedere quali sono nel dettaglio, vediamo con un esempio come implementare gli attributi relativi agli eventi:

```
<html>
<head>
<script type="text/javascript">
function displayDate()
{
document.getElementById("demo").innerHTML=Date();
}
</script>
</head>
<body>
<h1>My First Web Page</h1>
<p id="demo"></p>
```

```
<button type="button" onclick="displayDate()">Display Date</button>
</body>
</html>
```

Dall'esempio si può evincere il metodo più diffuso per eseguire script a seguito di determinati eventi. L'attributo dell'evento viene dichiarato all'interno dell'elemento che lo genera e ha come valore il nome della funzione di scripting da richiamare quando l'evento stesso ha luogo. In questo caso l'evento è il click del mouse sul pulsante button e si richiede di eseguire una funzione che seleziona l'elemento avente id "demo", ovvero il paragrafo <p>, e al suo interno inserisce la data attuale.

L'attributo onclick è solo uno dei tanti attributi disponibili in HTML5, vediamoli tutti. Utilizzo delle tabelle prelevate dal sito <http://www.w3schools.com>, a mio avviso immediate e complete.

Dividiamo gli attributi a seconda dell'elemento che genera l'evento.

2.7.5.1 Window Event Attributes

Generati dall'oggetto window. Tali attributi vengono implementati nel tag <body>:

Attribute	Value	Description	
onafterprint	New	script	Script to be run after the document is printed
onbeforeprint	New	script	Script to be run before the document is printed
onbeforeunload	New	script	Script to be run before the document loads
onblur		script	Script to be run when the window loses focus
onerror	New	script	Script to be run when an error occur
onfocus		script	Script to be run when the window gets focus
onhaschange	New	script	Script to be run when the document has change
onload		script	Script to be run when the document loads
onmessage	New	script	Script to be run when the message is triggered
onoffline	New	script	Script to be run when the document goes offline
ononline	New	script	Script to be run when the document comes online
onpagehide	New	script	Script to be run when the window is hidden
onpageshow	New	script	Script to be run when the window becomes visible
onpopstate	New	script	Script to be run when the window's history changes
onredo	New	script	Script to be run when the document performs a redo
onresize	New	script	Script to be run when the window is resized
onstorage	New	script	Script to be run when a document loads
onundo	New	script	Script to be run when a document performs an undo
onunload	New	script	Script to be run when the user leaves the document

2.7.5.2 Form Events

Eventi generati dalle interazioni dell'utente con gli elementi di un form. Anche se è possibile implementarli in tutti gli elementi è ovvio che sia più comune all'interno del tag <form>:

Attribute	Value	Description
onblur	script	Script to be run when an element loses focus
onchange	script	Script to be run when an element changes
oncontextmenu	New script	Script to be run when a context menu is triggered
onfocus	script	Script to be run when an element gets focus
onformchange	New script	Script to be run when a form changes
onforminput	New script	Script to be run when a form gets user input
oninput	New script	Script to be run when an element gets user input
oninvalid	New script	Script to be run when an element is invalid
onreset	script	Script to be run when a form is reset Not supported in HTML5
onselect	script	Script to be run when an element is selected
onsubmit	script	Script to be run when a form is submitted

2.7.5.3 Keyboard Events

Eventi generati dalla tastiera dell'utente. Gli attributi sono applicabili a tutti gli elementi:

Attribute	Value	Description
onkeydown	script	Script to be run when a key is pressed
onkeypress	script	Script to be run when a key is pressed and released
onkeyup	script	Script to be run when a key is released

2.7.5.4 Mouse Events

Eventi generati dal mouse dell'utente. Gli attributi sono applicabili a tutti gli elementi:

Attribute	Value	Description
onclick	script	Script to be run on a mouse click
ondblclick	script	Script to be run on a mouse double-click
ondrag	New script	Script to be run when an element is dragged
ondragend	New script	Script to be run at the end of a drag operation
ondragenter	New script	Script to be run when an element has been dragged to a valid drop target
ondragleave	New script	Script to be run when an element leaves a valid drop target
ondragover	New script	Script to be run when an element is being dragged over a valid drop target
ondragstart	New script	Script to be run at the start of a drag operation
ondrop	New script	Script to be run when dragged element is being dropped
onmousedown	script	Script to be run when a mouse button is pressed
onmousemove	script	Script to be run when the mouse pointer moves
onmouseout	script	Script to be run when the mouse pointer moves out of an element
onmouseover	script	Script to be run when the mouse pointer moves over an element
onmouseup	script	Script to be run when a mouse button is released
onmousewheel	New script	Script to be run when the mouse wheel is being rotated
onscroll	New script	Script to be run when an element's scrollbar is being scrolled

2.7.5.5 Media Events

Eventi generati da elementi multimediali come video, audio o immagini. Gli attributi vengono applicati a tutti gli elementi ma è più comune vengano applicati ai tags multimediali come <audio>, <embed>, , <object> e <video>:

Attribute		Value	Description
onabort		script	Script to be run on an abort event
oncanplay	New	script	Script to be run when media can start play, but might has to stop for buffering
oncanplaythrough	New	script	Script to be run when media can be played to the end, without stopping for buffering
ondurationchange	New	script	Script to be run when the length of the media is changed
onemptied	New	script	Script to be run when a media resource element suddenly becomes empty (network errors, errors on load etc.)
onended	New	script	Script to be run when media has reach the end
onerror	New	script	Script to be run when an error occurs during the loading of an element
onloadeddata	New	script	Script to be run when media data is loaded
onloadedmetadata	New	script	Script to be run when the duration and other media data of a media element is loaded
onloadstart	New	script	Script to be run when the browser starts to load the media data
onpause	New	script	Script to be run when media data is paused
onplay	New	script	Script to be run when media data is going to start playing
onplaying	New	script	Script to be run when media data has start playing
onprogress	New	script	Script to be run when the browser is fetching the media data
onratechange	New	script	Script to be run when the media data's playing rate has changed
onreadystatechange	New	script	Script to be run when the ready-state changes
onseeked	New	script	Script to be run when a media element's seeking attribute is no longer true, and the seeking has ended
onseeking	New	script	Script to be run when a media element's seeking attribute is true, and the seeking has begun
onstalled	New	script	Script to be run when there is an error in fetching media data (stalled)
onsuspend	New	script	Script to be run when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate	New	script	Script to be run when media changes its playing position
onvolumechange	New	script	Script to be run when media changes the volume, also when volume is set to "mute"
onwaiting	New	script	Script to be run when media has stopped playing, but is expected to resume

2.7.6 Sicurezza

Utilizzando gli script è facile che un malintenzionato ottenga informazioni dal vostro documento che non si vuole siano condivise oppure è facile possa modificare l'elaborato a suo piacimento per una determinata ragione.

Per questo è importante implementare metodi di sicurezza quando si adottano script nella propria pagina.

Alcuni accorgimenti vengono presi automaticamente: per esempio l'oggetto `window` non esegue script la cui origine effettiva è diversa da quella dell'oggetto `document` che contiene.

Altri accorgimenti vanno presi dall'utente, implementando metodi di controllo come CAPTCHA per esempio, non necessariamente tramite il linguaggio javascript.

Lascio però una disamina approfondita dell'argomento al lettore per evitare di allontanarmi troppo dal tema principale di questo elaborato.

3. NOVITÀ

3.1 *Introduzione*

HTML5 è stato introdotto e quasi tutte le sue caratteristiche sono state descritte dettagliatamente. Tuttavia, leggendo semplicemente le specifiche è difficile individuare quali sono tutte le novità introdotte da questa nuova versione, il motivo per il quale dovrebbero essere applicate o quale reale impatto dovrebbero avere sul nostro documento.

Il modo di pensare fino ad ora adottato dal tipico programmatore HTML dovrà cambiare radicalmente?

La risposta a questa domanda non è univoca e deve considerare a che livello il programmatore si interfaccia con questo linguaggio. Non tutti gli aspetti di HTML hanno subito stravolgimenti tali da imporre questo radicale cambiamento ma è importante notare come anche le novità più piccole siano frutto di un'attenta analisi della realtà che il web sta affrontando negli ultimi anni.

La nuova versione di HTML è sicuramente molto più adatta per affrontare questa realtà rispetto alla precedente, anche senza che ci addentriamo nell'implementazione delle features più complesse.

Dando invece una risposta più attenta alle esigenze dei programmatori più avanzati, mi sento di dire che questo radicale cambiamento nel modo di pensare non si potrà evitare. La struttura della pagina, la gestione dei forms, l'utilizzo degli script, sono tutti argomenti determinanti nella creazione di un documento HTML e tutti sono stati rivisitati con grosse innovazioni. Le novità introdotte non vanno solo ad aggiungere funzionalità e caratteristiche al linguaggio HTML fin'ora conosciuto ma vanno a modificare profondamente le modalità con cui esso dovrà essere utilizzato e a che scopo.

Nei prossimi paragrafi vedremo un'analisi dettagliata delle novità introdotte, con numerosi riferimenti alle specifiche e esempi per mostrare i loro effetti sulle future pagine HTML.

3.2 Struttura del documento

Tutti gli elementi possibili devono chiaramente indicare il tipo di contenuto al loro interno, rendendo il documento di più facile lettura e di conseguenza più facilmente modificabile anche a distanza di molto tempo dalla sua creazione.

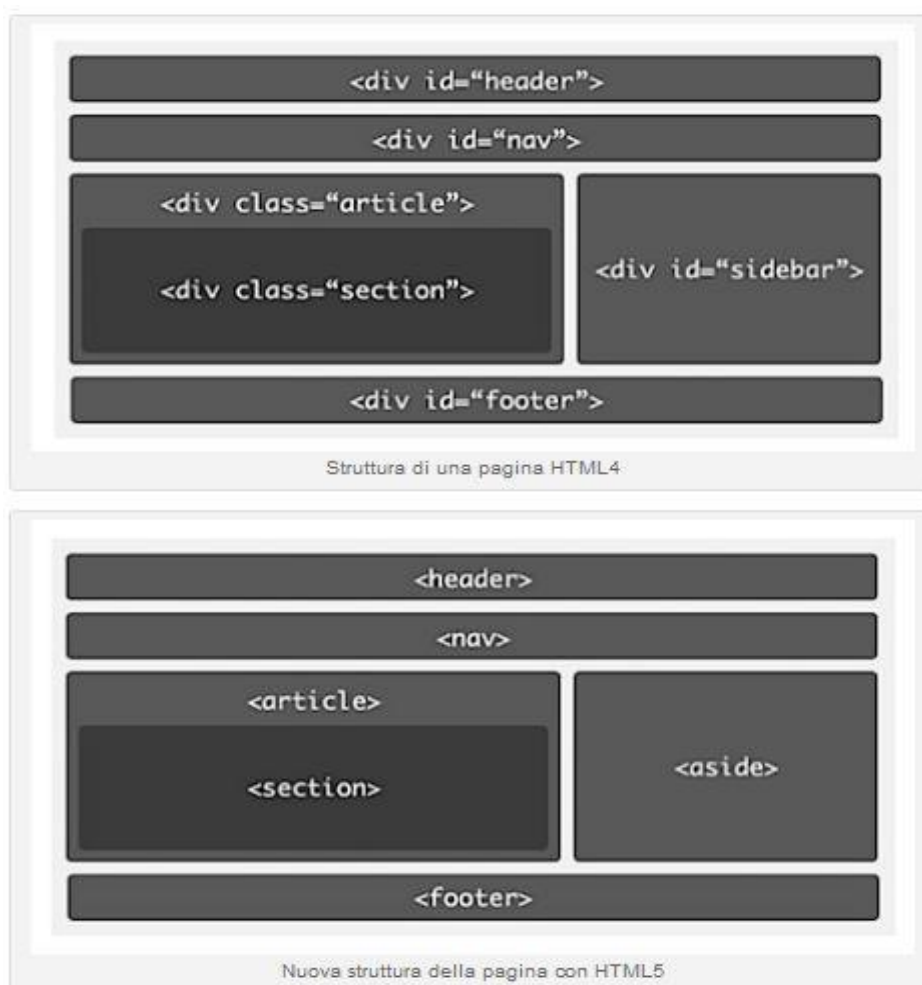
Nella precedente versione, per conferire un certo significato ad un tag, si ricorreva spesso all'utilizzo di elementi `div` a cui veniva associato un `id` avente come valore la descrizione del tipo di tag rappresentato da `<div>` stesso.

Questo creava diversi problemi in quanto ogni autore interpretava ovviamente queste dichiarazioni in maniera personale.

HTML 5 standardizza una molteplicità di elementi capaci di definire univocamente che contenuto essi devono contenere e che utilizzo il programmatore deve farne.

Al fine di descrivere meglio i nuovi tipi di tags, nelle specifiche come abbiamo visto è stato rivisitata anche la struttura del content model.

Vediamo di seguito un esempio di come la struttura è cambiata in questa nuova versione:



Notiamo subito come si sia semplificata sia la dichiarazione dei vari tag nonché la struttura stessa che essi creano. La pagina risulta sicuramente più ordinata e andarla ad analizzare e capire, anche se non la si è creata personalmente, ora sarà sicuramente più facile e immediato.

La definizione dei nuovi tags <footer>, <header>, <aside> etc. di cui abbiamo parlato nel precedente capitolo non va tuttavia analizzata superficialmente. Elementi già presenti in passato cambiano il loro comportamento e vengono interpretati diversamente dai browser ora che vengono applicati a tale struttura.

Vediamo l'esempio del tag <h*>. Prima per definire i titoli relativi alla pagina vera e propria, i suoi sottotitoli e i titoli dei suoi articoli dovevamo procedere nel seguente modo:

```
<h1>Titolo del sito</h1>
<h2>sottotitolo del sito</h2>
<h3>Titolo del post sull'africa</h3>
<h4>Sottotitolo del post</h4>
<h3>Titolo del post sulla nuova BMW</h3>
<h4>Sottotitolo del post</h4>
```

Per capire a che punto del documento ci riferivamo dichiarando l'<h*> dovevamo tener conto unicamente dell'indice del tag, il che rendeva tutto molto complicato nel caso stessimo trattando una pagina molto estesa e complessa.

Mettiamo poi il caso che volessimo riorganizzare i post secondo gli argomenti da essi trattati; avremo dovuto dichiarare un nuovo <h3> in questo caso per dare il titolo alla sezione, per esempio quella sui motori, quindi cambiare gli indici di tutti i post in modo che risultassero gerarchicamente sotto al nuovo <h>.

Ora ogni sezione tratta i tag al suo interno in modo indipendente e il browser interpreta correttamente la struttura di tali documenti. Riferendoci al precedente esempio:

```
<header>
  <hgroup>
    <h1>Titolo del sito</h1>
    <h2>sottotitolo del sito</h2>
  </hgroup>
</header>
<article>
  <hgroup>
    <h1>Titolo del post sull'Africa</h1>
    <h2>Sottotitolo del post</h2>
  </hgroup>
</article>
```

Notare la nuova semplicità ottenuta tramite la definizione dei nuovi tags.

La definizione di elementi semanticamente più stringenti permette anche una maggiore precisione nella definizione di nuovi attributi, molto più specifici rispetto alle nuove esigenze degli elementi.

3.3 *La nuova sintassi*

La nuova sintassi adottata da HTML5 si caratterizza per una maggiore flessibilità e semplicità. Non ha subito innovazioni eclatanti ma comunque è stata revisionata rispetto a quella definita nelle precedenti specifiche.

Un punto di interesse riguarda sicuramente la nuova dichiarazione del DOCTYPE.

Di seguito vediamo come è stata giustamente semplificata:

```
<!DOCTYPE html>
```

Prima era possibile dichiararne di tre tipi attraverso una dichiarazione come questa:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Le dichiarazioni degli elementi rimangono case-insensitive, gli attributi se vuoti vengono auto completati dal browser nella maniera descritta nelle specifiche e in alcuni casi è possibile che i tag non vengano chiusi pur rimanendo interpretabili correttamente. Tutto questo è descritto approfonditamente nel paragrafo sulla sintassi del capitolo precedente.

3.4 *I nuovi forms*

I forms sono stati oggetto di una doppia rivisitazione da parte di W3C: una si è concentrata sulla creazione di nuovi attributi atti ad aggiungere ad essi funzionalità e a migliorarne la struttura in modo analogo a quello esposto precedentemente, l'altra si è concentrata sull'aggiunta di attributi che semplificassero le operazioni di validazione dei suoi dati e riducessero il bisogno di utilizzare script al suo interno.

Si noti come ho parlato solo di attributi. La creazione di nuovi elementi infatti non è stata esplicita ma implicitamente determinata dall'aggiunta di molti nuovi valori per l'attributo 'type' del tag <input> che in pratica modella questo elemento.

Non ho parlato di nuovi elementi anche perchè molti elementi strutturali sono stati già trattati precedentemente e non sono applicabili solamente ai forms quindi non possono essere considerati novità strettamente correlate ad essi.

Altri elementi come <video> sono nuovi ma anche in questo caso gli ho trascurati in quanto gli considererò in un paragrafo a parte dedicato alle nuove funzionalità multimediali di HTML5. Ovviamente questo tipo di elementi sarebbero potuti benissimo essere parte integrante di questo paragrafo visto che ricoprono spesso un ruolo principale all'interno dei forms.

Vediamo nel dettaglio quali sono le nuove principali funzionalità che possiamo ottenere dai nostri e quali attributi possono permetterci di ottenerle.

3.4.1 Nuove funzionalità e accorgimenti strutturali

Meramente strutturali sono le novità introdotte dal nuovo attributo 'form' che ci permette di specificare su ogni input a che form o a quali forms esso appartiene. Il suo utilizzo semplificherà di molto la dichiarazione degli input nel nostro documento qualora uno di essi debba essere presente senza modifiche in più forms. Ora sarà possibile, nel caso risulti conveniente, specificare un solo elemento di input e inserirlo in più forms senza doverlo dichiarare in ognuno di essi.

Gli attributi autofocus e placeholder ci permettono invece di aggiungere alcuni accorgimenti al nostro form, rispettivamente decidere che esso evidenzia la presenza di un determinato elemento al suo interno o completa il campo di un determinato elemento input con interessanti istruzioni per l'utente prima di venire compilato. Queste due novità sicuramente non sono determinanti ma contribuiscono a caratterizzare meglio quelle che possono essere le nostre esigenze.

Interessante anche l'inserimento del nuovo valore "color" per l'attributo 'type' dell'elemento <input>. Specificando tale valore possiamo comodamente mettere a disposizione dell'utente una tavolozza sulla quale permettergli di scegliere il colore da lui desiderato. Per dar vita ad una simile feature sul nostro sito prima si sarebbe dovuto ricorrere a delle soluzioni con javascript. Se bisogna essere sinceri per ora la situazione non è cambiata in quanto la maggior parte dei browser non supporta questo nuovo tipo di input.

3.4.2 La validazione

I controlli sulla validazione implementati dai prossimi attributi, nella precedente versione di HTML, erano implementabili solo grazie all'ausilio di scripts.

La loro presenza in HTML5 è una delle novità più importanti introdotte da W3C ed uno degli aspetti che compongono quel grande progetto atto a rendere HTML sempre

più indipendente rispetto ai contenuti o alle applicazioni esterne che fino ad ora sono stati indispensabili parti integranti per la realizzazione di documenti completi e sicuri.

Fra gli elementi più ritoccati il tag `<input>`. Almeno la metà dei valori assumibili dal suo attributo `'type'` ed elencati nelle specifiche sono assolute novità. La definizione di questi nuovi tipi di oggetto implementa al suo interno specifiche ben precise su che tipo di dato ci si aspetta rappresentino. Se l'utente digita un contenuto non idoneo l'input non sarà accettato.

I casi più evidenti riguardano sicuramente la definizione dei nuovi tipi `"email"`, `"number"`, `"range"` e `"tel"`.

Prendendo il tipo `"email"`, prima della sua introduzione nelle specifiche l'utente doveva eseguire uno script per controllare ad esempio la presenza di una `"@"` all'interno del testo inserito dall'utente. Ora questo controllo viene implementato automaticamente dal tag `<input>`.

Con `number` possiamo specificare la sensibilità del numero che vogliamo si inserisca, con `range` l'intervallo di valori che lo comprende, con `tel` il fatto che sia un numero intero etc.

W3C non si è limitata solo al miglioramento dell'attributo `'type'` aggiungendo molte altre novità.

Fra queste sicuramente l'attributo `'required'` che se presente obbliga l'utente a compilare il campo al quale è stato associato, un vero must all'interno dei forms usati per le registrazioni degli utenti.

Da non sottovalutare anche l'attributo `'autocomplete'` con il quale possiamo evitare che il campo in questione venga autocompletato dal browser; questo ci eviterà l'introduzione di molti scripts.

Diventerà sicuramente fondamentale per ogni elemento del form che lo supporta l'attributo `'pattern'`, con il quale possiamo specificare personalmente il tipo di dato che vogliamo l'utente inserisca nel campo. Di seguito vediamo ad esempio come controllare che il dato sia un insieme di lettere minuscole e `"_"`, con il primo carattere obbligatoriamente alfabetico:

```
<input type="text" name="nickname" autocomplete="on"
  required pattern="[a-z]{1}[a-z_]{2,19}"
  title="Un nickname è composto da lettere minuscole e '_'; Sono consentiti da 3
  a 20 caratteri."
  placeholder="your_nickname">
```

Il tutto come al solito comodamente inserito come attributo di `<input>`.

Sono stati introdotti anche gli attributi ‘min’ e ‘max’ con i quali definire un range di valori per il dato dell’input. Ovviamente questi non sono tanto utili per un numero, visto il nuovo valore “range” di ‘type’, ma per definire range applicati a date o orari.

Da questo paragrafo è ovvio come W3C sia venuta notevolmente incontro al programmatore e come abbia reso HTML molto più indipendente da javascript. Tuttavia non bisogna pensare che questi accorgimenti bastino per rendere sicuro il form al quale stiamo lavorando. Gli script sono ancora importantissimi, viene solo ridotto il loro numero e quindi semplificata anche la loro implementazione.

Proprio di quest’ultimo argomento tratta il prossimo paragrafo.

3.5 *Lo scripting*

Nelle nuove specifiche è chiaro che non solo la necessità dello scripting è stata notevolmente ridotta, ma che dove esso rimane indispensabile il nuovo HTML5 fornisce features capaci di semplificarne l’implementazione.

A mio avviso, leggendo l’omonimo paragrafo del precedente capitolo sulle specifiche, questo salta subito all’occhio.

Se lo scripting relativo alla validazione dei dati è stato reso meno indispensabile rispetto al passato, l’utilizzo di quello relativo alla dinamicità della pagina è stato invece migliorato notevolmente.

La principale necessità a cui si va incontro nel momento in cui vogliamo rendere l’HTML un linguaggio dinamico è sicuramente quella di permettere che le pagine web possano reagire all’accadimento di determinati eventi al loro interno. Per questo W3C nelle nuove specifiche ha inserito un numero impressionante di attributi per la gestione degli eventi, utili per catturare praticamente ogni tipo di avvenimento e alla sua manifestazione richiamare l’opportuna funzione di scripting.

Fra i principali possiamo ricordare *onerror* (lo script viene chiamato quando si manifesta un errore), *onpageshow*, *ondrag* (vedremo la funzionalità di drag in seguito), *onpause*, *onplay* etc.

Nella precedente versione di HTML quasi l’80% degli attributi elencati nelle odierne specifiche non erano presenti. Questo significava che per gestire quel tipo di eventi spesso bisognava creare personalmente l’event handler, ad esempio con metodi del DOM ma non solo in questo modo.

La scomodità nell'utilizzo di tali metodi si ripercuoteva poi sulla leggibilità e sulla semplicità del codice HTML del documento. La creazione di eventi non standard poteva diventare difficilmente interpretabile da programmatori diversi dal "originale" e aumentava il numero di dichiarazioni nella pagina complicando il codice sorgente. Ora la maggior parte degli eventi può essere facilmente gestita attraverso i nuovi attributi e di seguito richiamo per comodità un esempio nel quale si può vedere come viene tipicamente implementato un event handler con relativa funzione di script:

```
<html>
  <head>
    <script type="text/javascript">
      function displayDate()
      {
        document.getElementById("demo").innerHTML=Date();
      }
    </script>
  </head>
<body>
  <h1>My First Web Page</h1>
  <p id="demo"></p>
  <button type="button" onclick="displayDate()">Display Date</button>
</body>
</html>
```

3.6 Multimedialità

Il campo della multimedialità è quello che ha subito, insieme a quello dello scripting e della struttura, le maggiori nonché le più importanti modifiche.

Anche in questo caso W3C vuole fare un notevole passo in avanti verso l'indipendenza, stavolta rispetto a player esterni e plugin del browser.

Video e audio ora hanno un tag a loro dedicato: <video> e <audio> rispettivamente. Prima entrambi dovevano essere implementati tramite un tag <embed>.

Il player esterno che fino ad ora era stata la principale soluzione per la visualizzazione dei video è stato sicuramente Adobe Flash Player.

Perché è conveniente rendere video e audio elementi nativi del documento?

- In primo luogo molti dispositivi mobili, fra cui ad esempio iPhone e iPad, non supportano la tecnologia Flash. Il fatto che questo sia stato considerato dalle nuove specifiche conferma il fatto che W3C abbia posto molta attenzione sulle nuove realtà che il Web sta affrontando e affronterà. Da non sottovalutare anche il fatto che uno dei padri di HTML5 è stata proprio la Apple.
- Trattandosi di un elemento nativo, il player si scarica in teoria più velocemente e le prestazioni risultano più efficienti.

- Il player del video potrà essere gestito direttamente dal programmatore mediante linguaggio Javascript ma non solo. Questo semplifica molto il compito del programmatore durante la definizione delle caratteristiche da applicare al video nonché dei sottotitoli ad esso relativi. Flash era infatti un linguaggio abbastanza complicato.
- Definendo un nuovo tag specifico per questi elementi multimediali si facilita la loro indicizzazione all'interno dei motori di ricerca.

Parlando di velocità di scaricamento e di prestazioni più efficienti si potrà notare come abbia scelto un approccio teorico. Di fatto, in pratica, questi miglioramenti per ora non si sono notati in generale ma su internet si possono comunque notare esempi particolari nei quali risultano evidenti. Uno di questi è visualizzabile sul sito “<http://sublimevideo.net/demo>”. Notare come la dimensione del video a schermo intero dipenda dalla dimensione della finestra del browser.

Di recente anche YouTube ha iniziato a inserire video con questo tipo di tecnologia e su internet proliferano prototipi di video player creati con linguaggi JavaScript e CSS.

Non avendo affrontato giustamente nel dettaglio il linguaggio javascript nel corso di questo elaborato, lascio l'approfondimento dei metodi per creare un player all'utente. In tutti i casi vediamo una semplice implementazione del tag video. L'attributo ‘controls’ è indispensabile per visualizzare i controlli per gestire il video. L'utilizzo del tag audio è analogo.

```
<video class="projekktor" src="nome_video.mp4" width="435" height="272"
poster="immagine_presentazione.png" title="titolo" controls></video>
```

In questo caso abbiamo importato delle librerie javascript relative al player projekktor, disponibile gratuitamente sul sito “projekktor.com”, che dobbiamo specificare come valore dell'attributo ‘class’. L'attributo ‘src’ definisce l'URL del video, ‘poster’ quello dell'immagine che voglio appaia come anteprima finché non viene fatto partire. Nella prossima pagina possiamo osservare uno screenshot relativo proprio all'esempio sopra esposto.



```
<video class="projekktor" src="intro.mp4"
width="435" height="272" poster="intro.png"
title="kewl" controls></video>
```



- ✓ Pre- & Postroll Ads
- ✓ [Playlists & Channels](#)
- ✓ [Plugins & Javascript API](#)
- ✓ [Flash fallback for ancient browsers](#)

[MORE DEMOS](#)[GET IT NOW](#)

Il supporto per i tags `<video>` e `<audio>` non è ancora presente su tutti i browser ma sicuramente non tarderà ad arrivare.

L'introduzione di questa nuova feature è il principale motivo per il quale in molti considerano HTML5 il futuro del web.

3.7 Geolocalizzazione

L'aspetto della geolocalizzazione non è stato toccato nelle specifiche ed è stato solo accennato durante l'introduzione. Tuttavia, rappresenta una novità molto interessante ed è sicuramente l'ennesima prova di quanto questo nuovo linguaggio sia sensibile alle nuove esigenze nel mondo del Web.

I social network stanno spopolando in questo momento storico ed è sempre più diffusa la navigazione su internet tramite smartphone, ovvero dispositivi ultraportatili che localizzano la persona che li utilizza durante tutto il corso della giornata.

Alla luce del nuovo modo con il quale l'utente medio si interfaccia alle pagine web, W3C decide ancora una volta di andare incontro al programmatore, rendendo disponibile questa nuova feature.

3.7.1 Implementazione

La funzione di GeoLocation prevede che il browser visualizzi le coordinate geografiche della posizione in cui si trova l'utente. Tali coordinate vengono ricavate dal sistema operativo su cui è installato il browser, sempre che il dispositivo portatile sia provvisto di GPS o WiFi per la localizzazione.

Per utilizzare questa nuova caratteristica bisogna però fare molta attenzione. Prima di ogni sua esecuzione va inoltrata all'utente una richiesta specifica con la quale gli si chiede se ci concede il permesso di localizzare la sua posizione.

La funzione di geolocalizzazione va implementata tramite l'utilizzo di linguaggio JavaScript e relative API.

```
<script>
function callback(pos) {
    var latitudine = pos.coords.latitude;
    var longitudine = pos.coords.longitude;
    alert('lat: '+latitudine+', long: '+longitudine);
}
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(callback);
} else {
    alert("Spiacente, il tuo browser non supporta la GeoLocation");
}
</script>
```

In questo estratto di codice si definisce una funzione callback richiamata nel caso il browser supporti feature 'geolocation' di HTML5.

In questo estratto si utilizzano le API 'latitude' e 'longitude', ma non sono le uniche disponibili in questo ambito:

- *altitude*: altitudine;
- *accuracy*: precisione delle misure di longitudine e latitudine;
- *altitudeAccuracy*: sensibilità della misura dell'altitudine;
- *heading*;
- *etc.*

Soprattutto gli attributi di accuracy potrebbero essere utili per specificare controlli sull'effettiva ricezione del GPS dell'utente che vogliamo localizzare.

3.7.2 Conclusioni

La geolocalizzazione si rivela una piacevole sorpresa. I possibili utilizzi sono innumerevoli e l'implementazione è semplice e immediata.

Possiamo ad esempio localizzare l'utente e proporgli una via per raggiungerci in base a dove si trova, localizzarlo per segnalare la sua posizione su un social network, implementare interessanti applicazioni con le quali localizzare l'utente e segnalargli quali servizi o punti di interesse si trovano nei suoi paraggi...

La nuova funzione di GeoLocation è rimane chiaramente un passo avanti enorme soprattutto se rapportato, come detto prima, alla diffusione massiccia di dispositivi ultraportatili con accesso ad internet.

3.8 Drag and drop

L'operazione di drag and drop è una delle operazioni più basilari nell'esperienza informatica di ognuno di noi.

Rappresenta il trascinamento di un oggetto all'interno di un contenitore target.

Fino a questo momento il drag and drop veniva implementato tramite complesse funzionalità create in linguaggio Javascript. HTML5 definisce delle API per dar vita ad un ambiente nativo dove sviluppare tale funzione.

3.8.1 Implementazione

Ogni elemento che sarà possibile trascinare dovrà avere attributo *'draggable = "true"'* e di conseguenza ogni contenitore dentro il quale un elemento potrà essere trascinato dovrà avere attributo *'droppable="true"'*.

Gli elementi `<img>` e `<a>` aventi attributo href valido hanno *'draggable'* settato di default uguale a true.

Dopo aver definito sia gli elementi draggable che quelli droppable possiamo gestire il comportamento del documento utilizzando gli attributi per la gestione degli eventi a disposizione in HTML5, ovvero *ondrop, ondrag, ondragover etc.*

Analizziamo come implementare la funzione partendo da un esempio per semplificare le cose:

```






<div id="field"></div>
```

```
#field { height: 200px; width: 200px; background: green; color: snow }
img { border: 3px solidblack; cursor: move }
img.no-drag { cursor: default }
```

In questo primo estratto abbiamo definito gli oggetti trascinabili e un riquadro che però non è ancora stato dichiarato droppable. Nel prossimo estratto gestiamo il trascinamento delle immagini.

Prima di tutto dovrò settare il campo *field* come target dei miei trascinamenti disattivando il comportamento di default previsto dal browser per l'evento *ondragover* che indica il momento in cui termina l'operazione di trascinamento dell'oggetto:

```
field.ondragover = function(event) {  
    return false;  
}
```

Ora specifico cosa voglio che accada quando rilascio un oggetto nel target:

```
field.ondrop = function(event) {  
    alert("Player dropped");  
}
```

L'evento *ondrop* per ora l'abbiamo specificato in relazione con il campo *field*, come si può notare dalla dichiarazione. Ora dobbiamo determinare quale oggetto è stato trascinato nel target. Il browser ci viene in contro includendo nel parametro *event* passato alla funzione l'oggetto *dataTransfer*.

Specifichiamo cosa ogni elemento inserirà al momento del rilascio all'interno di tale oggetto per poi individuare quale di essi è stato trascinato nel target. Posso farlo semplicemente specificando un *title* nelle immagini.

```
  
  
  
  

```

Utilizziamo *ondragstart* per salvare il *title* dell'oggetto selezionato all'inizio del trascinamento:

```
var players = document.getElementsByTagName("IMG");  
for(var i = 0; i < players.length; i++) {  
    players[i].ondragstart = function(event) {  
        event.dataTransfer.setData("player", this.title);  
    }  
}
```

Modifichiamo quindi la funzione che parte al momento del rilascio ricavando al suo interno il nome dell'elemento inserito nel target:

```
field.ondrop = function(event) {  
    event.preventDefault();  
    var title = event.dataTransfer.getData("player");  
    this.innerHTML += title + "<br>";  
}
```

A seconda del nostro gusto possiamo anche modificare l'immagine che viene trascinata sotto il mouse durante l'operazione di drag 'n drop.

3.8.2 Conclusioni

Questa features risulta molto interessante e semplifica l'implementazione di una delle operazioni più basilari per un utente informatico. Tuttavia il suo utilizzo all'interno delle pagine web rimane abbastanza complicato se si vuole renderlo il più possibile accattivante.

Per questo la grande attenzione che in molti ripongono su questa novità non è da me particolarmente condivisibile.

L'utilizzo principale prevede che il drag 'n drop serva per rendere graficamente più immediato e accattivante il raggruppamento da parte dell'utente di un insieme di elementi.