POLITECNICO DI TORINO

Facoltà di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Sistemi web di content management con funzionalità di redazione distribuita

Relatori:
prof. Fulvio Corno
ing. Giovanni Squillero

Candidato:
Michele Debandi

Settembre 2003

Introduzione

Il *World Wide Web* ha modificato radicalmente sia il mondo dell'informatica e delle telecomunicazioni che il rapporto con l'informatica per i non addetti ai lavori: solo l'introduzione del *personal computer* ha avuto risultati comparabili.

HTML era nato come un sistema di *word processing* con possibilità di inserire collegamenti ipertestuali, scritto in Objective C per la *workstation* NeXT. Il sistema utilizzato, per la sua scarsa diffusione e la sua diversità rispetto ad altri tipi di elaboratori elettronici non poteva far pensare al successo travolgente ottenuto dal web.

L'adozione di un protocollo di rete aperto, il TCP/IP, l'aver reso liberamente disponibile le specifiche per il funzionamento di *client* e *server*, e la relativa semplicità del *markup* HTML, hanno permesso di scrivere facilmente *client* e *server web* per le principali architetture, facendo così operare assieme computer molto diversi fra loro senza problemi e soprattutto senza che gli utilizzatori dovessero compiere particolari operazioni. Il *web* ha avuto un successo che non era stato né previsto né preventivato, mettendo in luce le limitazioni di HTML come linguaggio di *markup* per rappresentare contenuti generici e per il controllo dell'aspetto delle pagine. L'aumento delle dimensioni dei singoli siti e del *web* nel suo complesso ha fatto nascere tutta una serie di nuovi problemi organizzativi.

La prima versione di HTML aveva un insieme limitato di elementi: per estendere le sue funzionalità si sono aggiunti altri elementi ed attributi, spesso interpretati in maniera diversa da ogni singolo *browser*. Si sono sfruttati gli elementi per scopi completamente diversi da quelli per cui essi erano stati pensati per ottenere particolari effetti grafici, ad esempio tabelle ed immagini *bitmap* spesso sono utilizzate nell'allineare testi e grafica. Aggiungendo a questo che i *browser* cercano di visualizzare in ogni caso anche codice HTML completamente sbagliato, si è avuta la proliferazione di pagine che sono un ammasso di *tag* senza senso, difficili da aggiornare e da analizzare, spesso visibili correttamente solo dal *browser* sul *computer* del *webmaster* o quasi.

Le dimensioni sempre maggiori dei siti *web* hanno però creato altri problemi. I visitatori possono aver difficoltà a trovare le informazioni che desiderano, perché non esiste una struttura coerente di un sito, le pagine hanno informazioni vecchie od i

collegamenti ipertestuali non sono più validi, le pagine hanno un aspetto confuso, oppure perché le informazioni cercate non esistono proprio. Chi mantiene il sito deve fare un'enorme fatica a mantenerlo aggiornato e la complessità della sua struttura rende impossibile a chi fornisce le informazioni poter intervenire direttamente: tutto ciò che viene pubblicato deve essere gestito manualmente da poche persone .

Una soluzione ai problemi gestionali di un sito web può essere data dall'adozione dei *content management system* (CMS). Lo scopo dei CMS è di recuperare, gestire e pubblicare il contenuto su uno o più mezzi di comunicazione, come ad esempio il web, la carta stampata oppure le produzioni audiovisive. In alcuni casi è possibile l'uso degli stessi contenuti per più pubblicazioni diverse.

Trattandosi di applicazioni abbastanza nuove non si ha una definizione univoca e complessiva di che cosa sia un *content management system*: sono disponibili applicazioni molto diverse tra loro sia negli scopi che nelle funzionalità che offrono, con costi che variano tra i 200 € ed i 200.000 €, senza contare i programmi open source. Non è comunque possibile utilizzare un CMS senza prima personalizzarlo ed adattarlo alle proprie esigenze. Molte soluzioni costruite "in casa" per risolvere i problemi del proprio sito web si possono considerare dei sistemi di gestione del contenuto *ad hoc*, tanto che diversi *content management system* liberi e commerciali sono nati inizialmente per essere utilizzati all'interno della propria azienda e poi resi disponibili anche al suo esterno.

Per comprendere meglio i problemi legati all'adozione di un CMS per pubblicare i propri contenuti sul *web* è utile, anziché un approccio teorico, andare a studiare un sito reale. Le effettive necessità degli utilizzatori, le limitazioni dettate dai sistemi informatici esistenti ed il tipo di contenuto da pubblicare influenzano pesantemente la scelta e le modalità operative di un *content management system*. L'obiettivo di questo lavoro è quello di analizzare e successivamente adattare sistemi di gestione del contenuto esistenti nel contesto pratico del sito di informazione sull'handicap del Comune di Torino.

Il sito attualmente usa già un sistema di gestione del contenuto, basato su una serie di funzioni scritte con il linguaggio di *scripting* PHP e l'accesso ad una base di dati relazionale, ma la crescita e l' evoluzione del sito hanno reso necessaria una nuova versione, in particolare per l'aumento della complessità del flusso di lavoro per la pubblicazione dei contenuti e l'adozione di un motore di ricerca interno.

In una prima fase del lavoro si è valutato se fosse meglio estendere le funzionalità dell'attuale sistema di gestione del contenuto oppure se l'opzione migliore fosse di selezionare un CMS libero e successivamente personalizzarlo per la situazione specifica. Nella valutazione, le esigenze operative del sito e soprattutto la necessità di dover usare una configurazione *software* prefissata (*web server* Apache con PHP, base di dati Oracle 8.1.7) sono state quelle che hanno fatto decidere per la soluzione poi effettivamente adottata.

Dopo un'attenta analisi dei CMS liberi disponibili e delle loro richieste *software* si è stabilito che nessuno degli applicativi esaminati era facilmente adattabile al sistema di *hosting* disponibile e quindi si è deciso di adattare ed espandere l'applicazione esistente per le nuove esigenze.

Successivamente ad una prima fase di prove si è deciso si abbandonare la struttura di memorizzazione del contenuto utilizzata nel sistema di gestione del contenuto esistente, basata su diverse tabelle che contengono i testi delle varie pagine e posizione ed aspetto che devono avere, per passare ad un sistema di memorizzazione basato su XML, con il risultato sia di rendere l'applicazione più flessibile e potente che di semplificare il codice PHP.

XML è uno *standard* per la creazione di linguaggi di *markup*, così come lo è SGML. Quest'ultimo però è uno *standard* piuttosto complesso e non esistono molti strumenti in grado di gestire ed elaborare SGML generico: gli unici *markup* basati su SGML che hanno avuto una certa diffusione sono HTML e DocBook. XML nasce come semplificazione di SGML, eliminando le parti più difficili da elaborare, ma senza perdere in potenza espressiva: XHTML 1.0, che utilizza un *markup* XML, ha gli stessi elementi ed attributi di HTML 4.01 e si può passare da una sintassi all'altra con strumenti automatici. La maggior semplicità di XML ha fatto nascere molte applicazioni e librerie per i principali linguaggi di programmazione che possono elaborare XML generico. Sono stati definiti *markup* XML specifici per le più diverse applicazioni (*word processing*, fogli elettronici, grafica vettoriale, presentazioni multimediali, ...) e molti programmi sono in grado di leggere e scrivere XML.

Nel caso del CMS di Informahandicap Piemonte, l'adozione di XML ha permesso di utilizzare il *parser* XML presente in PHP per semplificare il codice per la visualizzazione delle pagine e per l'*editing*. Aggiungere nuovi elementi ed attributi non implica la modifica né del *database* né del codice che non si occupa della trasformazione da XML ad XHTML. L'*editor* basato su Javascript e *form* HTML viene anch'esso ottenuto trasformando XML in HTML: anche in questo caso per gestire l'aggiunta di elementi od attributi è necessario modificare solo una piccola parte del codice.

Il *web content management system* sviluppato presenta le seguenti caratteristiche principali:

- le pagine che vengono generate per i visitatori, oltre ad essere XHTML 1.0 valido, possono seguire le direttive per l'accessibilità W3C WAI a livello AAA ed è inoltre disponibile uno stile di visualizzazione ad alto contrasto per ipovedenti;

- è stato mantenuto un sistema di navigazione ed un aspetto delle pagine molto simile a quello del precedente sito dinamico;

- il sistema di gestione del contenuto permette di gestire i gruppi di lavoro, l'appartenenza degli utenti ai gruppi e le loro relazioni gerarchiche ed il flusso di lavoro per l'approvazione del contenuto;
- il sistema è in grado di gestire automaticamente le attività che ciascun redattore o giornalista deve compiere;
- si può fare in modo che contenuti approvati non appaiano immediatamente ma solo dopo una certa data, oppure che vengano eliminati automaticamente dopo una certa data;
- possono coesistere nel sistema più versioni dello stesso documento in modo da consentire di salvare versioni intermedie e di avere un archivio delle pagine pubblicate;
- si possono definire dei modelli di documento per una maggiore uniformità delle pagine del sito;
- viene utilizzato un *editor* XML per i modelli ed i contenuti basato su *form* HTML, utilizzabile con qualunque *browser* che esegua codice Javascript.

L'applicazione è stata pensata anche per essere espandibile ed aggiornabile nelle sue varie parti (motore di ricerca, *editor* del contenuto, codifica del contenuto,...) in modo da poter essere adattabile alle future esigenze del sito Informahandicap ed eventualmente evolvere in un *content management system* libero.

Nel *capitolo 1* vengono discusse la storia e l'evoluzione del *World Wide Web*, ed analizzati i problemi che sono sorti con la sua adozione massiccia.

Nel *capitolo 2* viene spiegato cos'è il contenuto, che cos'è *content management* e di quali funzioni normalmente dispongono i sistemi di gestione del contenuto. Viene quindi presentata una panoramica dei principali *content management system* liberi esistenti.

Nel *capitolo 3* si esaminano in dettaglio le attuali esigenze di gestione del contenuto del sito Informahandicap del servizio Passepartout del Comune di Torino, la sua storia ed i motivi che hanno portato a scrivere una nuova applicazione.

Nel *capitolo 4* viene descritto XML e le sue caratteristiche principali, l'utilizzo dei DTD per la definizione del contenuto accennando anche ad altri sistemi più avanzati come Xschema, RELAX NG ed XSLT e l'uso CSS per la presentazione dei documenti XML.

Nel *capitolo 5* si discute il funzionamento dell'applicazione creata e successivamente si spiegano le funzioni disponibili per la gestione del sito.

Nel *capitolo 6* infine si presentano le considerazioni finali, valutando il risultato ottenuto e le possibili linee di sviluppo future.

Indice

Elenco delle figure

Capitolo 1

Breve storia del World Wide Web

Il *World Wide Web*, la combinazione del protocollo HTTP e del linguaggio di *markup* HTML, ha in dieci anni completamente rivoluzionato sia il mondo dell'informatica e delle telecomunicazioni che il rapporto sociale con i mezzi di comunicazione e le possibilità espressive a disposizione di tutti. Le ricadute sull'economia sono state tali da creare nuove industrie, nuove figure professionali e nuove richieste di beni e servizi, che senza la presenza del *web* non sarebbero mai nate o non si sarebbero sviluppate così come sono oggi. Il rapporto con gli strumenti informatici da parte delle persone che non sono nell'area dell'*Information Technology* e degli appassionati si è modificato totalmente grazie a questa tecnologia, facendo però sorgere nuovi problemi legati all'utilizzo di questo nuovo mezzo di comunicazione.

1.1 Le origini

In un articolo del 1945 di Vannevar Bush contenuto in "Atlantic Monthly" venne descritto un sistema chiamato Memex, che doveva essere in grado di seguire collegamenti fra vari documenti utilizzando come supporto il microfilm, il primo oggetto teorico che utilizzava il concetto di ipertesto. Negli anni '60 Ted Nelson conia per la prima volta il termine *Hypertext* e nel 1967 Andy van Dam ed altri programmano *Hypertext Editing System* e *FRESS.* Tra il giugno ed il dicembre del 1980, Tim Berners-Lee, lavorando come consulente presso il CERN scrive *"Enquire-Within-Upon-Everything"*, un programma per scrivere note informative con la possibilità d'inserire collegamenti fra nodi arbitrari: ogni nodo aveva un titolo, un tipo ed una lista di collegamenti bidirezionali. Sempre Tim Berners-Lee, nell'ottobre del 1990, inizia a lavorare su un sistema di scrittura grafico ipertestuale con capacità di *editing* e di *browsing*, funzionante sulle *workstation* NeXT: il nome del progetto era "WorldWideWeb". Un mese dopo, il primo *web server* viene attivato e per fine dicembre 1990 erano disponibili le prime versioni funzionanti del programma.

Le *workstation* NeXT erano macchine relativamente poco diffuse e piuttosto diverse da altri sistemi di elaborazione: per favorire gli utilizzatori di altri sistemi nel marzo 1991 un *browser* a riga di comando, chiamato www, viene messo a disposizione ad un gruppo limitato di persone. Nell'agosto dello stesso anno sia www che WorldWideWeb (rinominato poi *Nexus* per non generare confusione tra il programma ed il concetto astratto di World Wide Web) programmi vengono resi disponibili via FTP. Il 12 dicembre il primo *web server* viene installato al di fuori dell'Europa, nello SLAC di Stanford, mettendo *online* una grande quantità di *abstract* di articoli di fisica per i ricercatori.

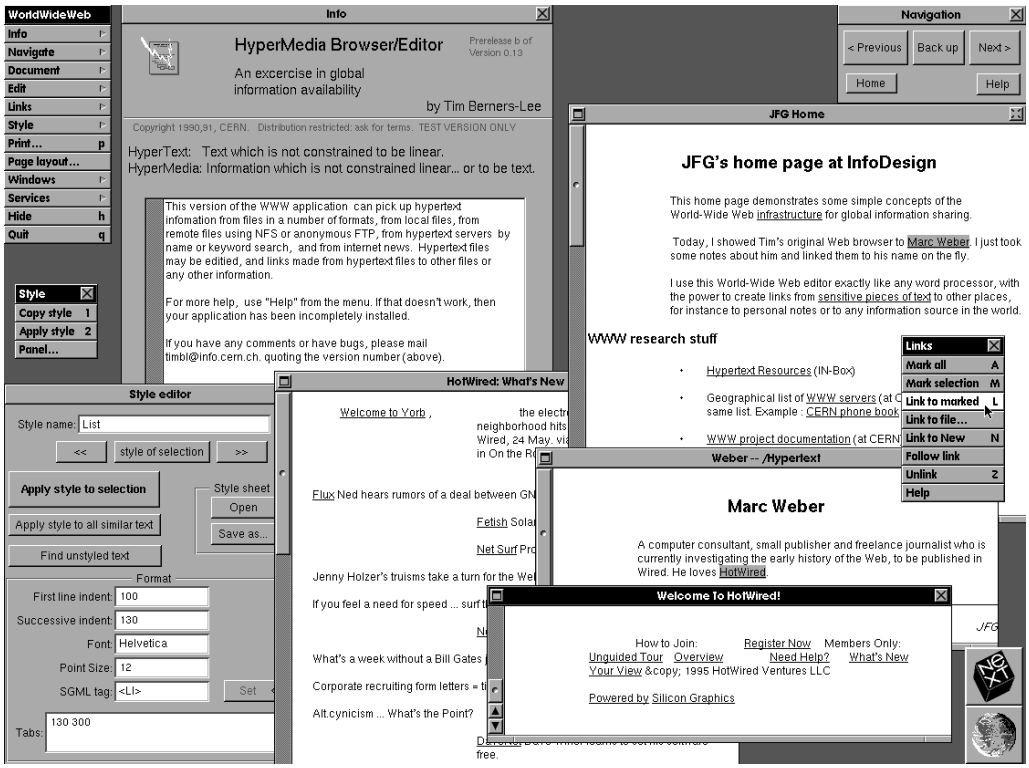


Figura 1.1. Schermata di una *workstation* NeXT con WorldWideWeb in esecuzione

A metà del '92 vengono scritti Erwise e Viola, altri due *browser* grafici. Contemporaneamente il sistema "Gopher" dell'università del Minnesota inizia ad avere una buona diffusione: rispetto ad un *web server* è più facile da installare e da gestire, non ha i *link* ipertestuali ma una struttura a menù. Sono pochi in questo periodo a pensare che il web sia meglio del Gopher. Mentre i Gopher in un anno si sono decuplicati, i *web server* si sono "solamente" triplicati. Nel gennaio 1993 viene rilasciato il primo *browser* per Macintosh, e nel febbraio la prima versione di

"Mosaic for X" viene rilasciata dall'NCSA. Rispetto al sistema su NeXT, Mosaic permette solamente la visualizzazione delle pagine, ma non più la loro modifica, iniziando così la suddivisione tra strumenti per la creazione e la fruizione del contenuto. Successivamente Netscape ha aggiunto funzionalità di *editing* a Netscape Communicator, con Composer, ma si tratta comunque di un programma separato rispetto a Navigator e poco integrato con quest'ultimo. L'unico programma che ha funzionalità di *editing* e *browsing* strettamente integrate oggi è Amaya del W3C. A marzo 1993 il traffico sulla porta 80 HTTP rappresenta lo 0,1% del traffico totale sul *backbone* NSF. Nel settembre dello stesso anno supera l'1%, mentre l'NCSA mette a disposizione versioni di Mosaic per X, Microsoft Windows e Macintosh.

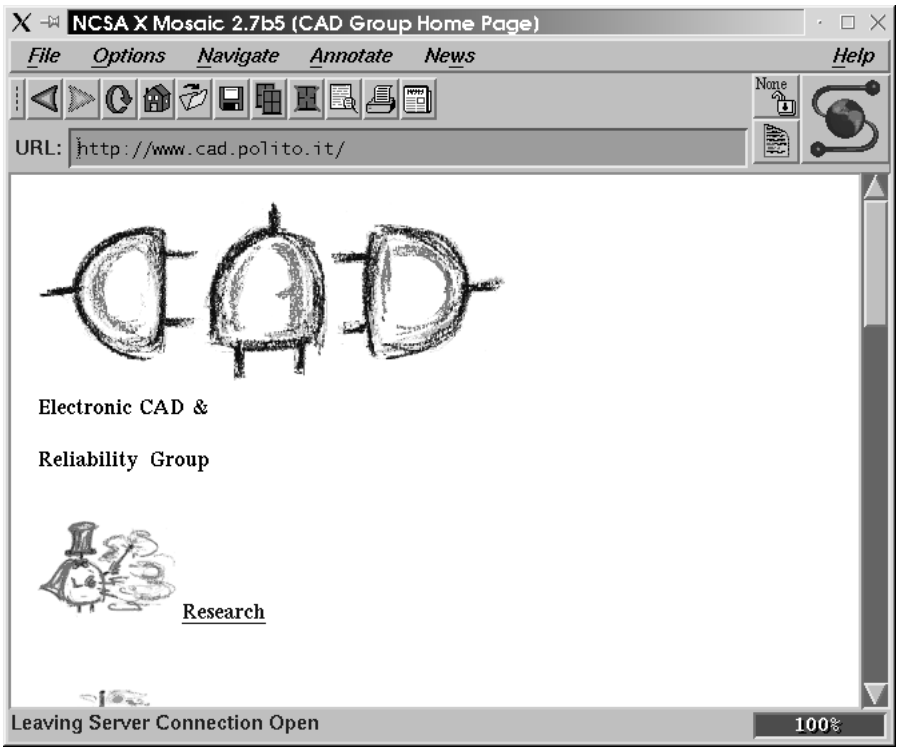


Figura 1.2. Immagine dell'ultima versione disponibile di Mosaic (2.7b5) in azione

Nel marzo del 1994 Marc Anderseen ed alcuni colleghi lasciano l'NCSA per fondare la "Mosaic Communications Corp", che diventerà in seguito Netscape Communications: il 10 di ottobre viene annunciata la prima versione di Netscape Navigator (0.9 beta). A giugno il carico sul primo *web server* del CERN è di mille volte superiore a quello di tre anni prima. MIT e CERN a luglio annunciano un accordo per la creazione della "W3 Organization", che il 14 dicembre si incontra, come "W3

consortium" al MIT a Cambridge (USA). Il 16 dicembre il CERN approva la costruzione dell'acceleratore di particelle LHC[1], ed a causa delle limitazioni di bilancio lo sviluppo del progetto WebCore passa all'INRIA[2].

Nel febbraio 1995 il *web* è il tema principale del G7 tenutosi a Brussel. Nel marzo successivo il traffico HTTP supera il traffico FTP. A maggio Sun Microsystems rende disponibile il suo *web browser*, HotJava ed introduce il linguaggio in cui è scritto, Java. Alla fine dell'anno Microsoft inizia a distribuire Internet Explorer all'interno dell' "Internet Jumpstart Kit" di Microsoft Plus! per Windows 95.

Lo sviluppo del World Wide Web è continuato fino ai nostri giorni in maniera esplosiva: esistono oltre trentasei milioni di siti raggiungibili e ha superato in maniera schiacciante altri metodi di accesso alle informazioni su Internet, in molti casi facendoli sparire o sopravvivere a stento: ad esempio Gopher non è più praticamente utilizzato ed è assai raro trovare sistemi disponibili al pubblico accessibili con telnet. Grazie alla sua diffusione ha fatto crescere moltissimo il TCP/IP, facendo diventare di nicchia altri protocolli di trasmissione dati, come DECNET ed SNA: sicuramente non fosse nato HTTP il mondo delle reti di calcolatori sarebbe stato molto diverso e molto più piccolo di quanto è oggi.

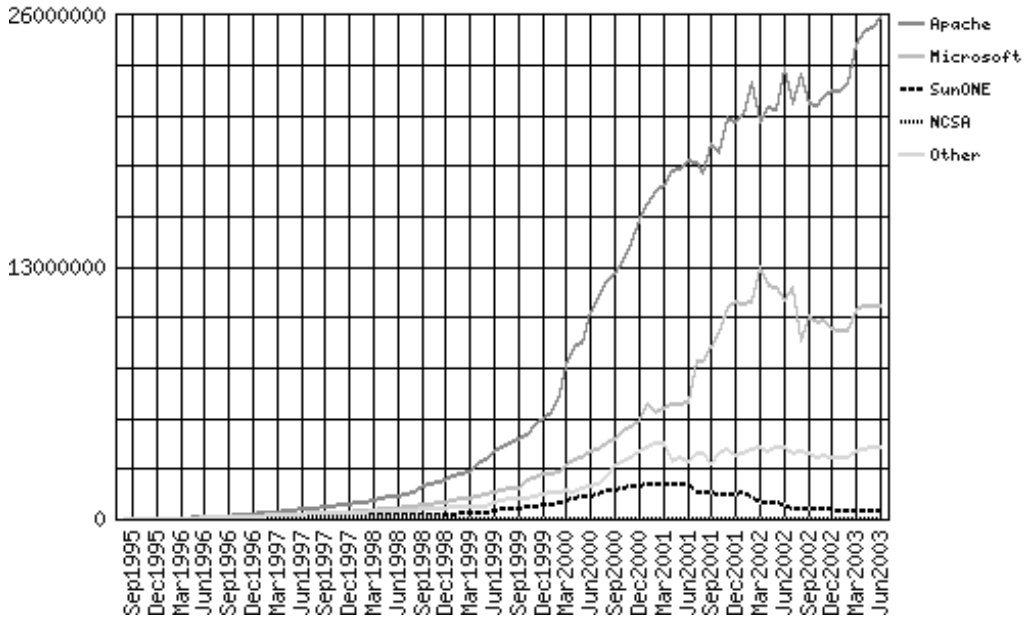


Figura 1.3. Crescita del numero dei siti web dal '96 ad oggi, suddivisa per *web server* utilizzato (fonte:Netcraft)

[1]Large Hadron Collider

[2]Institut National pour la Recherche en Informatique et Automatique

1.2 La rete per tutti

Come si è accennato poco fa, il *web* è nato come un sistema di *word processing* con la possibilità di inserire collegamenti ipertestuali all'interno dei documenti e soprattutto una modalità molto semplice di accesso via rete dei dati, causandone ha causato la crescita esplosiva: da un sistema all'inizio utilizzato principalmente in centri di ricerca sulle particelle ad alta energia si è arrivati alla diffusione capillare del giorno d'oggi. Si è creato un effetto moltiplicativo tale che l'aumento della domanda di connettività TCP/IP generato dall'uso di HTTP ha permesso di fare molti investimenti ed economie di scala sui sistemi di trasmissione dati. Si è avuta una diminuzione dei prezzi per l'accesso ad Internet, che ha attirato altre persone nella Rete, che a loro volta hanno fatto aumentare la domanda e così via. Tecnologie come ADSL oppure i modem analogici V.92 probabilmente non sarebbero nate se non ci fosse stata una richiesta per trasmissioni dati così imponente e così diffusa e "popolare". Prima dell'avvento del *web* per accedere ed inserire dati su Internet o su altri tipi di reti geografiche era necessario utilizzare più applicazioni diverse, con programmi differenti, con interfacce utente molto diverse fra di loro, e dall'uso spesso difficoltoso.

Il *web*, quindi si deve considerare anche per il suo aspetto sociale e non solo per quello tecnico. È un nuovo mezzo di comunicazione che rispetto ai *media* tradizionali ha costi di creazione e diffusione del contenuto decisamente più bassi. Contemporaneamente rende molto facile reperire le informazioni disponibili, senza porre barriere geografiche o temporali. Queste caratteristiche hanno attratto persone al di fuori del mondo dell'*Information Technology* e della ricerca o degli appassionati d'informatica. Altri sistemi pensati per l'utilizzo di massa o sono miseramente falliti o il loro utilizzo è attecchito solo nella cerchia degli "smanettoni". Della prima categoria il caso principale è stato il Videotel, in cui un terminale dedicato, comprensivo di modem, utilizzando la rete telefonica commutata e connessioni X.25 dava accesso agli utenti a pagine di testo con possibilità di interagire in maniera limitata con gli *host* che fornivano le informazioni. L'unico paese dove ha avuto diffusione è stata la Francia, ma solamente grazie a forti sovvenzioni governative, mentre in Italia la diffusione è stata praticamente nulla. Il secondo caso è fondamentalmente quello della tecnologia Fidonet: nodi gestiti da volontari, basati su *personal computer* e l'uso di linee commutate anche per l'interconnessione fra nodo e nodo, consente di gestire servizi di messaggistica e di scambio programmi. Pur avendo avuto una diffusione più capillare rispetto a sistemi simili nelle funzionalità ma utilizzabili solo su macchine multiutente, come ad esempio UUCP, è rimasta pressoché sconosciuta al grande pubblico.

Il WWW, pur non essendo nato con lo scopo di diffondere l'utilizzo della telematica a tutta la popolazione, è invece riuscito pienamente in questo scopo. Uno dei

fattori è stato che utilizza protocolli di comunicazione aperti, per cui è stato semplice scrivere programmi *client* per una grande varietà di macchine. La semplicità iniziale, unita anche a quella del TCP/IP ha permesso di scrivere *browser* anche per i *personal computer* del 1990. Si è creato un "linguaggio" comune nel mondo informatico che ha permesso di poter colloquiare direttamente con i sistemi più diversi senza grosse complicazioni.

La popolarità di HTML e l'utilizzo ben al di fuori dei suoi scopi iniziali ne ha messo in luce le limitazioni, che si sono dovute gestire e superare.

1.3 I problemi di HTML

Il primo *browser*, WorldWideWeb aveva la possibilità di utilizzare dei "fogli di stile" per modificare l'aspetto dei documenti, ma questa caratteristica si è persa nei *browser* successivi, cosicché per modificare l'aspetto grafico delle pagine HTML si è iniziato ad introdurre nuovi elementi e nuovi attributi per gli elementi già definiti al suo interno.

HTML ha una sorta di "peccato originale": quello di mescolare assieme elementi di presentazione (come deve apparire un documento) e di strutturazione (come è organizzato logicamente un documento) del contenuto. A peggiorare la situazione si è iniziato ad estendere ed utilizzare per la presentazione costrutti pensati per altri scopi. Ad esempio l'elemento `<table>`, nato per poter visualizzare dati tabellari, viene molto spesso utilizzato per mettere in posizione prefissata elementi grafici o parti di testo, parti testuali vengono inserite come *bitmap* per essere certi che il testo appaia con un carattere particolare o con un certo effetto, immagini vuote vengono messe per ottenere spaziature fisse o l'entità ` ` viene utilizzata per inserire spaziature fra parole. Si è anche proceduto al contrario: per sopperire alle limitazione dei *browser* più vecchi, come ad esempio Microsoft Internet Explorer 3 e Netscape 3, si sono spesso utilizzati degli elementi di presentazione per definire la struttura di un documento.

Molti *editor* HTML pur semplificando la redazione delle pagine e rendendo possibile generare delle pagine senza conoscere HTML, invogliano spesso a ragionare a livello di presentazione e non di contenuti e quindi a cambiare con molta facilità stile e tipo di carattere. Usando questi sistemi difficilmente si ha la possibilità di controllare la generazione del codice HTML, che risulta spesso incomprensibile se letto con un *editor* di testo oppure vengono generati costrutti poco efficienti. Diventa quindi difficoltoso sia riutilizzare i contenuti per mezzi diversi se non con un pesante intervento umano, sia aggiornare i contenuti, sia sopratutto poter avere diverse "viste" delle stesse informazioni, mirate a diverse tipologie di collegamento o di utenti.

In molti casi poi, nel pubblicare pagine HTML, si è cercato di utilizzare il metodo di lavoro utilizzato su altri mezzi di comunicazione. Nella carta stampata l'aspetto

grafico è importante, quindi colore, dimensione e forma dei caratteri, posizionamento nella pagina dei vari elementi, spaziature sono parametri essenziali per ottenere un buon prodotto ed inoltre il prodotto finito e stampato è uguale per tutti i lettori e quindi tutti leggono e vedono la stessa cosa. Tentare di avere una pagina HTML uguale per tutti è piuttosto difficile, vista sia la varietà di *software* ed *hardware* disponibile, che la possibilità da parte degli utenti di modificare il comportamento dei programmi utilizzati.

La possibilità che le pagine vengano visualizzate in maniera diversa a vari fruitori viene anzi considerata non un vantaggio, ma un problema da risolvere, fino a decidere una piattaforma specifica per la visualizzazione (i risultati sono i vari siti "ottimizzati" per Internet Explorer o Netscape, per una certa dimensione di schermo od un certo numero di colori o che impediscono l'accesso a chi non utilizza una determinata configurazione *software*) senza che porti a vantaggi sensibili per la fruizione del sito. In alcuni casi questo comportamento viene addirittura imposto in maniera totalmente inutile, tanto che è sufficiente impersonare il *browser* consigliato, cambiando opportunamente l'intestazione della richiesta HTTP per accedere tranquillamente al sito anziché ricevere una pagina in cui si consiglia di usare un certo *browser*.

Ci sono comunque applicazioni particolari, in cui risulta purtroppo assolutamente necessario utilizzare tecnologie disponibili solo su una piattaforma ben determinata, come ad esempio programmi Java od ActiveX per l'accesso a risorse aziendali. Normalmente queste tecnologie vengono utilizzate nelle *intranet*, ambienti controllati e ristretti che non si rivolgono al grande pubblico. I *web browser* hanno progressivamente sostituito l'utilizzo di emulatori di terminale nelle applicazioni di *data entry*: in questo caso è facile imporre una configurazione uguale per tutti gli utilizzatori.

L'uso di estensioni che per essere viste devono utilizzare *plug-in* può essere causa di problemi. La possibilità di inserire filmati ed animazioni in una pagina *web* è sicuramente utile. Bisogna però non abusare delle estensioni rendendone obbligatorio l'uso per poter accedere ai contenuti del sito[3]. In maniera simile l'utilizzo di Javascript quando non strettamente necessario può causare problemi nella navigazione di un sito, specie utilizzando *browser* testuali come Lynx o Links e rende difficile l'indicizzazione del contenuto da parte dei motori di ricerca.

Un altro problema che si riscontra, specialmente in siti di tipo commerciale, legato all'utilizzo di tecniche di comunicazione prese da altri *media*, è di fare prevalere la presentazione a scapito del contenuto. Il sito *web* aziendale viene cioè pensato ed

[3]Un caso tipico è Macromedia Flash, che ha indubbiamente una sua utilità quando ad esempio è necessario avere animazioni o si vuole utilizzare una grafica particolare, ma che se utilizzato come unico metodo per navigare in un sito risulta assai pesante e fastidioso ed inoltre impedisce l'accesso alle pagine se il *plug-in* non è disponibile per la combinazione di sistema operativo e di *browser* usato dall'utente

utilizzato come una versione elettronica di un volantino pubblicitario in quadricromia, oppure di uno spot televisivo, che deve stupire e colpire il navigatore che arriva sul sito e non come una fonte informativa per i porenziali clienti. La possibilità di fornire ulteriori informazioni viene riservata ad altri canali di comunicazione, come ad esempio i rivenditori od i *call center*. Non si vuole affermare che i siti spartani siano migliori di quelli che sono arricchiti di elementi grafici e multimediali: una buona presentazione grafica ed un corretto studio di come debbono apparire le varie informazioni sono anzi assai importanti per una buona fruibilità delle pagine *web*, ma piuttosto si vuole sottolineare come una grafica troppo invadente possa portare a grosse difficoltà nel trovare le informazioni presenti nel sito. L'utilizzo sbagliato del colore, dei tipi e delle dimensioni dei caratteri, delle immagini esplicative e del posizionamento del testo possono rendere difficoltoso e poco appagante navigare all'interno di un sito. Ma la grafica della carta stampata non si può trasferire sul *web* senza modifiche, perché dono due mezzi di comunicazione molto diversi.

Una motivazione che può spiegare questo stato di cose è che le aziende vedono il *web* come qualcosa di esterno alla propria struttura, spesso demandando la gestione solo al reparto *marketing.* Quest'ultimo fa realizzare il sito da entità esterne all'azienda ottenendo così una "scatola nera" completamente separata dal resto della realtà aziendale. Si ottengono, come risultato siti poco aggiornati e poveri di informazioni. Mantenere la propria presenza su Internet aggiornata richiede necessariamente l'utilizzo di personale che come minimo raccolga le informazioni all'interno del sistema aziendale e le consegni a chi deve intervenire sul codice HTML. Questo ovviamente ha dei costi che devono essere giustificati all'interno dell'azienda e se non esiste una certa sensibilità riguardo all'importanza di avere un sito aziendale aggiornato risulta difficile ottenere le risorse necessarie per poter fare un buon lavoro.

Per quanto riguarda invece la realtà delle amministrazioni pubbliche i problemi possono essere simili a quelli delle aziende private, ma fortunatamente è meno presente l'aspetto pubblicitario e più quello di fornire un servizio al cittadino. Ciò rende meno probabile che i siti pubblici siano "*brochure* su tubo catodico", anche se possono esserci casi in cui invece il *web* viene sentito come un "posto" in cui bisogna essere forzatamente presenti e non come un metodo utile ed economico per dare informazione agli utenti. Diventa però più importante che il sito sia fruibile ed accessibile da tutta la popolazione. Se un'azienda crea un sito poco funzionale in cui sia difficile trovare le informazioni che interessano, chi naviga semplicemente cercherà nei siti delle aziende concorrenti ed eventualmente si rivolgerà a queste ultime.

Un ente pubblico, visto che è pagato da tutti i cittadini per fornire servizi a tutti i cittadini, dovrebbe fare in modo che le informazioni presenti sul suo sito *web* possano essere accessibili da tutta la popolazione. Bisogna quindi evitare di fornire documenti in formati proprietari, particolarmente se per accedervi è necessario utilizzare programmi disponibili solo per determinate versioni di sistemi operativi o peggio

se i programmi sono a pagamento magari di costo elevato. Quando si forniscono informazioni con pagine HTML bisogna fare in modo che queste informazioni siano facilmente accessibili a tutti. Bisogna tener conto delle esigenze di chi ha disabilità percettive o di movimento e quelle di chi non ha un *computer* nuovo in grado di fare girare le ultime versioni di un certo *browser*.

Le pagine HTML devono essere scritte senza ricorrere a costruzioni strane, ma seguendo una certa metodologia in modo da rendere fruibile da tutti (e non solo dal PC del *webmaster*) i contenuti inseriti.

Il W3C ha creato un gruppo di lavoro che si occupa di scrivere delle linee guida per creare delle pagine *web* accessibili: il WAI (*Web Accessibility Initiative*) [4]. Le linee guida sono le WCAG 1.0[4] (*Web Content Access Guidelines*) i cui principi fondamentali si possono riassumere come:

1. Mettere a disposizione delle alternative al contenuto audio e video
2. Non fare affidamento solamente sui colori
3. Utilizzare *markup* e fogli di stile, e farlo correttamente
4. Chiarificare l'uso del linguaggio naturale
5. Creare tabelle che si trasformino senza problemi
6. Assicurarsi che le pagine che utilizzano nuove tecnologie si trasformino senza problemi
7. Assicurarsi che l'utente possa controllare i cambi temporizzati di contenuto
8. Assicurarsi l'accessibilità delle interfacce utente degli oggetti inseriti nella pagina
9. Progettare per l'indipendenza dalle periferiche
10. Utilizzare soluzioni *ad interim* per i sistemi più vecchi
11. Utilizzare le tecnologie e le direttive del W3C
12. Mettere a disposizione informazioni sul contesto nelle pagine complesse
13. Mettere a disposizione meccanismi di navigazione chiari
14. Fare in modo che i documenti siano chiari e semplici.

[4]Esiste una bozza della versione 2.0 di queste linee guida in corso di approvazione

Costruire un sito accessibile porta vantaggi per tutti e non solo alle persone disabili. Con la progressiva diffusione di telefoni cellulari e di *computer* palmari con un *web browser* integrato le persone che hanno sistemi per navigare che si discostano dalla classica *workstation* con uno schermo ad alta risoluzione sono sempre di più e con la diffusione dell'UMTS potrebbero diventare una percentuale significativa del *target* di un sito. Fare un sito accessibile significa renderlo disponibile non solo a chi usa una telescrivente Braille ma anche ha chi ha l'ultimo modello di GSM. Una struttura di navigazione chiara aiuta tutti i visitatori trovare più rapidamente le informazioni che interessano, aumentando l'interesse verso il sito. Allo stesso modo non usare Javascript o Flash come unico strumento di navigazione permette ai motori di ricerca di indicizzare tutto il sito, aumentando quindi il numero di persone che lo visitano.

Seguire le direttive WAI non ha solo dei vantaggi sociali, ma può portare anche a vantaggi economici e ad avere un sito di successo.

1.4 Pagine dinamiche e pagine statiche

Uno degli aspetti fondamentali del *web* è quello di essere un modo d'utilizzo delle informazioni *pu*ll, ovvero in cui il fruitore delle informazioni va attivamente alla ricerca di quello che gli interessa quando gli serve e non *push* in cui recepisce passivamente le informazioni dopo aver scelto un canale su cui ha poche possibilità di intervenire come ad esempio è il caso di radio, televisione ed in parte carta stampata. Questo significa che per attirare l'attenzione di chi utilizza la rete ed evitare che la visita ad un sito sia "mordi e fuggi" è fondamentale che le informazioni presenti siano interessanti, utili e complete.

Non è detto che chi è in grado di creare dei contenuti conosca gli aspetti tecnici collegati alla pubblicazione su *web* e viceversa i "tecnici" non sono in generale coloro che conoscono i contenuti da inserire. Esistono molti strumenti che permettono di scrivere pagine HTML senza conoscerlo e con relativa facilità ma presentano comunque delle controindicazioni:

- spesso il codice HTML generato, come già detto, è quasi illeggibile e difficilmente modificabile "a mano": è poco ottimizzato e quindi di dimensione elevata, sia per l'inserimento di elementi inutili che per la presenza di informazioni di servizio del programma generatore. In alcuni casi si arriva a generare pagine HTML con tutte le informazioni necessarie a ricreare esattamente una pagina stampata, caso ad esempio di alcuni *word processor* ed il loro "Salva come HTML";

- l'utilizzatore è libero di utilizzare qualunque stile grafico di base e di sceglierne uno diverso pagina per pagina, ed andare a modificare ogni singolo elemento in maniera totalmente incontrollata: lo si può riscontrare normalmente nelle

pagine amatoriali (ma non solo) con il rischio di avere una presentazione difficile da seguire e poco leggibile che ricorda più una poesia futurista che la pagina di un libro;

- per tentare di avere un aspetto grafico coerente la soluzione che risulta più semplice è quella, come detto prima, di utilizzare in modo eccessivo la grafica *bitmap*, a scapito del testo vero e proprio;
- risulta molto difficile inserire dati prelevati in maniera automatica da altri sistemi informatici;
- quando un sito supera una certa dimensione, e vi devono lavorare molte persone, la gestione ed il controllo del materiale generato per il sito diventa molto difficoltosa.

Se un sito è statico, cioè il *server* si limita semplicemente ad erogare le pagine HTML memorizzate sul *file system*, non è possibile risolvere questi problemi, in particolare gli ultimi due punti.

Chi scrive il contenuto, in questo caso, deve avere comunque una conoscenza sia pur minima della tecnologia sottostante. Se un sito supera una certa dimensione anche se si fa passare tutto il materiale da un esperto di HTML con una buona conoscenza della struttura del sito esistono grosse difficoltà a mantenere la coerenza delle pagine, sia dal punto di vista della presentazione che della raggiungibilità e dell'aggiornamento globale.

Per alleviare una parte dei problemi si possono utilizzare le pagine "attive", o "dinamiche": il *server* risponde ad una richiesta di una pagina eseguendo un programma i cui risultati vengono mandati al *client* richiedente. Facendo così si può ad esempio unificare le parti comuni delle pagine, oppure avere del contenuto aggiornato automaticamente prelevato da basi di dati esterne o calcolato sul momento. Allo stato attuale esistono molte tecnologie disponibili allo sviluppatore per poter scrivere *software* per generare pagine dinamiche le cui principali sono le seguenti:

Common Gateway Interface È stata storicamente la prima tecnica utilizzata per avere interazione dal *client* verso il *server*. Vengono utilizzati programmi scritti in un qualunque linguaggio, sia interpretato che compilato. I dati in ingresso ed in uscita devono seguire un protocollo ben definito e spesso esistono ulteriori limitazioni dovute a questioni di sicurezza. Normalmente i programmi risiedono in una particolare *directory* del *server*, e vengono richiamati con il loro nome e percorso nell'URL dal *client*. In diversi casi si possono utilizzare i CGI per interpretare linguaggi di *scripting* i cui sorgenti sono memorizzati all'interno del sito, configurando in modo appropriato *web server*.

L'uso dei CGI può generare problemi di scarse prestazioni del sistema sotto carico in quanto ad ogni chiamata viene generato un nuovo processo separato, che compete con gli altri processi simili per le risorse del sistema e costringe ad un numero elevato di cambi di contesto. Altro problema è che tutto l'*input* e l'*output* dev'essere gestito esplicitamente dal programma, compresa decodifica delle richieste e generazione completa di HTML, ma comunque è possibile in molti linguaggi di programmazione utilizzare delle librerie apposite.

Server scripting Si utilizzano dei linguaggi di programmazione progettati per la generazione di pagine HTML, che hanno caratteristiche che ne rendono decisamente più semplice l'utilizzo, come ad esempio la decodifica automatica dei dati trasmessi dall'utente oppure la possibilità di inserire senza problemi spezzoni di HTML da inviare all'utente senza elaborazione.

I programmi scritti con questi linguaggi possono essere utilizzati come CGI, ma normalmente negli ambienti di produzione sono utilizzati come moduli interni al *web server* ottenendo così migliori prestazioni: i processi generati sono in numero molto inferiore rispetto ciò che accade utilizzando i CGI perché si evitano troppi cambi di contesto. Si possono comunque utilizzare in questo modo alcuni linguaggi non pensati per la generazione di HTML, come ad esempio PERL, Python o TCL, con gli opportuni moduli inseriti nel *web server*, se questi sono stati adattati per questo tipo di funzionamento. I linguaggi di *server scripting* più noti sono:

ASP È il linguaggio di riferimento per Microsoft ed il suo Internet Information Server. Le pagine possono essere scritte sia VBScript, in cui il linguaggio è una parte di Microsoft Visual Basic opportunamente adattata per le esigenze del *web*, che in JScript, in cui il linguaggio è simile a server-side Javascript. Si possono integrare con le pagine ASP oggetti COM e la connettività con basi di dati. Con gli oggetti COM diventa possibile avere degli oggetti molto simili ai moduli per il *server scripting* od utilizzare versioni compilate di pagine ASP per un miglioramento delle prestazioni (con alcune limitazioni sul tipo delle pagine).

PHP È un linguaggio di programmazione molto adatto per applicazioni *web* che può esser inserito dentro parti di HTML. Può essere utilizzato anche per *script* da riga di comando od applicazioni grafiche lato *client*. Sono disponibili versioni per i più diffusi sistemi operativi e *web server*, sia come modulo che come interprete CGI. Si può utilizzare PHP come linguaggio procedurale oppure ad oggetti, od un ibrido fra i due. Oltre a poter generare HTML, XML od XHTML, si possono generare al volo immagini, documenti PDF o filmati Flash. Ci si può interfacciare ad un grande numero di *database* ed utilizzare vari protocolli di rete, come

LDAP, IMAP, SNMP, NNTP, POP3, HTTP oppure interagire a livello TCP.

J2EE (*Java 2 Platform, Enterprise Edition*) Sono una serie di tecnologie sviluppate da Sun, basate su Java, ovvero i componenti JavaBean, API per le *servlet* Java e le Java Server Page. I JavaBean permetto di utilizzare pacchetti che implementano funzionalità verso l'interno del sistema informativo aziendale, mentre le *servlet* si possono vedere come classi Java che implementano specifiche funzionalità lato *web*, permettendo di avere un'infrastruttura che estende le funzionalità del *web server*.

Le Java Server Page permettono di creare *script* in Java inseriti all'interno delle pagine HTML. Rispetto ad un linguaggio di *scripting* la differenza maggiore è che il codice viene compilato nel *bytecode* di Java una sola volta e non interpretato ogni volta che viene richiesta una pagina. Inoltre la modularità del sistema permette una forte integrazione con altri sistemi informativi anche utilizzando architetture complesse.

La creazione di *script* per un particolare sito, sia pur semplificando la gestibilità per quanto riguarda la parte tecnica, introduce ulteriori strati di complessità. Per chi è digiuno di HTML o ne ha una conoscenza limitata ciò significa avere dei problemi perché, come già detto, anziché dover semplicemente produrre un ipertesto, si deve invece scrivere un programma per elaboratore con le relative difficoltà di ordine concettuale: non si ha più a che fare con un *word processor* evoluto[5] (con concetti come parole, capitoli e sottolineature) ma bisogna si ha a che fare con variabili, cicli, salti condizionali e funzioni.

Anche per chi conosce bene i linguaggi di *scripting* dover mettere mano ad un sito dinamico fatto da un altro spesso può rivelarsi difficoltoso, in particolare quando l'uso delle parti dinamiche non deriva da un progetto complessivo, ma da una scrittura estemporanea per risolvere piccole parti. Poiché spesso questi *script* non sono né documentati né tanto meno commentati, diventa difficile capire che cosa fanno, e quindi la modifica ed il riutilizzo degli *script* e di codice HTML presente nelle pagine diventa un'attività che porta via molto tempo.

Sembrerebbe quasi che per risolvere il problema delle pagine statiche si siano introdotti altri problemi più gravi ed ancor più difficili da risolvere. In realtà in molti casi l'utilizzo delle pagine dinamiche risolve i problemi che si hanno con l'uso

[5]Esistono sistemi di scrittura che sono programmabili, come ad esempio Microsoft Word, Corel WordPerfect od OpenOffice, ma se non è necessario scrivere delle macro, si possono utilizzare questi programmi senza grossi problemi. TEX, invece è un linguaggio di programmazione a tutti gli effetti, ma se ci si limita ad utilizzare i pacchetti già esistenti, si comporta come un sistema di scrittura con l'uso di *mark up* se non come *back end* di sistemi di scrittura grafici come Scientific Word o LYX.

delle pagine statiche: bisogna aver però presente che non è detto che siano sempre la soluzione ideale in tutti i casi e che è necessario sapere che non si ha più a che fare con un "documento".

1.5 La vera potenza delle pagine dinamiche

Avere in mano la potenza di un linguaggio di programmazione vero e proprio permette però, visto in un'altra prospettiva, di risolvere in maniera elegante il problema della complessità di HTML e della gestione di un sito, introducendo un maggiore livello d'astrazione.

Si possono scrivere delle procedure che permettono di inserire del testo in maniera semplice (ad esempio utilizzando *form* HTML in cui l'utilizzatore va a scrivere quello che dovrà apparire sul *web*) e che automaticamente vanno a memorizzare il testo all'interno di una base di dati. Altre procedure vanno a recuperare dalla base di dati i testi necessari a comporre una pagina al momento della richiesta HTTP. In altri termini il contenuto diventa una serie di dati forniti da un programma che genererà un *output* opportuno.

Per evitare di dover riscrivere nuovo codice per siti diversi che richiedono prestazioni simili, sono nate delle librerie che implementano funzionalità standard e che possono essere inserite nelle pagine dinamiche con eventualmente piccole personalizzazioni. Da qui si è arrivati ad interi pacchetti applicativi preconfezionati che permettono di avere tutta l'infrastruttura necessaria per generare un sito *web* dinamico, semplicemente da personalizzare secondo le proprie esigenze. Utilizzando questi applicativi è possibile, con relativa facilità, ottenere risultati molto accattivanti dal punto di vista grafico e si riescono a gestire in maniera coerente anche siti di una certa dimensione con caratteristiche avanzate.

Grazie alla possibilità di utilizzare componenti già fatti è semplice aggiungere oggetti come *guestbook*, calendari o gallerie di immagini. Lo si può fare senza dover intervenire sul codice, ma semplicemente attivando le librerie già disponibili. Molti degli applicativi permettono di personalizzare l'aspetto grafico della pagina sia permettendo diverse "viste" degli stessi dati che di personalizzare le pagine in modo diverso per ogni utente.

Ci si è quindi avvicinati a sistemi in cui risulta centrale il contenuto e non la presentazione, o meglio in cui questi due aspetti sono gestibili in maniera separata ed eventualmente riciclabili in contesti diversi. I *Content Management System* infatti sono applicazioni che hanno come aspetto centrale appunto la gestione dei contenuti: alcuni pacchetti nati nel mondo *web* hanno espanso le loro funzionalità fino a poter essere considerati dei CMS. Delle caratteristiche dei CMS si discuterà più approfonditamente nel prossimo capitolo.

Questa presentazione è stata necessariamente sintetica e molti aspetti che avrebbero meritato comunque attenzione sono stati appena accennati, ma si è ritenuto importante fare notare come la nascita del WWW abbia modificato profondamente il mondo dell'informatica, sia dal punto di vista tecnico che dal punto di vista sociale, creando delle problematiche nuove dovute alla interazione tra questi due aspetti. Problematiche che si possono risolvere adottando gli opportuni strumenti, permettendo così ad un sempre maggior numero di persone di poter accedere e contemporaneamente creare informazione.

Capitolo 2

Content Management

Nella costruzione di un sito *web* o nella gestione di documentazione a livello aziendale si possono presentare diversi casi in cui può essere interessante valutare l'utilizzo di una classe di applicativi indicati come CMS, ovvero *Content Management System*. In questa parte si andrà a vedere quali siano le definizioni di CMS e di quali siano le caratteristiche tipiche di questi sistemi, soprattutto nell'ambito del *web* e si analizzeranno brevemente le caratteristiche principali di alcuni prodotti CMS liberi.

I sistemi di *content management* sono una presenza relativamente nuova nel mondo dell'informatica e quindi la definizione di cosa siano non è così semplice, rispetto ad esempio a definire cosa sia una base di dati relazionale. Esistono sistemi di CM il cui costo supera le centinaia di migliaia di euro, e sistemi da poche centinaia di euro, oltre naturalmente a diversi prodotti liberi, con aree d'applicazione e funzionalità molto diverse fra di loro. Molti sistemi costruiti "in casa" per la gestione di siti *web* hanno le caratteristiche e le funzioni di un CMS, anche se le possibilità di personalizzazione o di riutilizzo in altre situazioni possono risultare piuttosto limitate.

Risulta quindi piuttosto difficile confrontare i vari prodotti disponibili, sia per una loro catalogazione, sia nel caso si debba valutare se l'utilizzo di un prodotto presente sul mercato soddisfi in adeguatamente le proprie esigenze o se al contrario risulti più efficace sviluppare una soluzione *ad hoc* per la propria situazione organizzativa.

Bisogna tener presente che anche i sistemi commerciali più complessi tendono a gestire al meglio un sottoinsieme delle funzioni di un CMS ed a essere più adatto ad essere utilizzato in alcune applicazioni rispetto ad altre. Non esistendo ancora una definizione univoca di CM ogni azienda produttrice può darne la descrizione che meglio si adatta alle funzionalità del proprio prodotto. Perché un *content management system* sia effettivamente utile per un'organizzazione spesso è necessario integrare la gestione del contenuto con il sistema informativo preesistente e la gestione interna dell'organizzazione, o comunque definire bene le interazioni fra le entità già presenti ed il sistema di gestione del contenuto.

2.1 Definizione di Content Management

Il *content management* può essere visto come l'attività di recuperare, gestire e pubblicare il contenuto su uno o più mezzi di comunicazione. Il recupero dell'informazione può avvenire fondamentalmente in due modi: la si può creare *ex novo* oppure la si può acquisire da fonti esterne. Una volta fatto questo, il contenuto inserito viene convertito in un formato interno al sistema (tipicamente codificato in XML) ed infine il contenuto viene aggregato, sempre all'interno del sistema, correggendolo o suddividendolo in componenti ed aggiungendo i metadati appropriati.

Il contenuto viene gestito inserendolo normalmente in una base di dati, in un *file system* o con soluzioni ibride, assieme ai metadati associati ad esso. È importante avere una corretta gestione ed inserimento dei metadati e porre attenzione che i metadati inseriti descrivano correttamente i dati ad essi associati.

Nella pubblicazione il contenuto presente nella base di dati viene convertito nel formato appropriato per essere diffuso, ad esempio mediante siti *web*, pubblicazioni cartacee o bollettini di posta elettronica. Le pubblicazioni sono costruite mettendo assieme gli opportuni componenti, informazioni standard di contorno e ausili alla navigazione, per esempio i *link* nelle pagine HTTP e gli indici sulle pubblicazioni stampate. Dal punto di vista dell'amministrazione di un'azienda lo sfruttamento di un sistema di gestione del contenuto porta con sé anche conseguenze che possono arrivare ad espandersi a tutta l'organizzazione aziendale.

> Se il CM fosse solamente un processo per facilitare la creazione di grandi siti *web*, allora utilizzarlo per organizzazioni con grandi siti *web* ovviamente potrebbe valere lo sforzo necessario all'implementazione ed alla gestione di questi applicativi. Per organizzazioni con più pubblicazioni di grandi dimensioni, anche se si tratta semplicemente di siti *web* multipli, il *content management* può diventare una necessità. Esiste comunque una ragione più profonda sul motivo per cui il *content management* è importante: non si tratta solo dei tipi comuni d'informazione, o più precisamente di contenuto che è necessario recuperare, gestire e consegnare. È gestione aziendale. Quando le organizzazioni iniziano a gestire le proprie attività elettronicamente, vanno incontro allo stesso tipo di effetti che hanno quando iniziano a distribuire le informazioni elettronicamente, facendo sorgere alcuni problemi da risolvere: come suddividere le varie attività in parti "distribuibili" elettronicamente; come essere sicuri quali parti siano disponibili e se queste siano quelle giuste per il personale dell'azienda, per i fornitori ed i clienti; come essere sicuri che le parti giuste arrivino alla persona giusta al momento giusto. Se il CM è il processo di recuperare, gestire e pubblicare il contenuto, allora l'*e–business* è il processo di recuperare, gestire e pubblicare parte dell'attività aziendale.

Naturalmente molta di questa gestione aziendale implica la pubblicazione d'informazioni. Quindi, in generale, è possibile osservare come il *content management* può avere sotto un po' di *e–business* [5].

2.1.1 Contenuto

Che cos'è il contenuto che questi sistemi gestiscono? Il contenuto si può definire come l'informazione in un formato utilizzabile presentata o pubblicata per un certo scopo, e che viene posto in un contesto specifico dalla presentazione. Il contenuto è un concetto differente da quello di "dati", che sono informazione privata del contesto e ridotta in un formato preciso in maniera tale da potere essere elaborata in maniera facile.

Il contenuto è normalmente composto da più informazioni messe insieme in un ordine logico. Un giornale per esempio è composto di articoli, pubblicità ed indice. A sua volta un articolo di giornale è composto dal testo dell'articolo, dalle immagini, dal titolo e dalle didascalie. Il testo dell'articolo è composto da paragrafi, a loro volta composti da frasi, a loro volta composte da un insiemi di caratteri. Questo insieme di caratteri è un esempio di dato: si tratta di una stringa, di su cui è possibile effettuare operazioni in maniera univoca, come ad esempio calcolarne la lunghezza, convertire in maiuscolo o minuscolo, unire due stringhe insieme, cercare se esiste una sotto stringa all' interno, eccetera. Il risultato di queste funzioni, privato del contesto non è molto utile di per sé e va necessariamente reinserito in un nuovo contesto appropriato per poter dare significato ai dati ottenuti.

Dato	Contenuto
42	*La linea di autobus per andare da Piazza Sabotino a Corso Spezia è la 42.*

Figura 2.1. Differenza tra dato e contenuto

2.1.2 Metadati

Assieme ai contenuti, in un CMS vengono gestiti i metadati, cioè i dati che descrivono il contenuto. Esempi tipici di metadati relativi a un documento sono l'autore, la data di inserimento, le parole chiave o un riassunto. Un filmato può avere come metadati la durata, dove iniziano le varie scene con una loro descrizione, chi appare nel filmato o la provenienza del materiale. I metadati non vanno confusi con le informazioni di servizio associate ad un documento, come ad esempio il nome del *file* in cui è memorizzato o lo stato di lavorazione all'interno di un *workflow* per la pubblicazione: i metadati sono legati al contenuto e non variano se il documento viene estratto da un sistema informativo per essere inserito in un altro, mentre i dati di servizio possono variare da un sistema ad un altro fino anche ad esistere in uno solo dei sistemi.

Per una corretta catalogazione del contenuto è assolutamente necessario avere un insieme di metadati utili e che descrivano correttamente il contenuto ad essi associato. Avere dei metadati corretti, nel caso ad esempio di immagini, suoni o filmati, è spesso l'unico modo per poter effettuare facilmente delle ricerche. Anche nel caso di ricerche testuali l'utilizzo dei metadati permette di ottenere risultati più significativi rispetto all'utilizzo di catalogazioni automatiche.

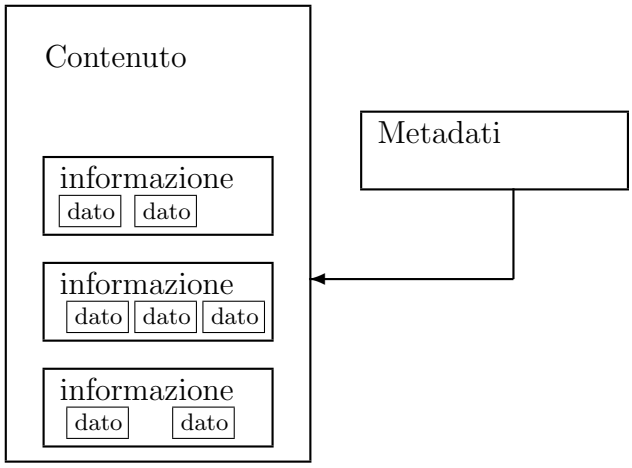


Figura 2.2. Relazione tra dati, informazione, contenuto e metadati

2.1.3 Obiettivi e componenti del contenuto

Un sistema di gestione del contenuto ha un obiettivo, cioè il tipo degli argomenti che vengono trattati dai documenti pubblicati al suo interno. Se il contenuto serve a generare una sola pubblicazione, obiettivo del contenuto e della pubblicazione

coincidono. La definizione degli obiettivi del contenuto deve essere breve e precisa: una definizione troppo ampia rende difficile organizzare e catalogare i documenti inseriti.

Una volta stabiliti gli obiettivi del contenuto si può procedere ad una suddivisione per argomenti e tipologia dei vari tipi di contenuto. Ad esempio se l'obiettivo di un sistema di gestione del contenuto è la gestione dei *data sheet* e delle note applicative di componenti elettronici, la prima suddivisione può essere tra note applicative e *data sheet*. I *data sheet* si possono suddividere per categorie: componenti discreti, circuiti integrati digitali, circuiti integrati analogici. I componenti discreti si possono suddividere tra componenti di segnale e componenti di potenza e così via. Ogni categoria può avere necessità differenti, che ad ogni livello possono esser meglio specificate e che portano ad avere contenuti organizzati in modo diverso.

Il contenuto si può anche suddividere per componenti. I componenti sono parti del contenuto che sono ancora contenuto, cioè hanno ancora un contesto ed una struttura e possono essere elaborati indipendentemente dalla presenza di altri contenuti. Se ad esempio si esamina un *data sheet* di un amplificatore operazionale, lo si può suddividere in descrizione, piedinatura, schema elettrico interno, caratteristiche limite, caratteristiche elettriche, grafici e applicazioni tipiche.

In un sistema di gestione del contenuto, le componenti vengono inserite ed elaborate in maniera indipendente: al momento della pubblicazione i vari componenti vengono messi assieme con una certa struttura a formare un documento.

2.1.4 Obiettivi di una pubblicazione

Chi legge o vede il contenuto non è interessato alla catalogazione dei componenti del contenuto, né come i componenti sono recuperati e gestiti. I lettori sono interessati semplicemente a ricevere l'informazione che interessa loro in una forma coerente e nelle forme in cui sono abituati come ad esempio siti *web*, libri, riviste o trasmissioni radiofoniche. Pubblicare è rendere disponibile l'informazione precedentemente elaborata con un formato ed una struttura precisa, con un formato ed una struttura definiti, destinata ad un certo tipo di pubblico.

Il formato di pubblicazione si può suddividere in due parti. Una è la codifica dell'informazione e l'altra è la riproduzione visiva dell'informazione. Per l'editoria tradizionale è sufficiente che casa editrice e tipografia si mettano d'accordo per un formato comune di interscambio, che può essere sia un formato elettronico che una descrizione manuale di come il materiale debba essere stampato. Per il *web* il formato di interscambio è HTML. Nel caso del *content management* i problemi sul formato di pubblicazione esistono quando si hanno più mezzi di comunicazione diversi, come ad esempio il WWW ed una pubblicazione cartacea, e risulta importante avere un approccio il più possibile neutro nella definizione del formato interno per non

sbilanciare troppo il sistema verso una sola forma di pubblicazione rendendo così difficile la gestione degli altri formati.

Tutte le pubblicazioni, anche quelle più semplici, hanno una struttura implicita che chi legge usa per comprendere meglio le informazioni presentate. All'aumentare della complessità di una pubblicazione la complessità struttura aumenta di pari passo, e diventa sempre più necessario definire con precisione la struttura nel processo di produzione, andando così ad utilizzare sistemi sempre più complessi per la sua gestione.

Il pubblico a cui viene destinata una pubblicazione è importante per capire come deve essere presentato e con che struttura e soprattutto quale contenuto deve essere pubblicato: il manuale d'uso di un'apparecchiatura e un volantino pubblicitario sono molto diversi fra loro, così come le informazioni utili per un sito internet sono diverse da quelle visibili all'interno di una intranet aziendale. Il contenuto stesso è influenzato dall'*audience* attesa: un articolo di fondo per un quotidiano sarà molto diverso da un capitolo di un saggio che tratta degli stessi argomenti, così come gli articoli per un settimanale di *gossip* saranno decisamente diversi da quelli di un settimanale economico.

Quando da uno stesso contenuto si vogliono ottenere più pubblicazioni con struttura diversa possono sorgere problemi. Come già detto, diventa necessario avere una struttura diversa per ogni pubblicazione ed una per il contenuto al fine di non bloccare quest'ultimo su un tipo solo di pubblicazione a scapito degli altri. La rappresentazione quindi deve essere il più possibile neutrale per poter derivare facilmente le altre strutture: perciò tipicamente si utilizzano codifiche basate su XML e la suddivisione del contenuto in blocchi logici.

2.1.5 Architettura di un CMS

Un sistema di gestione del contenuto può essere diviso in alcune aree funzionali: per l'acquisizione del contenuto, per la memorizzazione del contenuto all'interno del sistema, per la gestione del flusso di lavoro ed infine per la pubblicazione.

2.1.5.1 Recupero dei contenuti

Questa parte del sistema si occupa di tutto quanto è necessario per rendere disponibile il contenuto per la pubblicazione, trasformando le varie informazioni in un gruppo ben strutturato di contenuti. I processi principali che vengono effettuati nella fase di recupero sono:

Creazione in questo caso il contenuto viene creato da zero ed inserito direttamente nel sistema di gestione, possibilmente utilizzando sistemi che permettano agli autori di inserire comodamente e correttamente il contenuto ed i metadati relativi all'interno del sistema.

Acquisizione il contenuto viene prelevato automaticamente da una fonte esterna, come ad esempio una base di dati e trasformato in componenti utilizzabili in un CMS.

Conversione il contenuto creato od acquisito viene convertito, eliminando eventuali parti inutili e modificandone la struttura per poter essere meglio gestito successivamente.

Aggregazione il contenuto preesistente viene suddiviso, modificato ed adattato in modo da essere più facilmente gestibile con il sistema di metadati in uso.

L'importanza del sistema di recupero del contenuto è fondamentale perché un CMS funzioni. Se si utilizzano gli strumenti sbagliati per creare ed acquisire contenuto, si rischia di avere informazione poco riutilizzabile per la pubblicazione: ad esempio se in un testo non si distingue tra corpo, intestazioni di pagina e note diventa impossibile andare a modificare il formato delle pagine od anche semplicemente il carattere utilizzato senza pesanti interventi manuali.

2.1.5.2 Il sistema di gestione

Il sistema di gestione si occupa di memorizzare e recuperare tutti i dati inseriti all'interno del CMS, con più precisione le funzioni si possono suddividere in:

Memorizzazione il contenuto può essere salvato in una o più basi di dati, oppure può essere salvato sul *file system* od avere soluzioni ibride: ad esempio spesso i dati multimediali vengono salvati su disco di sistemi remoti e nella base di dati vengono inseriti puntatori che permettono di andare a recuperare il contenuto. Il sistema di memorizzazione deve essere in grado di gestire contenuto testuale, dati binari (per i contenuti audio/video) e soprattutto i metadati associati al contenuto stesso.

Selezione una volta salvato, il contenuto deve essere reso disponibile per la pubblicazione e l'utilizzo: indicando in una richiesta i metadati adatti all'identificazione del contenuto il sistema deve restituire il contenuto che si può associare ad essi.

Sicurezza/amministrazione l'accesso al contenuto deve essere controllato in maniera tale da impedire una modifica od una lettura indiscriminata nonché per poter sapere chi e quando ha effettuato determinate operazioni nel sistema. Bisogna anche prevedere sistemi di *back-up* e ridondanza per evitare la perdita di dati in caso di guasti del sistema.

Connessioni risulta utile poter collegare altri sistemi informativi per lo scambio di dati in maniera tale da poter distribuire il carico di lavoro oppure più semplicemente per poter condividere dati con sistemi eterogenei.

2.1.5.3 Flusso di lavoro

Il flusso di lavoro gestisce in maniera efficiente il processo di recupero, memorizzazione e pubblicazione del contenuto, seguendo tempistiche ed azioni ben definite. È utile che la gestione del flusso di lavoro comprenda tutti i passi dall'inizio alla fine del processo, il personale interessato, i processi, gli strumenti e le funzioni utilizzati, la scansione temporale ed il flusso informativo con il controllo delle transizioni e degli stati possibili.

La parte di gestione del flusso di lavoro può avere la possibilità di compiere automaticamente azioni sul contenuto, sia internamente al sistema, sia utilizzando sistemi informatici esterni, oltre a richiedere l'intervento umano notificando le persone interessate dei compiti loro assegnati.

2.1.5.4 Pubblicazione

Il sistema di pubblicazione recupera le parti di contenuto e le altre risorse necessarie per poter creare automaticamente le pubblicazioni, ad esempio pagine HTML, oppure documenti stampati su carta.

La parte più importante di un sistema di pubblicazione è la gestione dei *template*(o modelli), ovvero documenti che indicano come creare la pubblicazione a partire dal contenuto disponibile nel sistema di gestione. Questi *template* sono dei semplici programmi che implementano la logica necessaria a costruire le pubblicazioni e includono tre categorie di oggetti. La prima categoria è quella degli elementi statici cioè testi, grafica e contenuti multimediali che vengono inseriti direttamente nella pubblicazione senza altre modifiche. La seconda categoria è quella delle chiamate ai servizi di pubblicazione, che vanno a recuperare ed a rimpaginare parti di contenuto prelevate dal sistema di gestione e si occupano di personalizzazioni, conversioni di contenuto e di costruire indici od alberi di navigazione. La terza categoria è quella delle chiamate esterne, con cui è possibile richiamare automaticamente funzioni presenti su altri sistemi informatici esterni per la post produzione o l'inserimento in altri sistemi aziendali.

La pubblicazione può avvenire in maniera dinamica, come nel caso delle pagine HTML, in cui la richiesta attraverso un *browser* fa attivare il sistema di pubblicazione, che genera la pagina sul momento. In alternativa, come nelle pubblicazioni cartacee. la pubblicazione può essere richiesta manualmente oppure mediante una procedura automatica. In questo modo si creano i dati necessari alla stampa od alla pubblicazione elettronica.

2.2 Altri sistemi

Oltre ai CMS, esistono applicazioni che seppure presentano similitudini e caratteristiche comuni, se ne differenziano perché alcune funzionalità non sono presenti, mentre ne esistono altre che si adattano meglio ad alcune attività specifiche.

Le differenze comunque non sono così nette e questa suddivisione è abbastanza generica. È possibile che uno di questi sistemi possa essere espanso fino ad avere tutte le caratteristiche di un CMS: ad esempio è possibile che un *asset management system* sia in grado di aggiornare automaticamente i contenuti di un sito *web* partendo da un *template* e popolandolo con i contenuti multimediali e le descrizioni ottenute dai metadati ad essi associati.

2.2.1 Document Management System

Una prima categoria è quella dei cosiddetti DMS, ovvero *document management system*, la cui principale differenza rispetto ai CMS è quella di gestire non contenuto, ma documenti. Questi sistemi sono orientati a gestire in maniera coerente i documenti presenti all'interno di una organizzazione, senza entrare nel merito della loro pubblicazione, anzi è possibile che i documenti siano memorizzati con un formato già pronto per la pubblicazione, come ad esempio quello di un *word processor*.

Un DMS gestisce un archivio di documenti con un'ampia gestione dei metadati relativi a ciò che è contenuto al suo interno, oltre al flusso di lavoro collegato alla loro elaborazione. Viene tenuta traccia della storia dei vari documenti e delle relazioni presenti tra essi ed insieme viene messa a disposizione una serie di strumenti di ricerca e di navigazione per rendere facilmente recuperabili i documenti. I *document management system* si adattano bene a documenti fortemente strutturati, come ad esempio specifiche tecniche o nell'ambito legale.

2.2.2 Asset Management System

Di questa categoria fanno parte applicativi piuttosto simili ai *document management system*, salvo il fatto che i documenti che vengono gestiti sono normalmente *file* binari contenenti audio, video ed immagini.

I metadati vengono utilizzati per poter cercare e catalogare facilmente i contenuti, che in alcuni casi possono essere non memorizzati all'interno di un sistema informatico ma corrispondere ad oggetti esterni ad esso: per esempio un AMS che gestisce i contenuti di un'emittente televisiva può tenere traccia di dove siano le videoregistrazioni analogiche del materiale da montare e metterle in relazione al montato presente all'interno del sistema in forma numerica. Inoltre un AMS può

prevedere nel flusso di lavoro dei sistemi automatizzati per la conversione di formato, ad esempio per l'acquisizione da sorgente analogica oppure transcodificare un filmato da MPEG2 a Realvideo per la pubblicazione su *web*.

2.2.3 Web Content Management

Di questa categoria fanno parte sistemi di CMS focalizzati principalmente per la pubblicazione su *web*, anziché su altri mezzi. In particolare i WCMS sistemi permettono di gestire pagine HTML per internet ed intranet, oltre a prevedere la gestione semplificata di applicazioni di *e–commerce* e l'automazione delle *syndacation*[1]. Di questa categoria fanno parte anche sistemi più semplici che permettono di gestire in maniera più facile i siti, avendo come unità di contenuto la pagina HTML.

2.2.4 Learning Content Management System

Di questa categoria fanno parte quei sistemi di CMS e WCM in grado di strutturare il contenuto *online,* a partire da blocchi di contenuto che vanno combinati assieme per ottenere un evento didattico (permettendo il riutilizzo e la portabilità del contenuto) e la possibilità di controllare e verificare l'utilizzo del sistema attraverso i risultati ottenuti, l'avanzamento ed il completamento delle attività e di altri indicatori statistici.

2.2.5 Revision Control

In questa categoria si collocano una serie di sistemi che permettono la gestione di documenti di testo puro, in particolare dei *file* che compongono il codice sorgente di un programma.

Le funzioni che sono normalmente presenti sono quelle di permettere un *versioning* dei listati, cioè di poter avere presenti diverse versioni e di poter andare a scegliere quale andare a modificare o compilare in un dato momento, di indicare che un *file* sta venendo modificato, di poter gestire più sottoversioni diverse e di memorizzare chi ha fatto le modifiche ed eventuali annotazioni relative ai sorgenti, nonché ricavare in maniera automatica le differenze tra una versione e l'altra di un listato.

Il loro utilizzo principale è naturalmente nell'ambito della scrittura di programmi, ma è anche possibile utilizzarli per altri tipi di dato in forma testuale, come ad esempio documenti HTML ed XML.

[1] Trasferimento automatico di informazioni presenti in altri siti *web*: un esempio tipico è la *home page* di Slashdot, `http://slashdot.org`, che in una colonna ha i titoli dei nuovi articoli presenti sulle *home* di altri siti.

2.3 Sistemi di Content Management Open Source

Dopo aver visto i tratti salienti che contraddistinguono un sistema di gestione del contenuto, risulta utile andare a vedere i principali applicativi presenti sul mercato. Per quanto riguarda i sistemi liberi ed *open source*, si è fatta una rapida descrizione delle loro caratteristiche qui di seguito.

Nella tabella 2.1 sono invece indicati i nomi ed il costo indicativo dei principali prodotti commerciali [8]. Il prezzo indicato è quello relativo alle licenze dei programmi: va tenuto presente che per una installazione funzionante va aggiunto il costo dell'*hardware*, della formazione del personale e dei costi relativi alla personalizzazione dei sistemi per una applicazione specifica, cose che ovviamente vanno tenute in considerazione anche nei sistemi liberi ed *open source.*

In questo tipo di applicazioni le caratteristiche intrinseche del *software* libero possono risultare più utili rispetto ad altri ambiti. Poiché per poter essere utilizzati con successo, è necessario comunque effettuare delle personalizzazioni, la disponibilità dei codici sorgenti permette di intervenire con maggiore efficacia. Inoltre non essendo legati ad un produttore non si rischia di veder svanire il supporto per il prodotto acquistato se chi l'ha scritto va fuori dal mercato, oppure che le versioni vecchie dell'applicazione (su cui magari si è intervenuti pesantemente con personalizzazioni) non vengano più supportate dal produttore. Infine l'integrazione di sistemi costruiti in casa con le soluzioni libere "preconfezionate" è decisamente più facile, potendo eventualmente recuperare da un programma libero alcune parti ed inserendole nel proprio sistema e viceversa.

2.3.1 Zope

Zope, più che un CMS, si può considerare un *application server*, cioè un sistema integrato per la gestione di applicazioni *web* che fornisce un server HTTP, una interfaccia di controllo via *web*, una base di dati ad oggetti, la possibilità di collegarsi a fonti di dati esterne ed il supporto a linguaggi di *scripting* in un solo prodotto. Grazie alla sua struttura modulare, risulta possibile ampliare facilmente le sue funzionalità con caratteristiche tipiche dei portali o dei CMS oppure costruire in maniera rapida applicazioni di *e-commerce.*

Per Zope esiste la possibilità di avere supporto commerciale sia da Zope Corporation [10], la società che gestisce e sviluppa Zope, che da altre imprese commerciali che possono costruire applicazioni basate su Zope in *outsourcing.* Se invece si intende sviluppare una applicazione personalmente, oltre a prodotti editoriali [7], esiste su internet molta documentazione [9], nonché *mailing list* e *forum* di utilizzatori e di sviluppatori, come tutti i prodotti *open source* di successo.

La differenza principale rispetto alle altre soluzioni è che si tratta di un pacchetto totalmente autonomo. Dopo averlo installato, per il suo funzionamento non è

Tabella 2.1. Principali CMS commerciali

Piattaforme *Enterprise* – Pacchetti pensati per l'utilizzo a livello di grandi aziende e gruppi globali. I prezzi per una licenza a livello base superano i 250.000€.	
Vignette — V7 Content Management Suite	www.vignette.com
Documentum — Documentum 5	www.documentum.com
divine — (OpenMarket) Content Server	www.divine.com/servlet/ContentServer?pagename=home
Interwoven — TeamSite 5.5	www.interwoven.com
Livello superiore – Pacchetti pensati per grandi unità produttive. I prezzi per una licenza a livello base per l'implementazione sono di circa 125.000 ÷ 175.000€.	
Stellent — Stellent Content Management Suite	www.stellent.com
Percussion — Rhythmyx 4.5	www.percussion.com
Microsoft — Content Management Server	www.microsoft.com/cmserver/default.aspx
FatWire — UpdateEngine6	www.fatwire.com
FileNET — WCM (ex eGrail)	www.filenet.com
Mediasurface — Mediasurface 4.5	www.mediasurface.com
Gauss — Interprise VIP	www.gaussinterprise.com
Day — Communiqué	www.day.com/en.html
Tridion — R5	www.tridion.com/com/index.asp
Mercato medio – Pacchetti rivolti ad un mercato intermedio. I costi di licenza oscillano tra i 40.000 ÷ 100.000€ e normalmente impongono uno sforzo d'integrazione più modesto.	
Merant — Collage	www.merant.com/Products/WCM/wcmsolutions.asp
RedDot Solutions — RedDot	www.reddotsolutions.com
IXOS — Obtree - C4	www.obtree.com
Ingeniux — Ingeniux Content Management System	www.ingeniux.com
PaperThin — CommonSpot Content Server	www.paperthin.com
Roxen — Roxen CMS	www.roxen.com
Red Bridge Interactive — Dynabase / Engenda	www.rbii.com
Economici – Prodotti mirati a richieste piuttosto semplici il cui costo è fra i 1.000 ed i 10.000€.	
Ektron — eMPower3.0 e CMS200	www.ektron.com
UserLand — Manila	www.userland.com
GlobalSCAPE — PureCMS	www.globalscape.com/purecms
Orientati alla consegna – Questi pacchetti, tipicamente portali ed *application server*, si focalizzano sulla parte della consegna, alla fine del ciclo produttivo di un CMS: generazione pagine, personalizzazione, *caching*, ecc...	
BEA — WebLogic E-Business Platform	www.bea.com/framework.jsp?CNT=homepage_main.jsp&FP=/content
ATG — Dynamo e-Business Platform	www.atg.com/en/index.jhtml
Plumtree — Plumtree	www.plumtree.com
Oracle — Oracle9iAS	www.oracle.com/ip/deploy/ias/index.html
IBM — Websphere Portal	www-3.ibm.com/software/info1/websphere/index.jsp?tab=products/portal

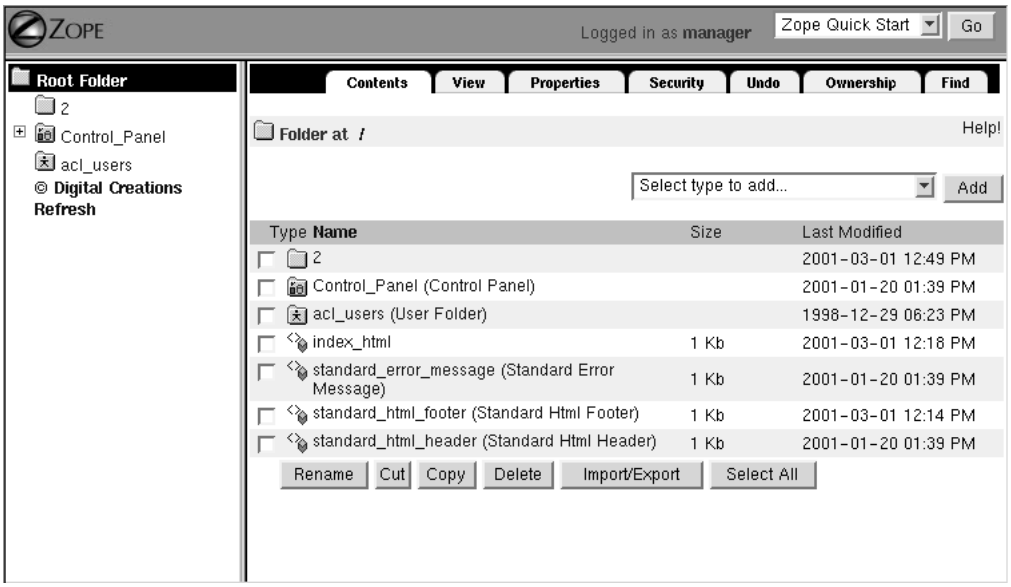


Figura 2.3. Schermata di amministrazione di Zope

necessario avere altri programmi di supporto per la gestione HTTP o per la memorizzazione dei dati interni. Questo vuole dire che può essere portato su qualunque sistema operativo che abbia un compilatore C ed un interprete Python funzionante, ma che nel caso si stia utilizzando un'architettura diversa per la gestione dei servizi *web* come per esempio J2EE od IIS/ASP non si possono riutilizzare le parti già esistenti all'interno della struttura di Zope.

Per quanto riguarda la sicurezza, grazie ad un sistema di *access list* è possibile controllare in maniera molto flessibile sia chi può andare a modificare parti del sito sia chi può accedervi. È prevista anche la possibilità di creare *virtual host* e gerarchie di utenti: ad esempio si possono gestire in maniera indipendente domini e sotto-domini diversi sulla stessa macchina.

Mediante il sistema ZEO (*Zope Enterprise Objects*) è possibile avere un sistema costituto da più macchine che fanno girare Zope e gestire il bilanciamento di carico e la sincronizzazione tra di esse oltre che per avere macchine ridondanti o tra siti che si trovano in luoghi diversi.

Le pagine dinamiche possono essere gestite utilizzando diversi linguaggi: oltre a Python e PERL sono disponibili DTML (*Document Template Markup Language*) e ZPT (*Zope Page Template*). Il primo permette di definire ed utilizzare oggetti e metodi DTML mediante appositi *tag*: si possono richiamare da queste pagine *script* in Python, oppure parti che si ripetono, visualizzare in maniera condizionale parti di una pagina ed anche recuperare dati da un *database* esterno. Questo linguaggio

però, richiedendo l'uso di elementi che non fanno parte di HTML, rende impossibile intervenire su una pagina con un *editor* HTML, in quanto non riconosce gli elementi specifici del DTML. Un esempio di come si presenta una pagina DTML è visibile nella figura 2.4.

```
<dtml-var standard_html_header>
 <form action="Report" method="get">
  <h2><dtml-var document_title></h2>
   Search for News Items:<br>
 <table>
  <tr>
   <th>Content</th>
   <td><input name="content" width=30 value=""></td>
  </tr>
  <tr>
   <th>Author</th>
   <td><input name="author" width=30 value=""></td>
  </tr>
  <tr>
   <td><p>modified since:</p></td>
    <td><input type="hidden" name="date_usage" value="range:min">
     <select name="date:date">
      <option value="<dtml-var expr="ZopeTime(0)" >">Ever</option>
      <option value="<dtml-var expr="ZopeTime() - 1" >">Yesterday</option>
      <option value="<dtml-var expr="ZopeTime() - 7" >">Last Week</option>
      <option value="<dtml-var expr="ZopeTime() - 30" >">Last Month</option>
      <option value="<dtml-var expr="ZopeTime() - 365" >">Last Year</option>
     </select>
    </td>
  </tr>
  <tr>
   <td colspan=2 align=center><input type="SUBMIT" value="Submit Query">
   </td>
  </tr>
  </table>
  </form>
 <dtml-var standard_html_footer>
```

Figura 2.4. Esempio di codice DTML

Un approccio alternativo è di utilizzare ZPT. All'interno dei *tag* HTML di una pagina vengono inseriti attributi TAL[2] (*Template Attribute Language*), permettendo di sostituire il testo statico presente all'interno degli attributi e dei blocchi con parti generate in maniera dinamica. Un esempio di come viene scritta una pagina che

[2]Questi attributi possono assumere valori diversi utilizzando TALES (*TAL Expression Syntax*), e si possono anche definire delle macro utilizzando METAL (*Macro Expansion for TAL*)

utilizza TAL e TALES si può vedere nella figura 2.5. Un *editor* HTML non va a toccare questi attributi, per cui è possibile modificare una pagina che utilizza TAL direttamente ma soprattutto le pagine nell'*editor* visuale possono contenere dati fittizi in maniera tale da vedere immediatamente ed in maniera intuitiva come la pagina dinamica apparirà, con tutti i vantaggi relativi.

La gestione e l'amministrazione di tutto il sistema risulta agevole e coerente per chi abbia un minimo di conoscenze informatiche. Per completezza va detto che la documentazione disponibile non è molto completa per quanto riguarda alcuni aspetti e che trattandosi di un prodotto complesso si può avere qualche difficoltà iniziale nell'imparare ad utilizzarlo.

In conclusione Zope rappresenta un ambiente per le applicazioni WWW molto interessante: la presenza di moduli aggiuntivi permette di poter aggiungere funzionalità complesse con poca fatica. Esistono soluzioni per il *content management* piuttosto interessanti.

La difficoltà più grande alla sua adozione è il fatto che essendo un prodotto indipendente da altri sistemi, nel caso si debba ricorrere ad una soluzione di *hosting* per un sito, è necessario che il fornitore di servizi preveda la possibilità di utilizzare Zope. In altri termini una soluzione di *hosting* basata su Linux, Apache, MySQL e PHP oppure Windows, IIS, Microsoft SQL Server ed ASP non può essere utilizzata per Zope. Se il sistema informativo aziendale utilizza già altre tecnologie, integrare Zope con queste e soprattutto formare gli sviluppatori può non essere una soluzione fattibile in tempi brevi.

```
<html>
<head>
<title tal:content="here/title">Page Title</title>
</head>
<body>
<table border="1" width="100%">
 <tr>
  <th>Number</th><th>Id</th><th>Meta-Type</th><th>Title</th>
 </tr>
 <tr tal:repeat="item container/objectValues">
   <td tal:content="repeat/item/number">#</td>
   <td tal:content="item/getId">Id</td>
   <td tal:content="item/meta_type">Meta-Type</td>
   <td tal:content="item/title">Title</td>
 </tr>
</table>
</body>
</html>
```

Figura 2.5. Esempio di codice TAL

2.3.1.1 Zope/CMF

Zope/CMF [11] è l'estensione per il *content management* di Zope. È composto da una serie di moduli aggiuntivi che permettono di avere oggetti utili per gestire il contenuto (BasicContentServices), flussi di lavoro (WorkflowServices), la personalizzazione utente per delle pagine visualizzate e relativa gestione degli utilizzatori (MembershipServices), la catalogazione del contenuto inserito (CatalogingServices), la costruzione di *forum* di discussione (DiscussionServices), la *syndacation* (SyndicationServices) ed altri componenti utili per la costruzione di un CMS (SiteDesignServices, IntegrationServices, ArchivingServices, RatingServices, TestingServices).

Zope/CMF non è una soluzione "chiavi in mano" ma in realtà fornisce le basi per poter costruire sopra di sé un'applicazione personalizzata, impiegando oggetti e metodi già pronti.

2.3.1.2 Plone

Una soluzione che mette a disposizione un sistema completo per la gestione del contenuto è invece Plone [12]. Basato su Zope e Zope/CMF mette a disposizione all'utilizzatore una applicazione pronta all'uso, mettendo a disposizione ed integrando assieme le componenti di Zope/CMF con una interfaccia utente semplificata.

Si può quindi costruire e personalizzare facilmente l'applicazione sia per creare un "portale internet", su cui fare incontrare una comunità virtuale, sia per costruire un archivio di documentazione interno con la possibilità di annotare i testi presenti, sia creare altri tipi di applicazioni *web*.

Icoya [13] è un'applicazione commerciale basata su Plone, venduta dalla Support AG che è composta da alcuni applicativi personalizzati ulteriormente e dal relativo supporto commerciale: OpenContent, una applicazione di WCMS, OpenCommerce, una soluzione di *e-commerce*, ed OpenCollaboration, una applicazione per l'organizzazione del contenuto per gruppi di lavoro.

Questo è anche un esempio della modularità e della potenza di Zope come *application server* per il *web*, e di come sia possibile un approccio modulare per arrivare a costruire applicazioni complesse.

2.3.2 OpenCMS

OpenCMS [14] è un sistema di *content management* libero basato su J2EE. Questo programma permette di creare siti web *offline* che possono venir pubblicati una volta arrivati ad una versione soddisfacente. Quando un sito è *offline* utenti differenti con differenti permessi possono lavorare su di esso ed inoltre il *project manager* può suddividere il lavoro assegnando compiti ai diversi utenti, che verranno automaticamente informati dei compiti che dovranno eseguire. Il *project manager* una

volta completato il sito lo metterà *online*: le modifiche successive verranno effettuate *offline*, e le parti modificate andranno a sostituire quelle in produzione una volta pronte. Una caratteristica interessante del sistema è di poter generare un sito statico a partire da quello dinamico.

OpenCMS utilizza un motore per *servlet* Java, come ad esempio Apache/Tomcat. Il *servlet* OpenCMS (una classe singola) si occupa sia della presentazione dei dati (tramite HTTP), che dell'accesso al database (tramite JDBC).

Per ogni sito viene definito un *frametemplate* (che, a dispetto del nome non obbliga all'utilizzo dei frame HTML, anche se ovviamente è possibile impiegarli) che definisce la struttura generale delle pagine del sito, un *contenttemplate* all'interno del *frametemplate* che permette di avere presentazioni differenti in parti diverse del sito ed infine il *bodyelement* che è la parte in HTML che definisce il contenuto delle pagine. In altre parole, non si utilizza XML ed XSLT per avere possibilità diverse di visualizzazione, ma XML definisce la struttura generale delle pagine, per cui risulta ragionevolmente facile, per esempio avere una versione *printer friendly* di una pagina, cioé senza banner o formati particolari, mentre è più difficoltoso, per come è strutturato il sistema, avere una versione senza immagini all'interno del corpo della pagina.

È possibile aggiungere nuove funzioni ad OpenCMS mediante l'uso dei moduli. Questi ultimi sono un'estensione dinamica del sistema mediante template XML: per avere ad esempio le funzioni di ricerca testuale per il sito basta installare il modulo appropriato. I moduli sono costituiti da un file compresso che può contenere classi Java, *template*, e documentazione, con una struttura definita. OpenCMS tiene traccia dei moduli installati rendendo così possibile una facile installazione e rimozione di essi.

Per quanto riguarda l'interfaccia utente per l'amministrazione bisogna utilizzare un *browser* recente, preferibilmente Microsoft Internet Explorer 5.5 (se si vuole utilizzare l'editor HTML WYSIWYG) oppure Mozilla 1.0.

2.3.3 Midgard

Midgard [15] è un modulo aggiuntivo del *web server* Apache che estende PHP aggiungendo al linguaggio delle funzionalità per la creazione e la gestione di *template* e per memorizzare e recuperare contenuto dal *database* MySQL. È rilasciato con licenza LGPL. Esiste anche una versione, attualmente in fase sperimentale, Midgard Lite, scritta completamente in PHP e quindi utilizzabile in caso di *hosting* su sistemi generici in cui non è possibile compilare ed installare moduli aggiuntivi per Apache.

Mediante i *template* è possibile creare "stili" che rappresentano la struttura delle varie pagine di un sito. Questi stili hanno un elemento "radice" che definisce la struttura principale ed è costituito da più subelementi, che a loro volta possono essere composti da altri subelementi. Dagli stili possono essere creati dei "sottostili" che

ereditano tutti gli elementi dallo stile superiore, ma che possono avere parti aggiunte o modificate, permettendo una agevole manutenzione della struttura delle pagine. Midgard può visualizzare sia contenuti statici che contenuti dinamici recuperati da una base di dati. Si ha a disposizione una struttura ad albero di argomenti a cui vengono associati articoli ed il corretto *template* ad ogni nodo, attraverso l'utilizzo di codice PHP. Midgard ha una serie fissa di oggetti: oltre agli articoli ed agli argomenti ha ad esempio eventi e calendari, e per questi oggetti sono definite procedure per la creazione, la modifica e la cancellazione, e metodi per ottenere liste di oggetti, le relazioni fra essi ecc...

Per l'amministrazione del sistema sono disponibili alcune applicazioni Midgard che permettono di gestire utilizzatori e struttura del sito attraverso una interfaccia *web*. È possibile avere più applicazioni diverse per la gestione del sistema che possono essere personalizzati per adattarsi meglio alle specifiche esigenze. Per una gestione di un piccolo sito, ad esempio è possibile utilizzare Asgard, contenuto nella distribuzione, oppure è possibile utilizzare Aegir-CMS per una soluzione di *content management* più completa (vedi figura 2.6).

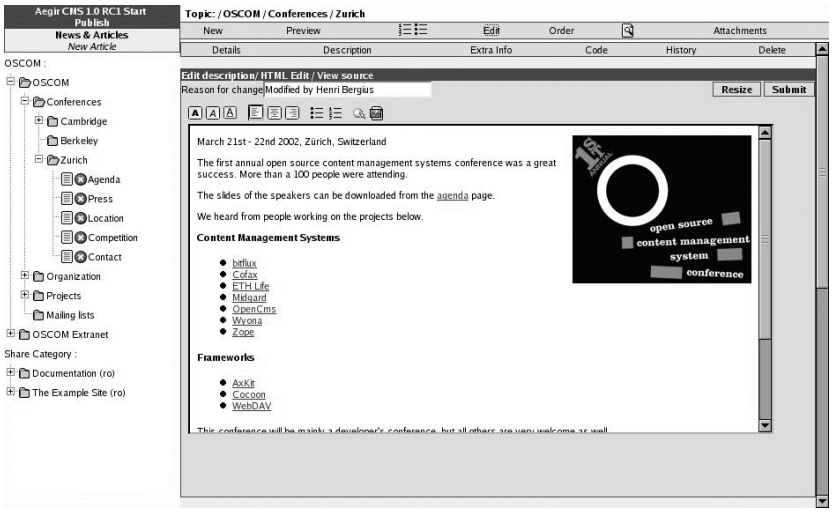


Figura 2.6. Interfaccia di amministrazione di Aegir CMS

Quest'ultimo, conosciuto precedentemente come Nadmin Studio, permette di gestire più siti *web* sulla stessa macchina, la possibilità di utilizzare più codifiche di caratteri e di avere l'interfaccia di amministrazione in più lingue, di avere un sito di test ed un sito di produzione e di mantenere la sincronia fra essi. Inoltre è possibile definire ruoli e flussi di lavoro ed avere un controllo delle revisioni. L'interfaccia per l'inserimento delle pagine è molto *user-friendly* ed è modellata in modo da somigliare a quella di un tipico *word processor*.

2.3.4 Cofax

Cofax [16] è un sistema per la pubblicazione e la gestione di contenuto testuale e multimediale su *web* pensato per facilitare la pubblicazione *online* di giornali: è nato inizialmente come applicazione utilizzata internamente al gruppo editoriale Knight Ridder, e successivamente rilasciato con una licenza *open source*.

Per quanto riguarda la tecnologia utilizzata, Cofax utilizza dei servlet Java, per implementare il nucleo del suo funzionamento e per la memorizzazione dei dati utilizza MySQL attraverso JDBC. Ha una architettura modulare che si può suddividere in quattro parti. La prima, *Cofax Feed System*, si occupa di inserire il contenuto degli articoli, che devono essere in un semplice formato XML, all'interno della base di dati, occupandosi del *versioning* degli stessi e della gestione dei metadati. La seconda, *Cofax Data Warehouse*, è una API che permette di avere un accesso uniforme a diverse basi di dati, senza che le altre parti di Cofax debbano preoccuparsi delle differenze implementative tra un sistema e l'altro. *Cofax CMS* gestisce il flusso di lavoro del sistema e permette un controllo redazionale su quanto viene pubblicato: la versione attuale utilizza Java Servlets e JSP. *Cofax Content Display System* si occupa della presentazione dei contenuti mediante l'utilizzo di *template* HTML ed implementa un sistema di *caching* delle pagine per migliorare le prestazioni.

2.4 Conclusioni

I sistemi di gestione del contenuto sono un tipo di applicazione informatica relativamente nuova, con un mercato molto dinamico, in cui le offerte, sia commerciali che libere ed *open source*, hanno funzionalità assai diverse fra loro ed un confronto diretto non è quindi possibile, anche perché non esiste una definizione precisa di cosa sia il *content management*. Esistono sistemi fortemente orientati al *web publishing*, altri che sono pensati per la gestione di documenti oppure per il commercio elettronico, altri ancora pensati per gestire pubblicazioni cartacee.

L'adozione di un CMS è quindi una operazione complessa e che va pianificata accuratamente a livello aziendale. Bisogna capire quali siano esattamente le esigenze che un CMS deve soddisfare all'interno della realtà organizzativa e quindi chi sono le persone all'interno dell'azienda che risulteranno più coinvolte nell'utilizzo del sistema.

L'attivazione del sistema spesso porta a dover riorganizzare le procedure interne o quanto meno analizzarle e formalizzarle ed inoltre un sistema di gestione del contenuto è un'applicazione che va sempre e comunque personalizzata per essere utilizzabile, con dei costi e delle tempistiche che devono essere tenuti in conto. Vanno quindi valutati gli aspetti più importanti che devono essere risolti da una soluzione CM, ad esempio valutare l'importanza relativa tra gli aspetti di acquisizione,

gestione, flusso di lavoro e pubblicazione.

L'offerta di prodotti è decisamente variegata ed oltretutto una volta decisa una soluzione tornare indietro e cambiare può risultare decisamente costoso per il fatto che le personalizzazioni e la formazione del personale non possono essere riutilizzate. L'uso di una soluzione sbagliata può anche rendere difficoltosi ed inutilmente complessi i flussi di lavoro aziendali ed essere molto inefficiente: alla fine si rischia di dover scrivere un CMS anziché utilizzare un CMS, magari dopo aver pagato costose licenze o peggio di avere un sistema sottoutilizzato e fondamentalmente inutile.

Decidere se utilizzare un sistema commerciale, piuttosto che una soluzione libera od *open source*, oppure costruire in casa una applicazione *ex novo* non è cosa facile, ma d'altra parte per l'attuale crescita del *web* e per soddisfare al meglio gli utilizzatori interni ed i visitatori esterni dei siti, l'uso di un sistema di gestione del contenuto diventa una necessità in molte situazioni, una volta che il numero di utenti o la quantità d'informazione supera una certa soglia.

Capitolo 3

Un caso pratico: il WCMS per il servizio Passepartout

Dopo avere affrontato da un punto di vista generale la storia del *world wide web* e di come i sistemi di gestione del contenuto possano aiutare a lavorare su siti di grandi dimensioni, in questa parte si andrà ad analizzare come tutto questo si colloca in un contesto reale. Il contesto di riferimento è il servizio Informahandicap[1], curato dal servizio Passepartout del Comune di Torino ed ospitato all'interno del sito *web* del Comune di Torino. Questo sito è in *hosting* all'interno delle strutture del Consorzio per il Sistema Informativo (CSI). Verranno discussi gli scopi del sito, la sua evoluzione e di come nel corso del tempo siano sorte nuove esigenze che lo hanno portato da essere un sito statico ad un sito dinamico. Si esamineranno inoltre quali siano le esigenze attuali e di come queste esigenze abbiano generato la necessità di avere un sistema di *web content management* diverso dal sistema di generazione dinamica delle pagine attualmente implementato.

3.1 Obiettivi del sito e situazione attuale

Lo scopo che si prefigge il sito Informahandicap Piemonte è di poter distribuire tempestivamente informazioni sulla legislazione, sui servizi, sugli aspetti fiscali, sanitari, lavorativi, educativi e sociali che riguardano il mondo dell'handicap, in maniera completa e certificata. L'*audience* prevista è quella dei portatori di *handicap* ed i loro famigliari, enti ed associazioni che erogano servizi ai disabili, operatori del settore ed i cittadini in generale. Per una discussione più approfondita sugli obiettivi del sito e per alcuni esempi del suo contenuto si rimanda a [30, §3.1-§3.3].

[1] `http://www.comune.torino.it/pass/`

3.2 La situazione precedente: il sito statico

Attualmente il sito viene erogato tramite una serie di pagine statiche, quindi scritte a mano da un *web master* e poi pubblicate manualmente. Questo modo di procedere ha rapidamente portato a mostrare le limitazioni tipiche dell'approccio in cui un'unica persona (od un gruppo ristretto) si trova a dover inserire nel sito tutte le informazioni che devono essere pubblicate, con i conseguenti ritardi negli aggiornamenti.

Non essendo possibile avere un controllo globale sulla presentazione delle pagine, è facile che parti diverse del sito possano apparire graficamente diverse. Inoltre dovendo gestire anche una gestione accessibile del sito, bisogna che vengano create e gestite due versioni distinte della stessa pagina, con identico contenuto duplicato in entrambe: è facile che venga aggiornato solamente il contenuto una delle pagine, creando disallineamenti tra la versione *standard* e la versione accessibile.

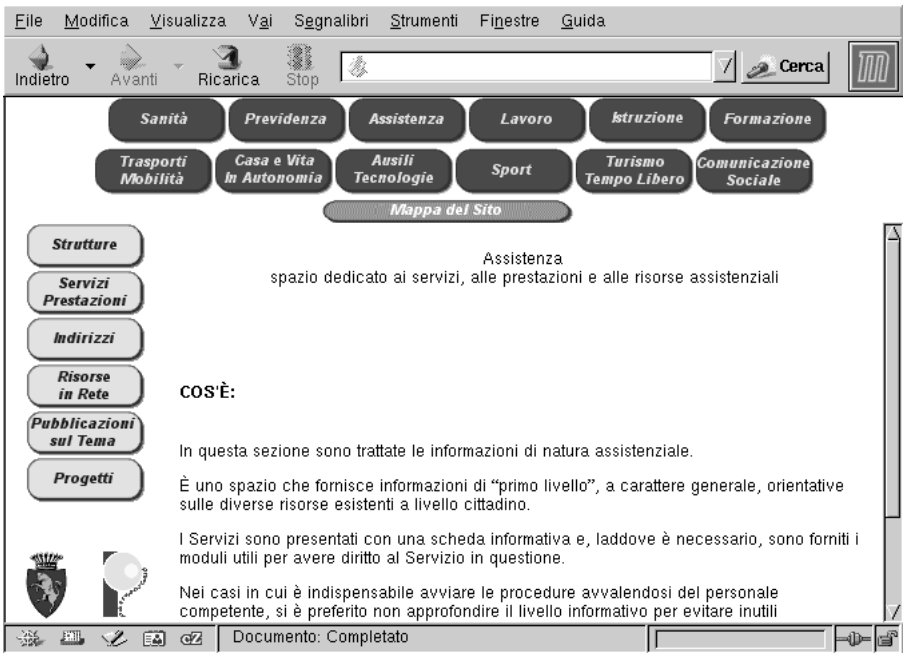


Figura 3.1. Una pagina del sito statico Informahandicap in Mozilla

La necessità di mantenere una versione accessibile deriva dal fatto che la versione *standard* utilizza per la navigazione una serie di *frame* ed inoltre *link* di navigazione sono tutti fatti con immagini *bitmap* che contengono del testo, causando problemi agli ipovedenti, che non possono ingrandire il testo delle *bitmap*. L'uso dei *frame* rende più complessa la navigazione nel caso vengano utilizzati *browser* testuali o *screen reader*. Visto il pubblico di riferimento del sito, risulta ancor più importante

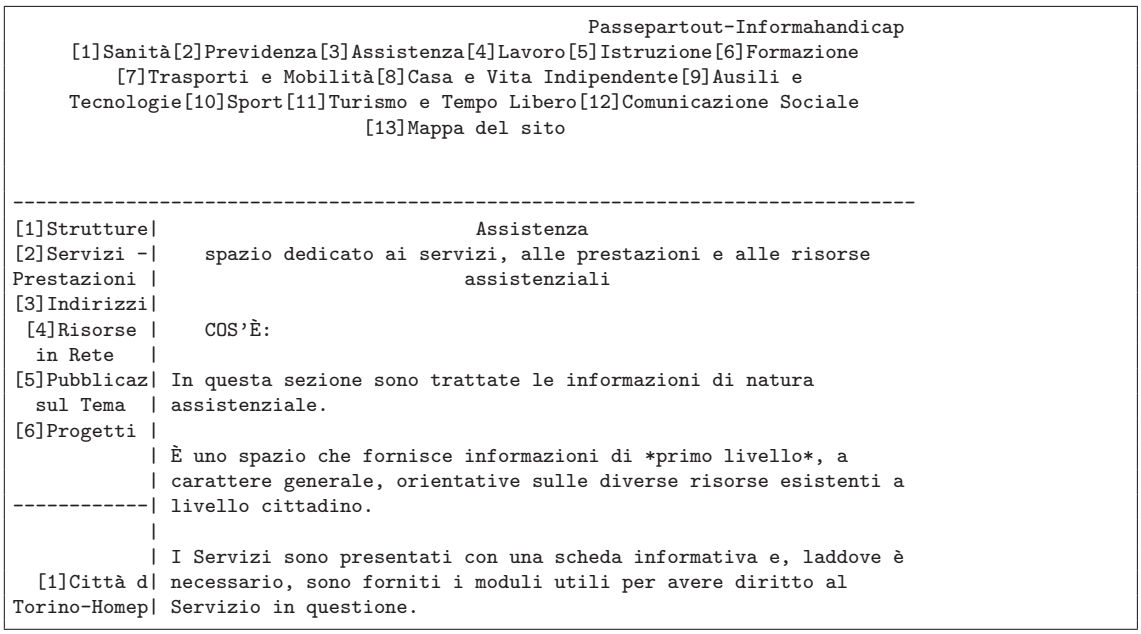


Figura 3.2. Una pagina del sito statico Informahandicap in un *browser* testuale

rispetto a siti *web* con scopi diversi che i contenuti siano accessibili a persone con disabilità percettive o motorie.

Uno degli obiettivi del servizio Passepartout è quello formare persone disabili per metterle in grado di creare contenuto per il sito *web*. Il dover utilizzare un sito statico implica comunque che i *web master* debbano comunque controllare e modificare codice HTML in modo che possa essere inserito all'interno del sito, creando così un collo di bottiglia e vanificando comunque parte degli scopi di questa formazione.

3.3 Il sito dinamico

Per risolvere questi problemi, nel 2001 si è deciso di riorganizzare il sito *web* e nel contempo trasformarlo da sito statico a sito dinamico, con la memorizzazione del contenuto all'interno di un RDBMS, non più in forma di pagine HTML, ma come una serie di campi di testo contenuti all'interno delle tabelle del *database*. L'applicazione creata ha come caratteristiche principali quelle elencate qui di seguito.

- Interfaccia basata su *web* per l'inserimento del testo: non è più necessario quindi conoscere HTML per inserire materiale nel sito.

- Semplice modello di controllo del materiale pubblicato. Esistono due classi di operatori: i "giornalisti" che inseriscono il contenuto ed i "redattori" che autorizzano la pubblicazione del sito.
- Uso di modelli di pagina inseriti nella base di dati, in modo da avere in maniera facile un *look-and-feel* consistente per tutto il sito.
- Erogazione di pagine con più stili a partire dagli stessi dati.
- Possibilità di includere sottopagine all'interno di altre pagine per i testi che si ripetono.

Nel contempo si è abbandonata la struttura di navigazione basata su *frame* e si è passati ad avere una barra di navigazione in cui veniva memorizzato il percorso che il singolo utilizzatore aveva seguito per giungere in una pagina. In altri termini si è avuta una destrutturazione del sito, che non risultava più avere una classica suddivisione ad albero di *directory*, ma si aveva una struttura piana, in cui le pagine formano un grafo dove i collegamenti all'interno delle pagine creano la struttura del sito [30, §5].

L'applicazione è basata sul linguaggio di *scripting* PHP. Vi sono una serie di pagine PHP che si occupano di erogare il contenuto richiesto dai visitatori del sito, ed una serie di pagine ne che servono all'amministrazione, cioè inserire o modificare i contenuti presenti nella base di dati ed approvare le variazioni eseguite per renderle disponibili agli utenti. Come *database* si è scelto di utilizzare Interbase di Borland.

Per l'accesso all'RDBMS anziché utilizzare direttamente le funzioni native del PHP, che sono diverse sia per nome che per funzionalità per ogni diverso tipo di base di dati, si è utilizzata una libreria di classi, ADODB, che permette di avere un accesso unificato ai vari *database* supportati da PHP con una serie di funzioni simili nel comportamento a quelle presenti nella libreria ADO disponibile sui sistemi Microsoft.Per memorizzare le pagine all'interno del *database* viene utilizzata una struttura in due parti. Esistono alcune tabelle che servono a definire una struttura di modello, cioè quali campi e di che tipo devono apparire in una pagina, in che ordine e con quale stile.

Ogni pagina è associata quindi ad un modello preciso. A sua volta i contenuti della pagina vengono memorizzati in una tabella. Nel momento in cui una pagina deve essere erogata, viene recuperato il modello associato alla pagina ed i contenuti da associare a ciascun campo, e seguendo in ordine i campi del modello viene associato ad essi il contenuto e viene così visualizzata la pagina.

Il sistema però a tutt'oggi non è ancora utilizzato in produzione per l'erogazione delle pagine presenti sul sito del Comune di Torino. Al momento di mettere *online* il sito, sulle macchine del CSI, si è scoperto che a causa di un cambiamento delle *policy* dell'ente non era possibile né mettere una macchina in *housing* nella loro sede, né

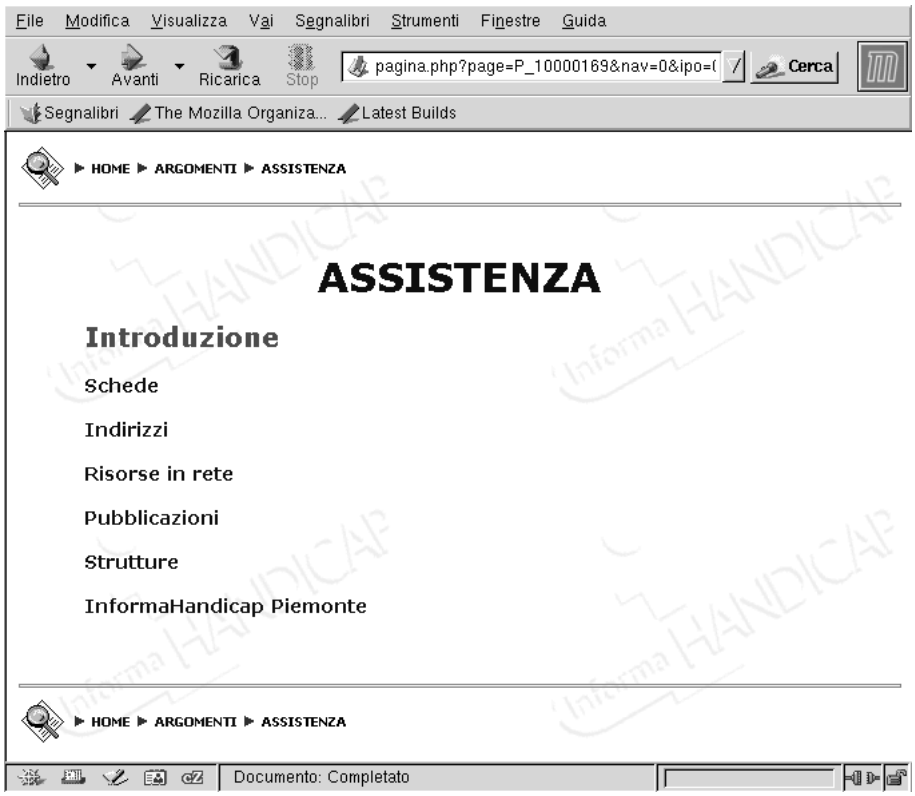


Figura 3.3. Pagina erogata dal sito dinamico

avere la possibilità di utilizzare Interbase come RDBMS sulle macchine in *hosting*. L'unico *database* disponibile al CSI è Oracle, e quindi è risultato necessario modificare l'applicazione costruita per poter essere utilizzata in produzione. L'utilizzo delle classi ADODB ha facilitato la transizione da Interbase ad Oracle, ma questo ha portato via comunque diverso tempo. Le *policy* di sicurezza del CSI inoltre impediscono di poter caricare nuovi *file* sui loro server *web* se non dalle macchine collegate alla loro *intranet*, rendendo difficoltoso poter procedere speditamente alla verifica del corretto funzionamento nell'ambiente di produzione e quindi alla messa in opera del sistema.

Durate tutto questo periodo il sistema di erogazione di pagine dinamiche è stato comunque ampiamente utilizzato in prova, permettendo così di definire meglio quali fossero le esigenze per l'amministrazione del sito. Le necessità del servizio Passepartout riguardanti il sito *web* si sono nel tempo modificate ed evolute: il sito InformaHandicap Piemonte tenderà in futuro a diventare un portale regionale sulle tematiche dell'*handicap* e quindi con la possibilità che più entità geograficamente distanti possano inserire direttamente contenuti specifici alle loro attività nel sito.

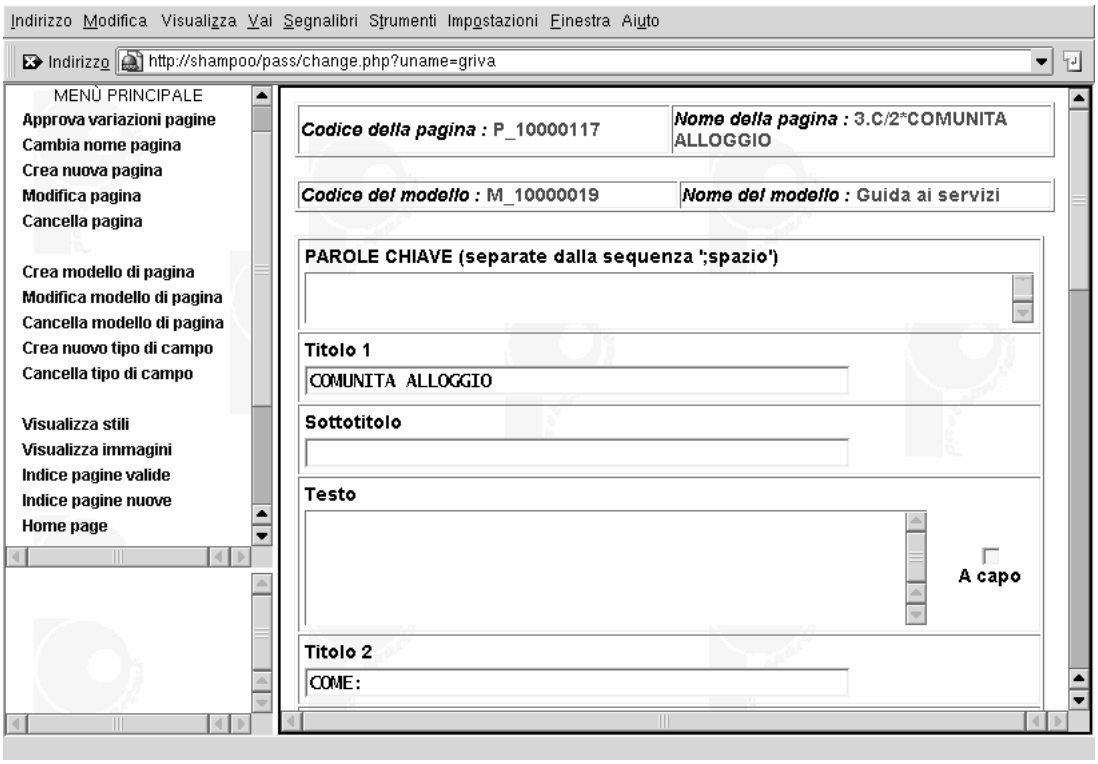


Figura 3.4. Pagina di *editing* del sito dinamico

Si è quindi reso necessario provvedere ad una serie di modifiche ed aggiornamenti al sistema di erogazione per rispondere meglio alle nuove esigenze.

3.4 Requisiti

Dall'esperienza maturata con la versione attuale del sito e dall'analisi delle nuove esigenze organizzative si è ricavata una serie di requisiti che deve avere il nuovo sito.

La caratteristica in cui la nuova versione si discosta di più da quella vecchia è che il *workflow* per la pubblicazione di una pagina è più complesso rispetto al semplice modello di creazione e successiva approvazione di una pagina.

Nel nuovo scenario esistono un certo numero di gruppi di lavoro, supervisionati da un redattore che si occupa di supervisionare il lavoro dei giornalisti del gruppo e di approvare o meno i contenuti a lui sottoposti dai membri del gruppo per la pubblicazione sul sito. I gruppi di lavoro sono organizzati in una struttura gerarchica ad albero che parte da un gruppo "radice" che ha la responsabilità ed il controllo di tutto quanto viene pubblicato nel sito.

Il contenuto prodotto dai giornalisti del gruppo, per essere prima di essere pubblicato, deve essere approvato dal suo redattore e quindi per chi è nel gruppo radice nulla cambia rispetto al *workflow* attualmente in uso per il sito dinamico.

Al di sotto del gruppo radice possono essere creati altri gruppi, a cui viene data delega per la gestione e la creazione di pagine all'interno del sito: a loro volta questi gruppi possono avere sotto di sé altri gruppi a cui delegare la gestione di parte del sito e così via.

Nelle deleghe possono essere di due tipi: con fiducia e senza fiducia. Le deleghe senza fiducia il contenuto approvato dal redattore del gruppo delegato deve essere comunque approvato anche dal redattore del gruppo delegante. In caso di delega con fiducia, è invece sufficiente che la variazione di contenuto sia approvata dal redattore del gruppo delegato perché la pagina sia approvata anche a livello del gruppo delegante ed il redattore di quest'ultimo viene semplicemente notificato. Se il gruppo delegante è il gruppo radice la pagina viene resa disponibile per la pubblicazione.

L'uso di un *workflow* più complesso implica che una pagina non ha solo gli stati approvata e da approvare, ma diversi stati, come ad esempio creata, approvata, in attesa, in linea, scaduta, aggiornata, ecc... Inoltre le pagine non sono tutte dello stesso tipo, ma appartengono ad un gruppo e vengono create da uno specifico giornalista in maniera da poter sapere chi ha effettuato le ultime modifiche ad una determinata pagina e fare in modo che il giornalista di un gruppo possa modificare solo le pagine appartenenti al suo gruppo ed eventualmente quelle dei gruppi cui ha dato deleghe, ma non a pagine create da gruppi a cui non ha dato deleghe dirette od indirette. Il gruppo radice quindi può operare comunque su tutto il sito senza limitazioni. Il redattore del gruppo a cui appartiene una pagina modificata ed approvata da un redattore di livello più elevato può essere notificato dell'avvenuta modifica della pagina.

Deve essere inoltre possibile mantenere disponibili all'interno del sistema le versioni precedenti e non più disponibili al pubblico di ogni pagina pubblicata, sia per avere la possibilità di avere una sorta di "archivio storico" di ogni pagina, sia per poter eventualmente recuperarne una versione precedente per rimetterla *online* in caso ad esempio fossero stati generati aggiornamenti erronei di una data pagina.

Le pagine disponibili per la pubblicazione, oltre a poter essere messe *online* per la fruizione da parte dell'utenza immediatamente all'approvazione possono anche rimanere in attesa ed andare in linea ad una data futura prestabilita. Allo stesso modo una pagina può avere una data di scadenza, dopo la quale la pagina non può essere più richiesta dai visitatori del sito.

Le pagine non possono essere create con un formato libero, ma devono seguire dei modelli di pagina, che vanno a definire il tipo di contenuto presente, con l'associazione allo stile grafico relativo e la sua disposizione nella pagina. Può essere possibile inserire anche del testo *standard* o di esempio all'interno dei modelli.

I modelli di pagina vengono decisi dal redattore del nodo principale che li costruisce e li modifica utilizzando le funzionalità preposte nel sistema di amministrazione. Per la scelta dell'aspetto grafico dei singoli elementi (colore, dimensione, tipo di carattere, giustificazione...) devono essere utilizzati i fogli di stile e non l'uso degli elementi di formattazione deprecati in HTML 4.01.

La struttura delle pagine deve essere meno vincolante rispetto a quella del sistema di gestione attuali e permettere di definire pochi e semplici modelli. Il sistema di *editing* dei modelli deve essere semplice da utilizzare, se possibile con una interfaccia WYSIWYG. Anche il sistema di creazione e modifica delle pagine deve essere facile da utilizzare e permettere una gestione flessibile delle pagine e se possibile utilizzare un'interfaccia WYSIWYG.

Per quanto risulti possibile la struttura del sito nella nuova versione deve essere uguale a quella della versione attuale, cioè con una organizzazione "piatta" e quindi è compito dei redattori verificare che le nuove pagine create siano collegate in maniera logica con le altre pagine. L'aspetto del sito per i visitatori deve rimanere il più possibile simile a quello della versione attuale ed in particolare devono essere previsti due stili differenti, per avere una versione normale ed una ad alto contrasto per ipovedenti.

Le pagine erogate all'utenza devono poter essere conformi alla raccomandazioni del W3C per l'accessibilità [4] ed essere conformi almeno a livello WCAG A, e se possibile a livello WCAG AA o WCAG AAA. Per quanto possibile anche le pagine generate per l'amministrazione del sito dovranno essere conformi alle raccomandazioni W3C per l'accessibilità: l'utilizzo di Javascript e di oggetti eseguibili lato *client* (come Applet Java od ActiveX) che rendano parte del sito di amministrazione non conforme alla raccomandazione dovranno essere utilizzati solo se assolutamente necessari.

Deve essere prevista un'interfaccia *web* per la creazione e l'eliminazione degli utenti del sistema, la creazione e l'eliminazione di gruppi, l'inserimento di un utente all'interno di un gruppo e l'indicazione di quale utente sia redattore di un gruppo. L'accesso al sistema di gestione pagine deve autenticare tramite *password* gli utenti che si collegano, e deve essere previsto che gli utenti possano cambiare liberamente la loro *password*. Gli utenti non autenticati non devono accedere all'area di amministrazione del sistema.

Deve essere inoltre presente un motore di ricerca per agevolare gli utilizzatori del sito nel trovare le informazioni di loro interesse. Per scopi statistici deve essere prevista la possibilità di tenere traccia del contenuto delle richieste fatte al motore di ricerca da parte degli utenti.

Deve essere prevista la possibilità di aggiornare o sostituire il motore di ricerca senza che questo vada ad influire eccessivamente sulle altre parti dell'applicazione. Non bisogna impedire deliberatamente l'accesso del sito ai motori di ricerca esterni e se possibile bisogna rendere facile l'indicizzazione esterna dei contenuti.

Nel caso risulti necessario per esigenze organizzative avere dati statistici sull'utilizzo del sito può essere anche presente un sistema di *logging* delle pagine richieste dagli utenti indipendente da quello utilizzato dal *web server* ed eventualmente con una quantità maggiore di dettagli rispetto a quest'ultimo.

Devono essere previste le funzionalità necessarie all'estrazione dalla base di dati dei dati statistici relativi all'accesso ed all'uso del motore di ricerca per la loro successiva elaborazione mediante l'utilizzo di strumenti esterni all'applicazione, come ad esempio l'uso di fogli di calcolo od applicazioni specifiche per l'analisi dei *log* di un *web server*.

3.5 Scelta dell'architettura del sistema

Le strade percorribili per poter costruire una applicazione *web* che risponda a queste caratteristiche sono sostanzialmente due. La prima possibilità è quella di costruire un'applicazione *ad hoc*, eventualmente basata su quella esistente. La seconda possibilità è quella di andare a scegliere un opportuno sistema di gestione del contenuto esistente e personalizzarlo per le esigenze specifiche del sito.

Come si è già accennato nel paragrafo 1.5, entrambe le scelte presentano sia vantaggi che svantaggi, che sono stati attentamente valutati prima di procedere ad un progetto più dettagliato del nuovo sistema di gestione del contenuto.

3.5.1 Uso di un CMS esistente

La soluzione di utilizzare un CMS già esistente anziché costruirne uno personalizzato è sicuramente la metodologia più interessante. Utilizzare un sistema già fatto permette di avere a disposizione tutta una serie di oggetti già costruiti e funzionanti correttamente, che non devono essere scritti da zero.

La presenza di una comunità di sviluppatori, nel caso di sistemi liberi, o di uno *staff* di supporto tecnico, nel caso di sistemi proprietari, permettono di recuperare le informazioni necessarie a personalizzare in maniera efficace la propria applicazione ed ottenere un sistema più facile da gestire nel tempo: avendo un sistema che si pone ad un livello di astrazione più alto, rispetto ad un programma scritto ad esempio in Java o PHP risulta più facile capire quali sono le personalizzazioni effettuate e le interazioni fra le varie funzioni esistenti nel sistema. Il tempo necessario per avere un'applicazione funzionante risulta quindi decisamente inferiore rispetto alla scrittura di una applicazione da zero, specie se il CMS è stato scelto in modo tale da adattarsi bene alla situazione specifica. L'aggiunta di funzionalità particolari, non previste inizialmente, può risultare molto semplice, e ridursi ad una modifica di un *file* di configurazione o l'abilitazione di un modulo, anziché dover scrivere nuove funzioni o classi come è necessario fare per una applicazione *ad hoc*. La valutazione di

quale sia il CMS già pronto deve essere fatta attentamente per ogni caso, altrimenti si rischia di dover costruire un CMS sopra un CMS, anzichè semplicemente usare un CMS, vanificando così tutti i vantaggi derivanti dall'uso di un'applicazione *off-the-shelf*.

La prima strada che si è quindi valutata è stata quella appunti di utilizzare un WCMS già costruito. Si sono quindi analizzati alcuni tra i principali *content management system* liberi presenti sul mercato per valutare la fattibilità di questo approccio. Si è quindi fatta un'analisi di fattibilità riguardante Zope/CMF/Plone, Midgard/Aegir-CMS, OpenCMS e Cofax. Per una descrizione delle caratteristiche principali di questi prodotti si rimanda al paragrafo 2.3. I CMS commerciali non sono stati presi in considerazione sia per una questione di costi che anche per la loro minore possibilità di personalizzazione.

La soluzione più promettente, in particolare per la possibilità di gestire *workflow* complessi, è sicuramente data da Zope: se il sito fosse su una macchina in *housing* molto probabilmente si sarebbe scelta una soluzione basata su Zope/CMF/Plone, ma come si è visto prima richiede per il suo funzionamento un sistema dedicato o comunque un'installazione di Zope e Python su una macchina.

Allo stesso modo Midgard richiede l'aggiunta di una libreria condivisa in Apache (esiste un'implementazione in PHP puro, ma allo stato attuale non è ancora completamente sviluppata), ed utilizza MySQL come sistema di archiviazione. Midgard/Asgard, con l'uso dell'implementazione di Midgard con PHP può funzionare con la configurazione che viene solitamente fornita da un *hosting provider* cioè Linux, Apache, MySQL e PHP. L'*hosting* fornito dal CSI, come si è accennato prima, usa come RDBMS Oracle e si discosta quindi da questa soluzione *"standard"*.

Cofax ed OpenCMS sono state invece scartate perché il loro funzionamento non è adatto ai nostri scopi, in particolare perché presentano una gestione del *workflow* semplice e non personalizzabile. Andare ad inserire un'applicazione basata su J2EE (si ricordi che Cofax ed OpenCMS sono applicazioni che utilizzano JSP, JDBC e JavaBean) avrebbe richiesto l'abilitazione a livello di *hosting* di questo ambiente, con la necessità di seguire una lunga trafila burocratica. Apache/Tomcat è disponibile al CSI e quindi tecnicamente sarebbe stato possibile andare ad utilizzare applicazioni basate su Java lato *server*, ma sarebbe stata necessaria una lunga trafila burocratica per ottenere le autorizzazioni necessarie per accedere all'ambiente J2EE.

3.5.2 Aggiornamento dell'applicazione in uso

Si è quindi deciso di andare ad aggiornare ed estendere l'applicazione dinamica esistente scritta in PHP. La soluzione presenta comunque alcuni vantaggi rispetto all'utilizzo di un CMS già esistenti. Il vantaggio principale, che come appena detto è stato quello che ha fatto decidere per questa strada, è che una applicazione scritta appositamente si può adattare molto bene per un ambiente operativo preesistente,

permettendo di sfruttare le sue particolarità ed aggirando le limitazioni presenti: nel nostro caso si ha a disposizione il *web server* Apache che può erogare pagine scritte in PHP che vanno ad interagire con la base di dati relazionale Oracle 8.1.7. Un altro vantaggio è che risulta possibile mantenere una struttura del sito, dal lato dell'erogazione del contenuto, praticamente identica a quella del sistema attuale.

Lavorare ad un livello di astrazione più basso permette, anzi obbliga, a gestire tutti gli aspetti relativi alla struttura interna del sito, il metodo di memorizzazione dei dati, la gestione del flusso di lavoro, la parte di pubblicazione del contenuto, l'interfaccia per l'amministrazione del sistema ed il metodo per l'inserimento del contenuto. Nel caso preso in esame, la presenza di un'applicazione per la gestione del contenuto già funzionante rende lo sviluppo più semplice, sia perché alcune parti dell'applicazione esistente possono essere riutilizzate, sia perché il lavoro di analisi fatto per produrre l'applicazione può comunque essere molto utile anche se alcune parti dell'applicazione esistente devono essere modificate in maniera profonda o riscritte da capo per poter gestire nuove funzionalità richieste.

Bisogna aggiungere per completezza, astraendo dal caso specifico, che molte soluzioni di *hosting* presso ISP generici non permettono comunque di decidere liberamente quali ambienti siano disponibili, anche se va detto che alcuni ISP mettono a disposizione un ambiente Zope, ma non sono così diffusi come quelli che mettono a disposizione ambienti MySQL/PHP. In molti casi però un sito che richieda soluzioni di *content management* complesse ha una dimensione tale per cui può risultare conveniente avere una macchina in *housing*, se non addirittura avere le macchine in casa anziché da un ISP, rispetto ad una soluzione di *hosting*.

Per un ISP risulta comunque conveniente dal punto di vista della gestione fornire un *hosting* standard e prevedere l'*housing* gestito per chi necessita di configurazioni particolari, anche perché molti siti sono statici od usano il PHP per semplici *script* di supporto: l'utilizzo di CMS complessi è ancora piuttosto limitato perché un numero elevato di ISP proponga soluzioni di *hosting* di questo tipo, anche se va detto che esiste qualche offerta commerciale in questo senso.

Se non si ha la possibilità di usare un ISP che supporti Zope come opzione di *hosting*, diventa necessario per chi volesse passare a soluzioni che utilizzano un CMS valutare se passare quantomeno ad una macchina in *housing*, se non avere il *web server* che eroga i servizi all'interno dell'azienda. Quest'ultima soluzione diventa una strada quasi obbligata se si vuole avere un sistema sia per *internet/intranet* oppure quando sia necessario integrare il sistema di gestione del contenuto con il sistema informativo esistente.

3.6 Lo sviluppo dell'applicazione

Per sviluppare la nuova versione dell'applicazione, come si è appena detto, si è deciso di partire dall'applicazione esistente espandendola e modificandola. Si è anche deciso un approccio evolutivo, sia per valutare la fattibilità delle varie soluzioni possibili e soprattutto per valutare se la strada dell'espansione del sistema di gestione del contenuto esistente era una strada fattibile.

Non avendo ancora, all'inizio dello sviluppo, la possibilità di utilizzare un'istanza di Oracle, si è deciso di effettuare un *porting* dell'applicazione esistente sotto il *database* relazionale PostgreSQL: l'utilizzo delle classi di ADODB ha reso la transizione possibile ed inoltre essendoci già stato un *porting* da Interbase ad Oracle, alcuni punti critici dell'applicazione erano stati risolti. L'obiettivo era quello di avere un ambiente adatto a creare rapidamente un prototipo della nuova applicazione e non andare direttamente in produzione, in quanto la scelta dell'RDBMS di produzione era comunque obbligata.

La scelta di utilizzare PostgreSQL anziché, come sarebbe apparso più logico, Interbase è dovuta fondamentalmente a due fattori. Il primo è che la vecchia versione dell'applicazione aveva subito diversi aggiornamenti anche dopo essere stata convertita in Oracle, e che quindi un *back porting* sarebbe stato comunque necessario anche utilizzando Interbase. Il secondo fattore è che avendo utilizzato per le prime prove una macchina che utilizza il sistema operativo GNU/Linux e la distribuzione Debian "Woody" 3.0, l'installazione di PostgreSQL ha richiesto pochi minuti: i pacchetti Debian necessari per PostgreSQL sono presenti nella distribuzione ufficiale e quindi non è necessario né aggiungere sorgenti di pacchetti esterni, né procedere all'installazione manuale. Si è scartato anche l'utilizzo di MySQL perché con questo *database* non è possibile avere integrità referenziale tra le varie tabelle, non si possono utilizzare *trigger* e transazioni. Inoltre il meccanismo di *lock* di righe e tabelle di PostgreSQL è compatibile a livello sintattico con quello utilizzato in Oracle.

Dopo la prima fase, in cui si è costruito un prototipo del sistema, si è giunti alla conclusione che era necessaria una riscrittura piuttosto pesante dell'applicazione per poter implementare le nuove funzionalità necessarie alla gestione del *workflow* e per superare le rigidità dell'applicazione esistente riguardanti la gestione dei modelli e delle pagine: si è deciso di utilizzare un sistema basato su XML invece del sistema basato su tabelle collegate fra loro.

L'adozione di XML ha permesso di semplificare il codice dell'applicazione ed allo stesso tempo rendere più semplice e flessibile l'utilizzo del sistema. La struttura delle tabelle della base di dati ha risentito della riscrittura dell'applicazione, subendo modifiche profonde: fondamentalmente il passaggio alla memorizzazione del contenuto in XML ha permesso di eliminare la tabella che memorizzava il contenuto e soprattutto il legame stretto che nella versione attuale esiste tra pagina, modello e contenuto, mentre l'uso di un *workflow* più complesso ha reso necessario sia

modificare la gestione degli utenti che avere traccia dello stato di una pagina.

Si è quindi proceduto ad una riscrittura e ad una riorganizzazione del sistema di gestione del contenuto, arrivando ad un prodotto diverso dalla versione attuale, anche se i legami tra la nuova e la vecchia versione sono piuttosto evidenti e l'esperienza accumulata con la scrittura della versione attuale ha permesso di avere molto chiare quali fossero le necessità più importanti .

Successivamente, una volta che il nuovo sistema di gestione del contenuto era arrivato ad uno stato sufficientemente stabile, si è di nuovo portata l'applicazione a funzionare sotto Oracle, anche perché nel frattempo si era riusciti ad avere un'istanza di Oracle funzionante ed utilizzabile per provvedere agli ulteriori sviluppi del sistema. In particolare si è provveduto ad una riorganizzazione della struttura del sistema di amministrazione e ad una generale ripulitura di HTML generato. Per il sistema di amministrazione si è abbandonata la struttura a *frame* preferendo un sistema a pagina unica.

Infine, una volta create le funzioni per la gestione di tutte le caratteristiche richieste si è passati ad una fase di prova e di *debug* dell'applicazione.

3.7 Conclusioni

La particolarità dell'ambiente di produzione su cui deve funzionare il *content management system* per il sito InformaHandicap Piemonte ha portato a scegliere come soluzione la riscrittura e l'espansione del WCMS già sviluppato, anziché l'utilizzo di pacchetti applicativi già pronti. La tecnologia utilizzata è rimasta quella utilizzata nella versione precedente, cioè l'uso di PHP con le classi ADODB e l'utilizzo del RDBMS Oracle per la memorizzazione dei contenuti: la struttura delle tabelle e del programma risulta comunque profondamente modificata rispetto alla versione precedente, sia per il passaggio all'uso di XML per la memorizzazione del contenuto all'interno della base di dati, che per la profonda modifica del *workflow* utilizzato per la pubblicazione delle pagine e relativa necessità di dover gestire gruppi di lavoro, con associati ad essi gli utenti del gruppo.

Se non ci fossero state limitazioni piuttosto pesanti sul sistema di produzione che deve essere utilizzato, la soluzione migliore sarebbe stata sicuramente quella di personalizzare un CMS libero già pronto che rispondesse alle esigenze del sito. In questo modo il tempo di sviluppo si sarebbe ridotto di molto, avendo a disposizione moduli già scritti per poter fornire i servizi richiesti. In un ambito in cui sia più flessibile la scelta del *software* per l'erogazione dei servizi *web* rispetto a quello del CSI si deve valutare più attentamente le possibilità che un CMS già disponibile può offrire.

Capitolo 4

XML

I sistemi di gestione del contenuto, come detto nel capitolo 2 utilizzano spesso XML. Sistemi di produttività per ufficio come OpenOffice di Sun utilizzano XML per memorizzare i propri dati. I browser più recenti come Netscape 6 e Microsoft Internet Explorer 5 possono visualizzare documenti in XML. Basi di dati relazionali possono presentare i contenuti delle *query* e la struttura delle tabelle mediante XML. Dalla sua introduzione, avvenuta alla fine degli anni '90, ha fatto nascere nell'industria informatica una grande quantità di *standard*, acronimi ed applicazioni.

In questo capitolo si spiegherà che cosa è XML, si percorrerà la storia che ha portato alla definizione dello *standard*, facendo notare le differenze che ci sono tra XML, SGML, XHTML ed HTML, cercando anche di far vedere alcuni possibili campi di applicazione. Infine si tratterà di come in XML sia possibile definire elementi e la loro semantica con DTD con accenni a RELAX NG ed Xschema ed infine di come da un documento XML si possano ottenere una presentazione gradevole utilizzando i CSS.

4.1 Cos'è XML

XML è un acronimo che sta per *eXtensible Markup Language*, ma a dispetto del nome non è un linguaggio di *markup*, come ad esempio HTML. È un metalinguaggio che permette, seguendo alcune regole, di definire infiniti linguaggi di *markup*. Bisogna quindi capire che cos'è esattamente un linguaggio di *markup*.

Il *markup* è l'informazione che viene aggiunta ad un documento, migliorandone e definendone meglio il significato, in quanto permette di identificare i vari blocchi ed a metterli in relazione fra di loro. In un libro per esempio la suddivisione in capitoli e paragrafi è ottenuta mediante l'utilizzo di caratteri differenti ed inserendo

opportunamente spaziature e margini diversi. Un linguaggio di *markup* è un insieme di simboli che vengono inseriti nel testo di un documento per suddividere ed identificare i vari pezzi del documento.

Nell'elaborazione automatica dei documenti è importante utilizzare un *markup* preciso e ben definito: se un documento ha suddivisioni ed identificazioni utilizzabili, allora per un programma è semplice trattare una parte di testo in maniera differente da un'altra ed associare ad esso un significato specifico.

In assenza di un *markup* bisogna avere un mezzo esterno per suddividere i dati in blocchi a cui associare un significato. In un file che contiene dei dati numerici se si è a conoscenza che il primo numero rappresenta il giorno dall'inizio dell'anno, il secondo il numero di minuti dall'inizio del giorno, il terzo la temperatura, il quarto la pressione atmosferica ed il quinto l'umidità si possono elaborare i dati. Se manca questa conoscenza quello che si ha è una serie di cifre incomprensibili. Se i dati sono scritti nel formato *floating point* della macchina che li ha generati diventa decisamente complicato decodificarli e dargli un significato.

Con un *markup* il testo è suddiviso in blocchi ben precisi, che iniziano e terminano in punti ben precisi. È possibile capire a che cosa serve e che cosa rappresenta un dato testo, ad esempio se si tratta di una temperatura o di una lunghezza. Un blocco di testo si trova in una posizione ben precisa, prima e dopo altri blocchi ed inoltre può essere contenuto in un blocco più grande ed a sua volta contenere blocchi più piccoli. Si possono anche avere indicate delle relazioni con altri blocchi di testo oppure altri oggetti mediante dei collegamenti.

Il termine *documento* in XML ha un'accezione più ampia rispetto al significato comune di articolo giornalistico o di libro. Un documento XML è una serie di informazioni strutturate: ad esempio SMIL è un tipo di documento XML che permette di gestire presentazioni multimediali mentre MathML permette di scrivere formule matematiche.

Un documento XML può essere anche utilizzato per codificare informazioni allo scopo di memorizzarle o trasmetterle con un *computer*, anziché essere destinate alla stampa o alla visualizzazione. XML risulta un contenitore molto comodo per facilitare lo scambio di informazioni complesse e di oggetti fra applicazioni diverse.

La definizione di documento nello standard [20] dice che: "Un oggetto è un documento XML se è *well formed* (...). Ogni documento XML ha sia una struttura logica che una fisica. Fisicamente il documento è composto da parti chiamate entità. Un entità può riferirsi ad altre entità per causarne la loro inclusione nel documento. Un documento inizia in una 'radice' o entità documento. Logicamente il documento è composto da dichiarazioni, elementi, commenti, riferimenti ed istruzioni di elaborazione, le quali sono tutte indicate nel documento da un *markup* esplicito. La struttura fisica e quella logica debbono annidarsi correttamente (...)".

Per creare un linguaggio di *markup* utilizzando XML il primo passo è quello di scrivere i documenti seguendo alcune regole su come creare e posizionare i *tag*.

Seguendo le regole si ottiene un documento *well-formed*: se non si aggiungono indicazioni su elementi ed attributi si ha il cosiddetto *freeform* XML. I controlli che si possono fare su documenti XML *freeform* sono piuttosto limitati: gli elementi e gli attributi possono essere qualsiasi ed essere messi in qualunque ordine.

Il passo successivo è quello di creare un modello di documento, andando a specificare quali elementi ed attributi si possano utilizzare ed in quale contesto. Si ha quindi un oggetto che può essere utilizzato per verificare se un'istanza di un documento XML è corretta rispetto alla definizione che è stata data. Il processo di verifica è detto validazione: se un documento è valido significa che non ci sono elementi inventati e che tutti i dati necessari sono presenti nell'ordine giusto. Per definire un modello esistono tre metodi principali.

Il primo, più comune, è utilizzare un *Document Type Definition* (DTD), cioè una serie di regole o dichiarazioni che specificano quali attributi possono essere utilizzati e che cosa possono contenere. All'inizio del documento viene indicato quale DTD dev'essere seguito per la loro creazione.

Il secondo, più recente, è detto "XML Schema" oppure "Xschema": viene utilizzato un *markup* XML per definire un modello di documento. Rispetto ai DTD è possibile specificare in maniera molto precisa i tipi di dato ammissibili in un elemento, cosa assai utile nel caso di applicazioni di codifica dei dati.

Il terzo, RELAX NG è anch'esso un *markup* XML che permette di definire modelli di documento, che utilizza elementi diversi rispetto ad Xschema.

Un linguaggio di *markup* creato con XML è anche detto "applicazione XML"[1] oppure "tipo di documento".

Quando un documento deve essere stampato o visualizzato è normalmente necessario legare i vari attributi a differenti rappresentazioni grafiche cioè si deve passare da un linguaggio di *markup* ad un *markup* tipografico. Le informazioni non stanno all'interno di un documento XML, ma sono normalmente contenute in un'entità separata, detta "foglio di stile". Si può costruire un linguaggio di *markup* che ha al suo interno informazioni stilistiche, come è il caso di HTML. Il problema che si crea è di non poter convertire facilmente in altre forme un documento: una parte in corsivo di un testo può essere sia una parola straniera che una evidenziazione del contenuto. Se si decide successivamente che le parole straniere devono apparire scritte in tondo bisogna modificare a mano tutto il documento. Indicando nel *markup* che invece una parola è straniera, piuttosto che si tratti di un passaggio evidenziato, semplicemente modificando il foglio di stile si può cambiare l'aspetto di quanto viene visualizzato. Con più fogli di stile dallo stesso documento XML si possono ottenere più versioni dello stesso documento. Se si usa un *markup* basato sul ruolo e non sullo stile, diventa anche più facile trasformare un documento XML in un altro documento XML

[1] Da non confondersi con applicazione che utilizza tipi di documento XML, intesa come programma.

che ha un'altra organizzazione od altri elementi definiti.

Un *XML Processor* è un programma che si occupa di leggere un flusso di dati XML suddividendoli in unità adatte ad essere elaborate da altre applicazioni. Esempi di applicazioni che usano i processori XML sono i validatori XML, *web browser*, *editor* XML, sistemi di archiviazione dati, sistemi di controllo ecc...

L'impiego di XML, rispetto all'uso di altri formati di memorizzazione, permette di avere a disposizione in molti linguaggi di programmazione un *parser* già fatto a cui si può accedere mediante una API. Ci sono due tipi principali di processori XML: uno detto normalmente *Document Object Mode* (DOM) crea in memoria un albero che rappresenta la struttura del documento XML, l'altro detto *Simple API for XML* (SAX) genera una serie di eventi ogni volta che nel documento XML incontra un blocco a cui è stata associata un'azione.

Questi *parser* hanno una tolleranza agli errori molto stretta, un comportamento diametralmente opposto rispetto a quello tipico dei *web browser*, che cercano di continuare a visualizzare pagine HTML completamente sbagliate, con elementi inesistenti, annidamento errato dei blocchi e parti obbligatorie mancanti. Anche i *parser* SGML tentano di continuare l'elaborazione dei documenti in caso di errore, anche se non arrivano a quello che fanno i *browser* HTML. Un processore XML invece si ferma nell'elaborazione al primo errore, come un *tag* mal posto, od un elemento scritto in maiuscolo anziché minuscolo: un documento XML per essere elaborato dev'essere sempre *well-formed*.

Quella che sembra una limitazione pesante è in realtà un vantaggio: costringendo ad eliminare gli errori più macroscopici si evita che un eventuale errore di battitura che poteva essere trovato all'inizio delle operazioni di elaborazione venga ignorato causando poi problemi difficili da risolvere da parte dell'applicazione che usa XML. Si rende anche più ripetibile il comportamento di diversi processori XML che altrimenti, utilizzando diverse strategie di *error recovery*, potrebbero comportarsi in maniera diversa nel tentativo di correggere un errore. Del resto l'attitudine a perdonare ed a correggere gli errori delle applicazioni HTML ha portato il proliferare di pagine con una mescolanza illeggibile di *tag* ed attributi, inutilizzabile per null'altro che la visualizzazione in un *browser* specifico.

4.2 Storia

L'utilizzo di forme di *markup* è nata assieme alla gestione informatizzata dei testi, perché, come già accennato, per riuscire ad organizzare un testo è necessario dare ad un programma, oltre al testo stesso, delle indicazioni sugli attributi che le varie parti di testo devono avere. Per esempio i programmi TEX e troff fanno ampio uso di *markup*. Nell'ambito del *word processing* su *personal computer* programmi come Wordstar utilizzavano una forma di *markup* ibrida, parzialmente testuale e

parzialmente binaria[2], mentre WordPerfect utilizza una forma interna di *markup*[3] che può essere resa visibile e modificata mediante il comando "Rivela Codici". Questi sistemi sono stati pensati per la visualizzazione di informazioni su video, oppure per la stampa, per cui le informazioni aggiunte erano per gestire l'aspetto grafico dei diversi elementi testuali.

Nel caso dei *word processor* la stampa era, almeno inizialmente, orientata a stampanti a spaziatura fissa: gli elaboratori di testi, specialmente per i primi *personal computer*, erano pensati per un mercato in cui il sistema tipico di scrittura era la macchina per scrivere e quindi da una parte l'*hardware* normalmente utilizzato aveva capacità piuttosto limitate e dall'altra si cercava di dare agli utilizzatori un sistema simile a quello che erano abituati ad utilizzare. Il paradigma della macchina per scrivere è ben presente adesso, sia nel funzionamento dei programmi di *word processing*, sia nel modo d'uso degli stessi: di nuovo contenuto e presentazione vengono messi insieme e non possono essere separati.

Un formato mirato alla presentazione rende difficile utilizzare i dati per cose che siano diverse dalla presentazione stessa. la soluzione che iniziò ad essere utilizzata fu quella di introdurre una "codifica generica", con attributi descrittivi anziché codici di formattazione: la prima organizzazione che esplorò questa strada fu la *Graphic Communications Association* (GCA) alla fine degli anni '60 con il suo progetto "GenCode". Il passo successivo fu fatto da IBM con il suo progetto *Generalized Markup Language* (GML), creato da Charles Goldfarb, Edward Mosher e Raymond Lorie. Con GML era possibile risolvere il problema di codificare documenti con diversi sistemi informativi: i documenti in GML potevano essere modificati, pubblicati ed indicizzati da programmi differenti grazie all'uso di un *markup* generico: la dimostrazione della bontà dell'idea era data ad esempio dal fatto che IBM ne faceva ampio uso per la gestione dei suoi manuali tecnici.

Successivamente, visti i risultati promettenti, il comitato ANSI sull'*information processing* creò un gruppo di lavoro, con a capo Goldfarb, per creare uno *standard* per la descrizione di testi basato su GML, basandosi anche sull'esperienza maturata con il GenCode della GCA.

Nel 1983 finalmente venne creato il candidato per uno standard industriale (GCA 101-1983) chiamato *Standard Generalized Markup Language* (SGML), che venne rapidamente adottato da diverse entità governative degli Stati Uniti come ad esempio il Dipartimento della difesa e l'IRS. Negli anni successivi SGML iniziò ad essere

[2]In Wordstar venivano date informazioni sulla formattazione della pagina mediante i "comandi punto" ma le informazioni su grassetto e sottolineatura erano invece date con caratteri di controllo ed inoltre veniva usato il *bit* più significativo per indicare i ritorni a capo forzati ed altre caratteristiche del testo.

[3]WordPerfect può leggere e scrivere documenti in formato DocBook SGML mediante l'utilizzo di moduli aggiuntivi, anche se il formato nativo, in cui normalmente salva i suoi documenti è un formato binario.

adottato in ambiti sempre diversi. L'*Association of American Publishers* (AAP) spinse l'uso di SGML per codificare documenti generici come libri e giornali. Infine l'ISO nel 1986 ratificò SGML come un suo *standard* (ISO 8879:1986). In seguito SGML venne adottato da molti produttori di *software* per gli utilizzi e per descrivere i tipi di documento più diversi: da trascrizioni di antichi scritti irlandesi alla documentazione tecnica per gli aerei militari, dagli spartiti musicali alle cartelle mediche. Ma il tipo di documento certamente più utilizzato di SGML è HTML, creato agli inizi degli anni '90 per creare semplici documenti ipertestuali: anzi è l'unico *markup*, insieme in parte a DocBook, ad essere stato utilizzato al di fuori di applicazioni di nicchia.

Sfortunatamente HTML, sia per il fatto di avere un numero limitato di elementi disponibili, sia per utilizzare un solo tipo di documento per tutte le necessità ha portato a creare documenti complessi e ad usare elementi ed attributi per fare cose diverse e slegate logicamente tra loro, così come usare indifferentemente elementi di presentazione e di contenuto per ottenere un certo risultato grafico porta ad avere documenti difficilmente riutilizzabili. Per ovviare a questi problemi si tentò inizialmente di adattare SGML per essere utilizzato nel *web*.

L' SGML è molto potente, ma ha il una elevata complessità: è quindi piuttosto difficile da imparare, ma soprattutto è troppo pesante per essere utilizzato in un *web browser*.

Alla fine del 1996, alla conferenza SGML di Boston venne presentata la prima bozza dello standard XML, cioè una versione semplificata di SGML, che mantenendone le potenzialità, ne toglieva le parti più complesse da gestire e da implementare. Lo standard è stato elaborato dall'*XML Working Group*, originariamente *SGML Editorial Review Board*, diretto da Jon Bosak di Sun Microsystems. I dieci obiettivi di XML, indicati nello standard [20] sono:

1. XML dev'essere utilizzabile direttamente su Internet
2. XML dev'essere utile per un'ampia varietà di applicazioni
3. XML dev'essere compatibile con SGML
4. Dev'essere facile scrivere programmi che elaborano documenti XML
5. Il numero di parti opzionali di XML deve essere tenuto minimo, se possibile zero
6. I documenti XML devono essere leggibili dalle persone e ragionevolmente chiari
7. Il progetto di XML dev'essere preparato rapidamente
8. Il progetto di XML dev'essere formale e coinciso

9. I documenti XML devono essere facili da creare

10. La brevità del *markup* XML è d'importanza minima

Si tratta insomma di un SGML più facile da gestire e da capire. Il documento di specifica di SGML è di oltre 500 pagine, contro la trentina delle specifiche di XML. Evitare le parti opzionali nello standard e non permettere la minimizzazione degli attributi che usa SGML ha permesso di avere degli strumenti di elaborazione di XML decisamente più semplici di quelli di SGML e soprattutto di evitare che sistemi diversi potessero interpretare in maniera diversa lo stesso documento. Mentre l'adozione di SGML era comunque frenata dalla sua complessità, XML ha avuto rapidamente una grande diffusione, e le applicazioni che utilizzano XML sono ormai migliaia.

I *browser* principali, come Microsoft Internet Explorer, Netscape/Mozilla, Opera ed Amaya sono in grado di visualizzare pagine XML. Diversi RDBMS sono in grado di memorizzare pagine XML o di visualizzare tabelle utilizzando XML. Soprattutto diventa facile utilizzare XML per memorizzare i dati che vengono utilizzati dai propri programmi, grazie alle librerie presenti in molti linguaggi di programmazione. Diversi programmi commerciali vengono modificati per utilizzare XML anziché i loro formati interni.

4.3 XML da vicino

Dopo aver esaminato dal punto di vista generare cos'è XML, in questa parte si andrà ad esaminare da vicino come sono fatti i documenti XML. A questo scopo si prenderà come riferimento un semplice documento XML, che descrive il contenuto di un 33 giri.

Esempio 4.1 *Un semplice documento XML*

```
1  <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2  <!DOCTYPE disco
3        SYSTEM "/var/www/pass2/dischi.dtd"
4  [
5     <!ENTITY euro "&#x20ac;">
6  ]>
7  <disco tipo="pop">
8    <supporto stato="ottimo">33 giri<stereo/></supporto>
9    <titolo>Sono solo canzonette</titolo>
10   <cantante>Edoardo Bennato</cantante>
11   <catalogo>SMRL 6279</catalogo>
12   <editore>Dischi Ricordi S.p.A.</editore>
13   <pubblicazione>1980</pubblicazione>
```

```
14    <contenuto>
15      <brano lato="A" numero="1"><titolo>Ma che sarà</titolo>
16      </brano>
17      <brano lato="A" numero="2"><titolo>Il rock di Capitan Uncino</titolo>
18      </brano>
19      <brano lato="A" numero="3"><titolo>Nel covo dei pirati</titolo>
20        <arrangiamento>Eugenio Bennato</arrangiamento>
21      </brano>
22      <brano lato="A" numero="4"><titolo>Dopo il liceo che potevo far</titolo>
23        <arrangiamento>Antonio Sinagra</arrangiamento>
24      </brano>
25      <brano lato="B" numero="1"><titolo>L'isola che non c'è</titolo></brano>
26      <brano lato="B" numero="2"><titolo>Rockcoccodrillo</titolo></brano>
27      <brano lato="B" numero="3"><titolo>Tutti insieme lo denunciam</titolo>
28        <arrangiamento>Antonio Sinagra</arrangiamento>
29        <cantante registro="soprano">Edith Martelli</cantante>
30        <cantante registro="baritono">Orazio Mori</cantante>
31      </brano>
32      <brano lato="B" numero="4"><titolo>Sono solo canzonette</titolo></brano>
33    </contenuto>
34    <note>
35      Edizione con inserito nella copertina del disco
36      un fumetto di Capitan uncino<em>(non quello Disney)</em>.
37      Acquistato usato a 4 &euro;
38    </note>
39  </disco>
40  <!-- fine -->
```

Gli elementi hanno una sintassi simile a quella di HTML, ma hanno dei nomi diversi: HTML è un tipo di documento SGML, mentre XML è una versione semplificata di quest'ultimo che permette di creare tipi di documento con elementi personalizzati: quello che si è utilizzato è un *markup* pensato per un tipo di documento in particolare. I vari tipi di documento XML avranno poi un insieme definito di attributi, specifici per quel tipo di documento.

Una scelta opportuna dei nomi degli attributi permette di capire il tipo di dati che rappresenta ogni singola parte: un attributo del tipo `<tape speed="9.5" duration="40">` è decisamente più comprensibile di `<ort0 sp="9.5" ltm="40">`. XML da solo però non basta per arrivare ad avere un documento finito da stampare o da visualizzare su *web*: del resto un *browser* non sa come trattare questi attributi e quindi che aspetto devono avere e se sono testi da visualizzare o qualcosa d'altro. In HTML il problema non esiste, perché gli attributi e i nomi degli elementi sono fissati, e quindi è per il *browser* è possibile associare ad essi uno stile senza dover fornire altre informazioni aggiuntive. Per visualizzare un documento è quindi necessario utilizzare dei metodi che trasformino un documento XML in un formato più utile: come si spiegherà nella parte 4.6 esistono diversi sistemi che permettono di

convertire e visualizzare i documenti XML che vengono sfruttati dai vari programmi di impaginazione e di visualizzazione sia per una visualizzazione diretta che per generare dati utilizzabili da altri programmi per ottenere l'*output* finale.

Nella parte successiva si analizzeranno le parti di cui si compone un documento XML.

4.3.1 Prologo

All'inizio del documento ci sono alcune informazioni utili per elaborare in maniera corretta il documento stesso.

La prima riga dell'esempio 4.1 contiene la "dichiarazione XML" che permette di capire che si tratta di un documento XML, sia per identificare da sistema operativo di che tipo di *file* si tratti che per un processore XML od SGML capire che i dati sono effettivamente XML e non qualche cosa di simile[4].

I primi cinque caratteri sono "`<?xml` ". Questo permette di rilevare quale sia la codifica dei caratteri e se l'ordine in cui sono memorizzati i caratteri nel sistema di origine è lo stesso di quello del sistema di arrivo e sopratutto che si tratta di XML.

Seguono tre proprietà che possono essere indicate:

`version` Indica il numero di versione: attualmente esiste solo la versione 1.0 di XML, ma se dovesse uscire un futuro una nuova versione, in questo modo è possibile modificare opportunamente il comportamento dei programmi nel caso le future versioni di XML dovessero essere molto differenti dalla versione attuale. Questo parametro è obbligatorio.

`encoding` Indica la codifica dei caratteri del documento, ad esempio `US-ASCII`, `ISO-8859-15`, `Shift_JIS` oppure `UTF-8`. Se si utilizzano codifiche diverse da UTF-8[5] od UTF-16 è obbligatorio utilizzare questa proprietà.

`standalone` Indica al processore XML se ci sono altri dati da recuperare per la corretta elaborazione del documento. Se non ci sono entità esterne o DTD da caricare mettere questa proprietà a `yes` può migliorare le prestazioni del sistema.

Tutti i parametri devono essere minuscoli ed essere compresi fra apici (`'`) o virgolette (`"`).

Le righe successive (2-6) sono la dichiarazione del tipo di documento[6]. La dichiarazione inizia con la stringa "`<!DOCTYPE`", seguita dal nome dell'elemento radice,

[4] In realtà lo standard non richiede la presenza della dichiarazione XML perché un documento sia *well formed*, ma la sua presenza è altamente consigliabile.

[5] Oppure il *set* di caratteri US-ASCII, in quanto sottoinsieme di UTF-8

[6] *Document type declaration*. Da non confondersi con la *Document Type Definition* (DTD), di cui si parlerà più avanti.

cioè quello che contiene tutto il resto del documento (riga 2). Se si utilizza un DTD bisogna indicare l'entità esterna in cui è contenuto in modo che il processore XML riesca a trovarle (riga 3).

Le entità esterne possono essere indicate da un identificatore di sistema oppure un identificatore pubblico. Il primo viene dichiarato utilizzando la parola chiave `SYSTEM` seguita da un URI[7]. Esempi di identificatori di sistema validi sono:

```
SYSTEM "/Documenti/Prove%20XML/prova1.dtd"
SYSTEM "/var/www/pass2/dischi.dtd"
SYSTEM 'http://www.w3.org/TR/REC-html40-971218/loose.dtd'
```

Da notare che i caratteri ammessi in un identificatore devono essere ASCII: nel caso venissero usati caratteri diversi il processore può convertire questi caratteri in sequenze opportune (ad esempio il carattere "`é`" viene convertito in "`%dd%81`", che è la sua codifica UTF-8 trasformata con il sistema di *escape standard* per gli URI). In alternativa è possibile utilizzare un identificatore pubblico, che è dichiarato utilizzando la parola chiave PUBLIC, seguita da un *formal public identifier* (FPI)[8] e da un URI.

Questo permette sia di avere la compatibilità con i sistemi SGML che ad un processore XML di non dover andare a cercare il *file* nel posto indicato dall'URI, ma usare un catalogo per recuperare l'entità esterna, se è ad esempio installata insieme al programma sul *filesystem* locale oppure recuperarla con un URI diverso. Esempi di identificatori pubblici validi sono:

```
PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
       "http://www.textuality.com/boilerplate/OpenHatch.dtd"
PUBLIC '-//W3C//DTD XHTML 1.0 Strict//EN'
       'http://www.w3.org/TR/xhtml1/DTD/xhtml-strict.dtd'
PUBLIC "-//OASIS//DTD DocBook XML V4.2CR1//EN"
       "http://www.oasis-open.org/docbook/xml/4.2CR1/docbookx.dtd"
```

Infine, nelle righe 4, 5 e 6 si trova la dichiarazione del cosiddetto *internal subset*, racchiuso tra parentesi quadre. Qui si possono aggiungere dichiarazioni non presenti nel DTD, cioè definire le entità interne, molto utili per memorizzare piccole parti di testo che si ripetono frequentemente oppure, come nel nostro caso, definire con un nome i caratteri Unicode che si utilizzano frequentemente.

[7] *Uniform Resource Indicator*. Fondamentalmente è l'unione dell'insieme degli URL (*Uniform Resource Locator*) e degli URN (*Uniform Resource Name*). Si ricorda che un URL può anche essere il nome di un *file*, oltre che una risorsa presente in rete.

[8] Definito dallo standard ISO-8879

4.3.2 Gli elementi

Il corpo di un documento XML è costituito da una serie di elementi uno dentro l'altro. Gli elementi sono hanno questa sintassi:

```
<nome attr1="val1" attr2="val2">   ← tag di apertura
 Nome   Attributo    Attributo
  ...
  contenuto dell'elemento          ← character data ed altri elementi
  ...
</nome>                            ← tag di chiusura
  Nome
```

Il *tag* di apertura è fatto dal carattere "<", seguito dal nome dell'attributo, senza spazi in mezzo. Gli spazi possono essere messi invece in qualunque altro posto all'interno del *tag* di apertura, cosa utile per migliorare la leggibilità nel caso vi siano molti attributi. successivamente vengono inseriti i vari attributi, costituiti dal nome dell'attributo, dal segno "=" e dal valore dell'attributo, compreso tra virgolette. Infine il carattere ">" chiude il *tag*. Successivamente c'è il contenuto dell'elemento, che può anche essere vuoto. Segue il *tag* di chiusura, costituito dai caratteri "</", dal nome dell'elemento e dal carattere ">". Il tag di chiusura, al contrario di HTML ed SGML deve essere sempre presente.

Se l'elemento ha un contenuto nullo si può utilizzare la sintassi abbreviata, equivalente a scrivere `<nome attr1="val1" attr2="val2"></nome>`:

```
<nome attr1="val1" attr2="val2" />   ← tag di apertura e chiusura
 Nome   Attributo    Attributo
```

La presenza obbligatoria del *tag* di chiusura permette di semplificare il compito del processore XML, che non ha bisogno di sapere se un elemento può essere annidato, ma può verificare la correttezza sintattica dei documenti senza dover fare ricorso al DTD associato. Si prenda come esempio questo frammento di un documento HTML:

```
<p align="left">
  Questo è un paragrafo.
<p align="center">
  Questo è un altro paragrafo.
  <br>
<ul>
 <li>Primo elemento
 <li>Secondo elemento
</ul>
```

Senza altre informazioni aggiuntive, non è possibile sapere se il secondo elemento "<p>" inizi dopo il primo, o se l'elemento "<ul>" sia all'interno dell'elemento "<br>" od all'elemento "<p>", od invece sia indipendente. Con XML non ci sono questi problemi di interpretazione ed il risultato è che il *parser* di un processore XML è molto più semplice rispetto al suo corrispondente in grado di leggere SGML[9]. Le regole con cui devono essere inseriti i *tag* sono due: il *tag* di chiusura dev'essere successivo a quello di apertura e gli elementi di apertura e di chiusura di un *tag* devono stare entrambi dentro lo stesso elemento di livello superiore. In altri termini una costruzione del tipo:

```
<p align="left">
Esempio<strong>di annidamento<em>sbagliato</strong>
di elementi HTML</em>
<h1>Altro errore di annidamento</h1>!</h1>
```

non è valida (non è neanche valida in HTML, ma molti *browser* la accettano comunque).

Un elemento con lo stesso nome può apparire all'interno di elementi diversi: nel nostro esempio l'elemento "`titolo`" appare sia dentro l'elemento "`disco`" alla linea 7, che dentro al'elemento "`brano`" alle linee 15, 17, 19, 22, 25, 26, 27 e 32. In questo caso il significato dei due elementi è leggermente diverso e l'applicazione può trattare i due elementi in maniera diversa e nel caso di trasformazioni e fogli di stile è possibile rappresentarli diversamente.

Nel caso in cui in un documento XML sia necessario inserire entità provenienti da più tipi di documento, si può ricorrere ai *namespace*. Si possono così distinguere le entità che appartenenti all'uno od all'altro insieme, specie nel caso che ne esistano con lo stesso nome. In questo caso il nome dell'entità diventa del tipo "`namespace:element`", cioè prima del nome dell'elemento va messo il nome del *namespace* seguito da ":". Il nome del *namespace* va dichiarato prima del suo utilizzo con una dichiarazione all'interno di un elemento che inizia con la parola chiave "`xmlns`". Se non viene seguita da un nome, tutti gli elementi annidati verranno considerati automaticamente appartenenti al nuovo *namespace* senza la necessità di indicare il nome.

```
<spazio:pianeta xmlns:spazio="http://www.superspace/XML/decl">
<pianeta xmlns="http://www.superspace/XML/decl">
```

Sia i nomi degli elementi che quelli del *namespace* non devono iniziare con "`xml`" o con altri nomi riservati in XML o linguaggi correlati, come ad esempio "`xsl`".

[9]Di fatto la sintassi di XML è equivalente a quella delle *S-expression* del LISP.

4.3.3 Testo

Tutto ciò che in un documento non è un *tag* è testo, o più precisamente *character data*. Può essere costituito da qualunque carattere appartenente alla codifica di caratteri indicata nel prologo ed è possibile la rappresentazione di caratteri presenti nel *set* Unicode mediante la loro rappresentazione numerica. Alcuni caratteri, come il segno di maggiore e di minore sono utilizzati per indicare l'inizio e la fine di un *tag*. Per evitare rappresentazioni ambigue si possono utilizzare delle *entity reference* predefinite. Ad esempio per inserire il testo "$\omega < \varphi$" utilizzando una codifica del documento iso-8859-1 si deve scrivere "`&#x03c6; < &#x03c4;`", in cui si possono vedere due esempi di *numbered entity* ed una *predefined entity* (`<`). Nell'esempio 4.1 è stata usata una *named entity reference* personalizzata alla riga 37, per scrivere il simbolo "€". Si possono anche definire delle *general entity*, che possono contenere delle stringhe di testo, anziché un singolo carattere, come nell'esempio seguente.

```
<!ENTITY versione "1.03-pre2">
<!ENTITY AC "The &W3C; Advisory Council">
<!ENTITY W3C "WWW Consortium">
```

Un'alternativa per poter inserire caratteri che possono essere interpretati *markup* è di utilizzare le sezioni CDATA. Se in un documento XML all'interno di un blocco di testo viene incontrata la sequenza "`<![CDATA[`", tutto ciò che appare fino a quando non viene incontrata la sequenza "`]]>`" viene passato all'applicazione come *character data*. Si tratta di un metodo alternativo rispetto a quello di utilizzare delle *entity reference*, che può risultare comodo quando ci sono parti di testo con molti caratteri che andrebbero sostituiti, come ad esempio nel caso di listati.

Possono venir dichiarate anche entità esterne, cioè richiami ad altri *file* esterni al documento che vengono inseriti nel documento corrente, con una sintassi simile a quella della dichiarazione di tipo di documento. Le entità esterne possono anche essere dichiarate *unparsed*, nel caso si tratti di riferimenti ad oggetti che non sono XML ma ad esempio immagini che devono essere elaborate esternamente al processore XML.

Gli spazi all'interno del testo di un documento XML vengono sempre passati all'applicazione, al contrario ad esempio di quanto che succede in HTML e deve essere cura dell'applicazione gestirli. I caratteri di fine riga, invece vengono trasformati automaticamente dal processore XML nel carattere ASCII *linefeed*.

L'ordine degli attributi in un elemento è libero, ma non possono esistere più attributi con lo stesso nome all'interno di un elemento. Infine si possono utilizzare due elementi speciali "`id`" ed "`idref`" per avere riferimenti incrociati all'interno di un documento. I valori degli elementi "`id`" devono essere tutti distinti all'interno del documento, mentre i valori dell'attributo "`idref`" devono avere lo stesso valore di un attributo "`id`" esistente.

La scelta di rappresentare un dato come testo o come valore di un attributo è in generale molto dipendente dall'applicazione. La regola normalmente da seguire è quella che un documento dovrebbe rimanere utilizzabile togliendo tutti i *tag* e che se un dato è costituito da più di poche parole oppure è importante l'ordine in cui appare deve essere rappresentato come *character data*. Nel nostro esempio la riga 8 avrebbe potuto contenere le stesse informazioni utilizzando ad esempio una codifica come quella seguente.

```
<supporto stato="ottimo" tipo="33 giri" canali="2" />
```

Alla fine la scelta di una rappresentazione piuttosto che un'altra dipende molto dall'ambito in cui si utilizza un documento XML e dalle abitudini di chi ha definito il DTD.

La riga 40 dell'esempio 4.1 è un commento. In XML i commenti sono simili a quelli che si incontrano in HTML: "`<!--`" seguito dal commento (che può contenete qualunque carattere, esclusa la sequenza "`--`", e terminati dalla sequenza "`-->`". Possono apparire in qualunque posto che non sia all'interno di parti di *markup*.

In un documento XML possono essere inserite le *processing instruction* (PI). Si possono cioè inserire in delle istruzioni destinate alle applicazioni, ma che non fanno parte del contenuto del documento e che vengono passate dal processore XML direttamente all'applicazione. La sintassi di una PI è "`<?nome dati dell'applicazione ?>`". Non esistono limitazioni a quanto contenuto all'interno della PI, salvo che non devono essere presenti sequenze "`?>`".

4.4 XHTML

Una famiglia di linguaggi di *markup* basati su XML a cui bisogna accennare è quella di XHTML. Come si può capire dall'acronimo, si tratta di una serie di linguaggi pensati per i documenti ipertestuali che si evolvono da HTML, cercando di risolvere i problemi che sono stati creati dalla sua enorme penetrazione e l'evoluzione che ne è seguita ed arrivando ad un sistema di pubblicazione ipertestuale flessibile e facilmente utilizzabile anche nelle applicazioni che non prevedono l'uso di *browser* tradizionali.

XHTML 1.0 [21] è una riscrittura di HTML 4.01 come applicazione XML. Gli elementi e gli attributi sono gli stessi che sono definiti nelle tre versioni della raccomandazione W3C per HTML 4.01: *strict*, *transitional* e *frameset*. La versione *strict* è quella che si allontana di più dalle vecchie versioni di HTML: la maggior parte degli elementi di presentazione sono stati deprecati e diventa obbligatorio usare i fogli di stile per la presentazione. La versione *transitional* invece mantiene la possibilità di usare gli elementi di presentazione, mentre la versione *frameset* aggiunge la possibilità di utilizzare i *frame* per visualizzare le pagine.

Con alcuni accorgimenti è possibile scrivere pagine XHTML compatibili con i *browser* già esistenti, ma allo stesso tempo poter utilizzare tutti gli strumenti disponibili per l'elaborazione di documenti XML.

Le differenze tra un documento XHTML 1.0 ed un documento HTML 4.01 sono piuttosto piccole: vista la mancanza in XML di alcune caratteristiche di SGML è necessaria una scrittura più precisa di elementi ed attributi. Ogni elemento deve essere chiuso, e se l'elemento è vuoto (come ad esempio "`<BR>`" od "`<IMG src='immagine.png'>`") deve essere scritto con la sintassi abbreviata, inserendo uno spazio prima della barra per compatibilità con i *browser* più vecchi (quindi si deve scrivere "`<br />`" ed "`<img src="immagine.png" />`".

Gli elementi e gli attributi devono essere scritti in minuscolo ed i valori degli attributi devono essere messi tra apici e devono essere sempre presenti, come è previsto da XML. HTML 4.01 ed XHTML 1.0 sono molto simili: per facilitare la compatibilità fra i due sistemi non è necessario specificare un foglio di stile se il documento XHTML viene visualizzato con un *browser*: ne viene utilizzato uno *standard*, che normalmente è lo stesso che viene utilizzato da HTML.

Chi scrive pagine HTML probabilmente troverà più difficile adattare le vecchie pagine HTML ad HTML 4.01 che passare da questo a XHTML 1.0, specie se come spesso accade sono anche scritte in maniera non corretta. La conversione in XHTML 1.0 si riduce ad aggiungere i *tag* di chiusura opzionali in HTML ed a modificare gli elementi vuoti: per facilitare il compito esistono programmi, ad esempio HTML Tidy[10] che convertono automaticamente un documento da un formato all'altro.

Il percorso che sta seguendo il W3C nello scrivere le nuove raccomandazioni per XHTML è mirato ad avere la modularizzazione del *markup* e la possibilità di poter decidere quali parti di XHTML devono essere utilizzate per la generazione di un documento.

La seconda raccomandazione prodotta dal W3C, XHTML Basic, definisce in gruppo minimale di elementi per poter generare delle pagine *web* in modo da poter essere utilizzato con sistemi che non siano i classici *browser* su PC. La struttura di XHTML Basic è progettata in modo da poter essere estesa seguendo la raccomandazione *Modularization of* XHTML [22]. Questa raccomandazione scompone XHTML 1.0 in una serie di moduli astratti che ne raggruppano le funzionalità e definisce come creare ed implementare mediante DTD questi moduli. Gli obiettivi della raccomandazione sono di raggruppare le parti semanticamente correlate di XHTML, di permettere di creare linguaggi correlati per scopi specifici utilizzando i DTD mantenendo però le parti comuni definite allo stesso modo, di facilitare gli sviluppi futuri permettendo di sostituire alcuni moduli con versioni migliorate senza andare a toccare il resto ed infine di incoraggiare e facilitare il riutilizzo dei moduli in altri *markup*.

[10] `http://www.w3c.org/People/Raggett/tidy/`

XHTML 1.1 [23] è un nuovo tipo di documento che prendendo come base quanto definito in *Modularization of* XHTML, ridefinisce XHTML 1.0 *Strict* utilizzando i moduli XHTML. La compatibilità verso i *browser* HTML inizia ad essere abbandonata: tutte le funzionalità deprecate in HTML 4.01 non sono più presenti ed alcuni elementi presenti in HTML, come ad esempio i *frameset* non sono definiti. Questi ultimi, insieme ad altri moduli aggiuntivi, possono però essere utilizzati tramite l'inserimento dei moduli relativi, definiti in apposite raccomandazioni. Per uniformare la sintassi gli attributi "`lang`" sono stati eliminati in favore di "`xml:lang`", così come l'elemento "`name`" per indicare i collegamenti ipertestuali è stato sostituito con l'attributo "`id`".

XHTML 2.0, attualmente ancora in fase bozza, si discosta completamente da HTML 4.0. Alcuni elementi non esistono più e sono stati sostituiti da nuovi, non è prevista alcuna compatibilità all'indietro: il *browser* deve necessariamente utilizzare XML e fogli di stile per visualizzare i contenuti. Si punta a definire la struttura del documento mentre la presentazione è demandata ai fogli di stile, a rendere più facile scrivere ed utilizzare i documenti XHTML, ad una migliore accessibilità dei documenti ed all'internazionalizzazione, alla riduzione dell'uso dei linguaggi di *scripting* includendo nel *markup* le funzionalità più usate ed alla indipendenza dal dispositivo di uscita.

4.5 DTD e modelli di documento

Precedentemente si è accennato che un documento XML oltre ad essere *well formed* possa essere anche valido. Si può cioè verificare che un documento abbia una semantica corretta per un certo tipo di applicazione e che tutti i dati necessari siano presenti e nell'ordine giusto. Uno dei sistemi per specificare questi è l'uso dei DTD (*Document Type Declaration*) cioè una versione semplificata del DTD di SGML.

Prima di andare ad analizzare la struttura dei modelli di documento è necessario fare una premessa. Un processore XML non deve obbligatoriamente validare i documenti che vengono letti. Il programma a cui passa i dati può, nel caso che i dati estratti dal documento non siano corretti, emettere un messaggio di errore e fermarsi. Non è quindi necessario dover scrivere un DTD per poter scrivere documenti XML ed in alcuni casi, come ad esempio un una fase iniziale di progettazione si può lavorare anche senza.

Utilizzare un modello di documento è vantaggioso sotto diversi punti di vista, e quindi una volta stabilizzata l'applicazione XML è quasi obbligatorio scrivere il modello di documento che la descrive. Per prima cosa si ha un metodo per documentare che cosa accetta il programma come documento XML, permettendo ad altre persone di capire come deve essere strutturato un documento e che dati devono essere presenti al suo interno per avere un contenuto utilizzabile. Con un modello di

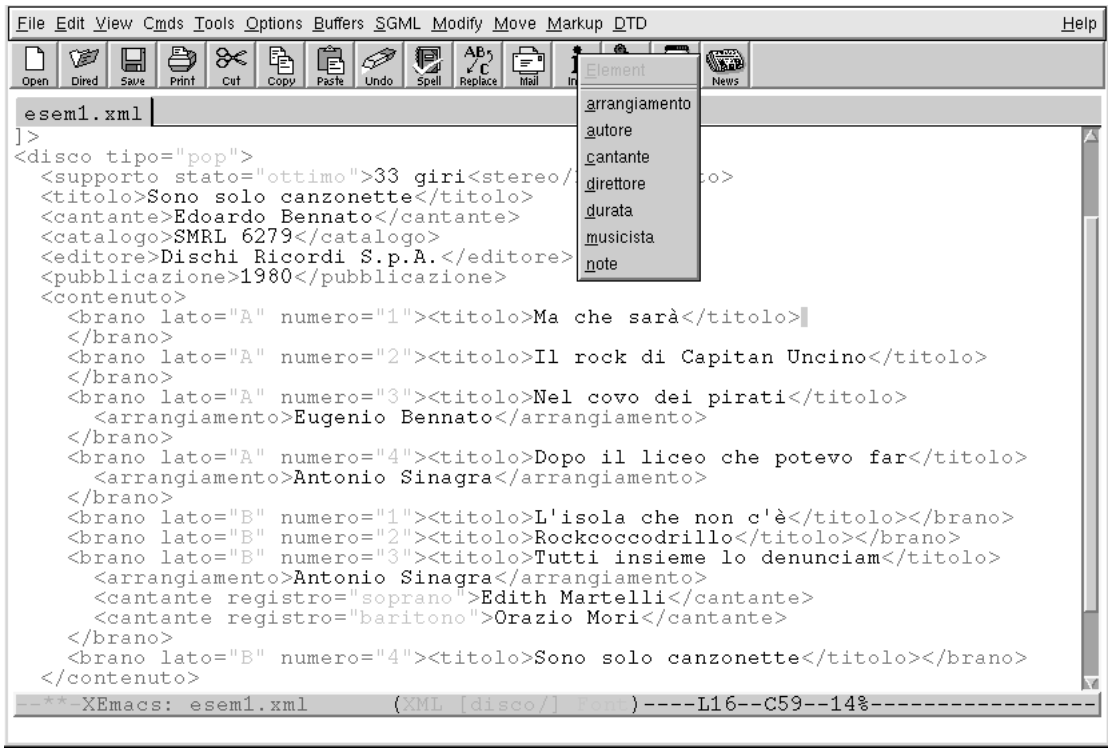


Figura 4.1. Xemacs in modo XML: inserimento facilitato degli elementi

documento a disposizione è possibile per un *editor* che può gestire XML, come ad esempio Emacs (vedi figura 4.1) verificare la correttezza di quanto inserito ed indicare quali elementi ed attributi possano essere inseriti. Se invece XML viene generato da un programma, si può verificare che quanto generato sia corretto e poter capire subito se un problema nasce da un *bug* del programma che crea il documento oppure da un *bug* del programma che lo legge. Infine utilizzando un DTD è possibile definire dei modelli *standard* che tutti possano facilmente interpretare, cosa essenziale per l'interoperabilità e l'uso di applicazioni diverse.

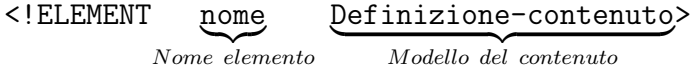
Un DTD definisce un modello di documento dichiarando un insieme di nomi di elementi ammissibili (viene definito il vocabolario del documento), definendo un modello di contenuto per il documento (si indica quali e quanti elementi possono esistere all'interno di altri elementi ed in che ordine) e definendo quali attributi sono ammissibili per un elemento (indicandone il nome, i valori di *default* e se gli attributi sono obbligatori). Per strutturare gestire più facilmente il DTD si possono utilizzare alcuni meccanismi, ad esempio utilizzare le entità parametriche ed importare parti del modello di documento da altri *file*.

4.5.1 Struttura di un DTD

Un DTD è costituito da un prologo opzionale seguito da una serie di dichiarazioni. L'ordine delle dichiarazioni è importante perché in caso dichiarazioni multiple di attributi ed entità solo la prima viene utilizzata, mentre le entità parametriche devono essere dichiarate prima di essere usate e gli elementi possono essere dichiarati una sola volta. Le dichiarazioni presenti possono essere di quattro tipi: *element declaration*, *attribute list declaration*, *notation declaration* ed *entity declaration*.

Element declaration

Per definire e dare un nome agli elementi si utilizza una *element declaration* che ha questa forma:



Il nome è quello che deve essere usato nel *markup* per indicare l'elemento definito e viene fatta distinzione tra maiuscole e minuscole. Possono essere definiti cinque categorie diverse di elementi:

Vuoti L'elemento non può contenere nulla al suo interno, e viene utilizzata la parola chiave `EMPTY`.
Esempio: `<!ELEMENT br EMPTY>`

Senza restrizioni L'elemento può contenere qualunque altra cosa non essendo specificato nessun vincolo: la parola chiave utilizzata è `ANY`. L'utilità di questo tipo di dichiarazione si ha quando si cerca di definire un DTD a partire da un documento XML di esempio: si può iniziare a definire tutti gli elementi così e poi iniziare a definire il contenuto ammissibile di qualche elemento, fino ad arrivare ad una completa definizione.
Esempio: `<!ELEMENT java ANY>`

Solo *character data* Per definire elementi che possono contenere solo testo al loro interno si utilizza come modello di contenuto `(#PCDATA)`. PCDATA è un'abbreviazione di *parsed character data*: il processore XML provvede a sostituire le *entity reference* con il loro valore.
Esempio: `<!ELEMENT riassunto (#PCDATA)>`

Solo altri elementi Gli elementi che possono avere al loro interno solo altri elementi utilizzano una sintassi simile a quella delle espressioni regolari per indicare quali elementi ed in che ordine devono apparire all'interno dell'elemento
Esempio: `<!ELEMENT nodo ((nodo|foglia),(nodo|foglia))>`
Esempio: `<!ELEMENT capitolo (titolo,riassunto?,paragrafo+)>`

Contenuto misto Gli elementi possono contenere sia testo che altri elementi. In questo caso però non è possibile andare a definire l'ordine e l'obbligo della presenza degli elementi all'interno, mentre se un elemento contiene altri elementi è possibile farlo.
Esempio: `<!ELEMENT paragrafo (#PCDATA | strong | em | br)*>`

Attribute list declaration

La dichiarazione della lista degli attributi serve a specificare nome, tipo ed eventuale valore di *default* di ogni attributo associato ad un elemento.

```
<!ATTLIST     nome
              (Nome elemento)
                nomeatt1     tattrib1        descrattr1
              (Nome 1°attributo  Tipo 1°attributo  Comportamento 1°attributo)
                nomeatt2     tattrib2        descrattr2
              (Nome 2°attributo  Tipo 2°attributo  Comportamento 2°attributo)
              ...
>
```

Il nome dell'attributo serve ad identificare l'attributo. Il tipo dell'attributo serve ad identificare che tipo di dati oppure una lista di valori ammissibili per l'attributo.
I tipi di attributo disponibili sono i seguenti:

`CDATA` Questo tipo di attributo ammette qualunque *character data*, comprese *character entity* ed entità interne. Non possono essere presenti parti di *markup* all'interno. Attributi validi di questo tipo possono essere:
`name="ACME &srl;"`
`price="&pound; 12.95"`
`nomeoriginale="&#x72ac;&#x591c;&#x53c9;"`
Esempio: `<!ATTLIST company name CDATA #IMPLIED>`

`NMTOKEN` Questo tipo di attributo contiene un *name token*, cioè una stringa di caratteri composta da lettere, numeri ed i caratteri "`._:-`". Lettere e numeri non sono limitati a quelle presenti nel *set* di caratteri ASCII, ma sono tutti i caratteri Unicode indicati come tali nella definizione di XML. Attributi validi di questo tipo possono essere:
`stazione="Briançon"`
`colloc="p109.r009"`
`file="COMMAND.COM"`
Esempi di attributi non validi sono:
`stazione="Nice Ville"`
`colloc="5/97.003"`

file="C:\DOS\FORMAT.COM"
Esempio: <!ATTLIST libro colloc NMTOKEN #REQUIRED>

NMTOKENS Questo tipo di attributo contiene una serie di *name token* separati da spazi. Attributi validi di questo tipo sono:
telefono="555-1234 555-9988 555-9123"
Esempio: <!ATTLIST impiegato telefono NMTOKENS #IMPLIED>

ID Questo tipo di attributo è dello stesso tipo di un *name token*, ma due elementi non possono avere lo stesso valore: è quindi un *unique identifier*. Il comportamento può essere solo #IMPLIED o #REQUIRED
Esempio: <!ATTLIST nodo numero ID #REQUIRED>

IDREF Questo tipo di attributo è dello stesso tipo di un *name token* ma il suo valore deve essere uguale ad uno dei valori di tipo ID presenti nel documento.
Esempio: <!ATTLIST nodo nodopadre IDREF #REQUIRED>

IDREFS Questo tipo di attributo contiene una serie di valori del tipo IDREF, separati da spazi.
Esempio: <!ATTLIST nodo nodifigli IDREFS #IMPLIED>

ENTITY Questo tipo di attributo contiene una *general entity*. Prima di poterlo utilizzare è necessario dichiarare l'entità nel DTD.
Esempio: <!NOTATION vrml PUBLIC "VRML 2">
<!ENTITY Antarctica SYSTEM "http://www.antarctica.net"
NDATA vrml>
<!ATTLIST World src ENTITY #REQUIRED>

ENTITIES Questo tipo di attributo contiene una serie di *general entity*. Per l'uso si seguono le stesse regole di ENTITY.

NOTATION Questo tipo di attributo serve a dichiarare che quanto contenuto all'interno dell'elemento va trattato come *unparsed entity*, cioè il testo non è XML ma deve essere elaborato da un'altra applicazione, che va indicata mediante l'uso di una dichiarazione NOTATION.
Esempio: <!NOTATION ps PUBLIC "PostScript Level 3">
<ATTLIST Rendered In NOTATION (ps) #IMPLIED>
<!-- uso nel documento -->
<Rendered In='ps'>
gsave 112 75 moveto 112 300 lineto showpage grestore
</Rendered>

Liste di valori In questo tipo di attributo può contenere una serie ben precisa di parole chiave indicate nel DTD, che devono essere messi fra parentesi e separate dal carattere "`|`".
Esempio: `<!ATTLIST data giornosett (lunedì | martedì | mercoledì | giovedì | venerdì | sabato | domenica) #REQUIRED>`

Il comportamento degli attributi può essere di quattro tipi diversi: Un attributo può avere un valore di *default*, essere opzionale, essere obbligatorio oppure essere fisso.

Valori di *default* Se non viene dato il valore dell'attributo all'interno dell'elemento inserito nel documento il processore XML inserisce automaticamente il valore di *default*, che deve essere inserito tra virgolette
Esempio: `<!ATTLIST list type (bullets|ordered|glossary) "ordered">`

Opzionale L'attributo è opzionale ed in questo caso il processore XML non inserisce alcun valore se in un elemento manca l'attributo. Si usa la parola chiave "`#IMPLIED`".
Esempio: `<!ATTLIST terminal name CDATA #IMPLIED>`

Obbligatorio L'attributo deve essere sempre presente negli elementi inseriti nel documento e la sua mancanza è un errore di validità del documento. Si usa la parola chiave "`#REQUIRED`".
Esempio: `<!ATTLIST part id ID #REQUIRED>`

Fisso L'attributo può essere di un solo valore e non può essere cambiato: questo è utile ad esempio quando si modifica un DTD oppure se per qualche motivo un valore di un attributo deve essere lo stesso. Si usa la parola chiave "`#FIXED`" seguita dal valore dell'attributo.
Esempio:
```
<!ATTLIST form method CDATA #FIXED "POST">
<!-- uso nel documento:  il processore XML tratta allo stesso modo i due tag di apertura -->
<form method="POST">
<form>
```

Le dichiarazioni degli elementi di un attributo non devono necessariamente essere messe in un unico blocco, ma le dichiarazioni possono essere divise in più parti, eventualmente sparse su vari *file*. In questo modo diventa facile aggiungere attributi da un DTD di base per ottenere versioni specializzate di alcuni elementi.

Notation

Questa dichiarazione serve ad associare un nome ad una *notation*, cioè l'identificazione del formato o della applicazione che può elaborare una *nonparsed entity*. L'identificatore può essere un tipo MIME, un numero di standard, un URL, il nome di un programma oppure un *formal public identifier*. Questi dati vengono passati all'applicazione che usa il processore XML, che opzionalmente può fornire anche l'identificatore di sistema associato a questi dati. Un esempio di questo tipo di dichiarazione è `<!NOTATION mid SYSTEM "audio/midi">`.

Entity Declaration

Si è già parlato delle entità nel paragrafo 4.3.3. Esistono anche altri tipi di entità che possono essere dichiarate all'interno di un DTD.

External general entity Questo tipo di entità contiene testo proveniente da una fonte esterna, ad esempio un *file* oppure un URL. In questo modo è possibile inserire in un documento XML delle parti *standard* contenute in un *file* esterno in modo che queste parti si possano modificare in un solo punto e non in magari centinaia di *file*.
Esempio: `<!ENTITY open-hatch SYSTEM`
`"http://www.textuality.com/boilerplate/OpenHatch.xml">`
`<!ENTITY open-hatch PUBLIC`
`"-//Textuality//TEXT Standard open-hatch boilerplate//EN"`
`"http://www.textuality.com/boilerplate/OpenHatch.xml">`

External nonparsed entity Questo tipo di entità permette di inserire riferimenti ad oggetti esterni che non sono XML, come ad esempio immagini o suoni. Questa informazione viene passata all'applicazione in modo che essa possa elaborare correttamente i dati od ignorarli.
Esempio: `<!ENTITY bgmusic`
`SYSTEM "../tunes/MID/DontMakeMeWild.mid" NDATA mid>`

Parameter entity Questo tipo di entità sono simili alle *general entity* ma servono per sostituire il testo all'interno del DTD e non all'interno del documento: se per esempio un grande numero di elementi contengono una serie di attributi uguali è possibile scrivere in un solo punto del DTD. La dichiarazione di una *parameter entity* si differenzia da quella di una *general entity* per la presenza del carattere "`%`" dopo la parola chiave "`ENTITY`". I riferimenti all'entità parametrica iniziano con il carattere "`%`" anziché con "`&`".
Esempio: `<!ENTITY % decorations`
`"bullet (circle|cross|dot|none) #IMPLIED`

```
clone (no|yes) #IMPLIED
inline (no|yes) #REQUIRED">
<!-- uso nella dichiarazione di un attributo -->
<!ATTLIST text %decorations;>
```

External parameter entity Questo tipo di entità permette di inserire una entità parametrica da una sorgente esterna. In questo modo è possibile riuscire a comporre dei modelli di documento modulari, in cui esistono parti comuni, che possono essere prese da DTD *standard* e parti personalizzate. In questo modo, insieme all'utilizzo delle sezioni condizionali si riesce ad gestire abbastanza facilmente modelli complessi.
Esempio:
```
<!-- declare the parameter entity "ISOLat2"...  -->
<!ENTITY % ISOLat2
SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ...  now reference it.  -->
%ISOLat2;
```

Un DTD infine può utilizzare le *conditional section*, ovvero delle parti che possono essere inserite nel DTD: si possono avere più sezioni condizionali annidate l'una dentro l'altra.

Le parti comprese tra "`<![INCLUDE[`" ed il corrispondente "`]]>`" vengono inserite nel DTD, mentre quelle comprese tra "`<![IGNORE[`" ed il corrispondente "`]]>`" vengono omesse: se all'interno di una sezione IGNORE esistono sezioni INCLUDE, queste ultime vengono comunque omesse. Con l'utilizzo delle entità parametriche si può facilmente inserire o meno una definizione all'interno di un DTD, come nell'esempio seguente.

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >
<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
 <!ELEMENT book (title, body, supplements?)>
]]>
```

Per concludere si riporta il modello di documento completo che si riferisce all'esempio 4.1. Si noti come lo stesso elemento può essere presente all'interno di più elementi.

Esempio 4.2 *Il DTD che valida l'esempio 4.1*

```
1 <!ELEMENT disco (supporto?,titolo,(autore)*,(cantante)*,
2                  direttore?,(musicista)*,catalogo?,editore,
```

```
3                 pubblicazione,contenuto,note?)>
4  <!ATTLIST disco
5            tipo (pop|jazz|classica|lirica|altro) "altro">
6  <!ELEMENT supporto     (#PCDATA|stereo|mono)*>
7  <!ELEMENT stereo       EMPTY>
8  <!ELEMENT mono         EMPTY>
9  <!ATTLIST supporto
10           stato CDATA #IMPLIED>
11 <!ELEMENT titolo     (#PCDATA)>
12 <!ELEMENT cantante   (#PCDATA)>
13 <!ELEMENT editore    (#PCDATA)>
14 <!ELEMENT direttore  (#PCDATA)>
15 <!ELEMENT musicista  (#PCDATA)>
16 <!ATTLIST musicista
17           strum CDATA #IMPLIED>
18 <!ATTLIST cantante
19           registro CDATA #IMPLIED>
20 <!ELEMENT pubblicazione (#PCDATA)>
21 <!ELEMENT catalogo   (#PCDATA)>
22 <!ELEMENT note       (#PCDATA|em)*>
23 <!ELEMENT autore     (#PCDATA)>
24 <!ELEMENT arrangiamento (#PCDATA)>
25 <!ELEMENT em         (#PCDATA)>
26 <!ELEMENT contenuto  ((brano)*)>
27 <!ELEMENT brano (titolo,durata?,(arrangiamento|cantante|
28                 autore|direttore|musicista|note)*)>
29 <!ATTLIST brano
30           lato CDATA #IMPLIED
31           numero CDATA #REQUIRED>
32 <!ELEMENT durata (#PCDATA)>
33 <!ATTLIST durata
34           formato (ss|mmss) "mmss">
```

4.5.2 XML Schema e RELAX NG

Il sempre maggiore uso di XML per le applicazioni più diverse ha mostrato alcune limitazioni dei DTD per definire alcune categorie di documenti XML.

La sintassi per definire un DTD è piuttosto diversa da quella che ha un documento XML: è una versione semplificata dell'equivalente in SGML: per effettuare delle elaborazioni su un DTD non si possono usare gli stessi strumenti che si usano sui file XML. Con i DTD è difficile gestire i *namespace*, anche perché sono stati introdotti successivamente alla definizione di XML e non esistono in SGML. Utilizzando i DTD risulta piuttosto difficile andare a definire con precisione il formato dei dati ammissibili negli attributi e nel testo di un documento XML: se un attributo deve essere un numero in virgola mobile compreso fra -1 ed 1, piuttosto che una data od una stringa di caratteri di una certa lunghezza, non esiste modo di inserire

l'informazione in un DTD. Se il documento XML è destinato alla pubblicazione è un problema relativo. Nelle applicazioni di trasferimento dati e di accesso alle basi di dati questo problema risulta essere abbastanza importante.

Per superare le limitazioni del DTD sono stati sviluppati altri sistemi per la descrizione dei modelli di documento, che utilizzano un *markup* XML invece della sintassi DTD.

Uno di questi sistemi è definito da una raccomandazione del W3C, XML Schema [25], anche detta Xschema. Un documento Xschema è costituito da un elemento radice, `schema` e da una serie di elementi, i cui più importanti sono `attribute`, `element`, `complexType`, e `simpleType`. I `simpleType` sono delle definizioni di oggetti che non contengono nulla al loro interno e normalmente vengono utilizzati per definire i tipi di dato per il testo e gli attributi e degli elementi. I `complexType` sono invece oggetti che possono contenere altri oggetti al loro interno e nono normalmente utilizzati per definire gli elementi (definiti con `element`), i loro attibuti (definiti con `attribute`) e cosa può essere contenuto al loro interno. Il contenuto di un `complexType` può essere definito direttamente al suo interno (il cosiddetto approccio *matrijoska*), oppure utilizzando l'attributo `ref`, tenendo separate le dichiarazioni di elementi ed attributi, in maniera simile a quello che accade con i DTD. Con questo metodo è anche possibile avere un approccio ad oggetti nella definizione degli elementi, con elementi derivati da un elemento base e simili.

Si può specificare con precisione che tipo di dato può essere presente all'interno di un attributo e del testo di un elemento, e se i quarantuno tipi di dato predefiniti non bastano, se ne possono definire agevolmente di nuovi, ad esempio specificando con *regular expression* come devono essere costruiti i dati.

Un'altro *markup* per definire modelli di documento, proposto da OASIS[11] è RELAX NG [26]. Rispetto ad Xschema ha una struttura più semplice. Per specificare tipo di attributi, tipo di testo ed elementi ammissibili all'interno di un elemento si utilizzano costruzioni molto simili. Non esistono dichiarazioni e definizioni ma solo definizioni di elementi ed attributi. Non esiste un concetto di ereditarietà. Non si possono specificare valori di *default* per gli attributi, né definire entità o *notation*. Un modello di documento scritto con RELAX NG ha come elemento principale `grammar`, al cui interno esiste un elemento `start`, che definisce l'elemento radice del *markup* che viene costruito, e diversi elementi `define` che permettono di definire gli altri elementi del nuovo *markup*. Rispetto ad un DTD un elemento può avere attributi diversi quando si trova all'interno di elementi diversi.

L'esistenza di più modi per poter specificare linguaggi di *markup* basati su XML dimostra come l'utilizzo di XML possa essere flessibile ed adattabile per molte applicazioni diverse. Se il sistema classico di scrivere un DTD si rivela insufficiente

[11]The Organization for the Advancement of Structured Information Standards, `http://www.oasis-open.org/`.

per una situazione specifica, si può utilizzare XML Schema o RELAX NG. Se invece l'uso di XML Schema introduce troppa complessità per un documento semplice di può utilizzare un DTD.

4.6 CSS

Un documento XML permette di rappresentare il contenuto di un documento in forma organizzata e facile da gestire. Se i dati devono essere pubblicati non esiste alcuna informazione su come devono apparire. Un *browser* HTML ha predefinito al suo interno tutta una serie di regole per la presentazione del contenuto di un documento HTML, e quindi non è necessario indicare esplicitamente come deve essere rappresentato un elemento. In HTML esistono anche elementi specifici per la presentazione, per cui è possibile ad esempio indicare il colore e la dimensione di un carattere. Con un *markup* generico questo non è possibile, ed anche con HTML l'uso degli elementi di presentazione può risultare scomodo.

La soluzione è di avere un secondo documento che definisca lo stile di presentazione di un documento XML, ed è qui che entrano in azione i *Cascading Style Sheet* (CSS) [27, 28]. Come si intuisce dal nome l'aspetto più interessante dei CSS è che più stili possono essere applicati a cascata, in modo da avere ampia flessibilità su come debbono apparire i documenti generati. Si può definire degli aspetti di base che si applicano a tutti gli elementi, e poi andare a definire più in dettaglio come ogni singolo elemento deve apparire. Si può anche avere un foglio di stile ed andare a modificare il suo comportamento per una presentazione particolare, caso tipico dell'uso che viene fatto dei CSS nelle pagine HTML in cui si va a modificare uno stile predefinito, eventualmente su più livelli, ad esempio con un foglio di stile globale per il sito, e delle aggiunte specifiche per ogni pagina.

Esistono due versioni di CSS: CSS1 che definisce circa 50 proprietà modificabili, pubblicata nel 1996 e CSS2 che arriva a gestire circa 120 proprietà diverse, pubblicata nel maggio 1998. Il W3C sta lavorando su una nuova versione, CSS3, ma vista la lenta adozione di CSS2 da parte dei maggiori *web browser*, non si prevede che questa raccomandazione venga pubblicata presto.

4.6.1 Uso del CSS

Per associare un CSS ad un documento XML il metodo consigliato è quello di aggiungere una *processing instruction* `xml-stylesheet` all'inizio del *file*. Ad esempio il listato dell'esempio 4.1 va modificato nel modo che si vede più sotto. Un programma può però utilizzare un foglio di stile programmato internamente, oppure utilizzare altri metodi, come per esempio un parametro sulla riga di comando od un menù interattivo.

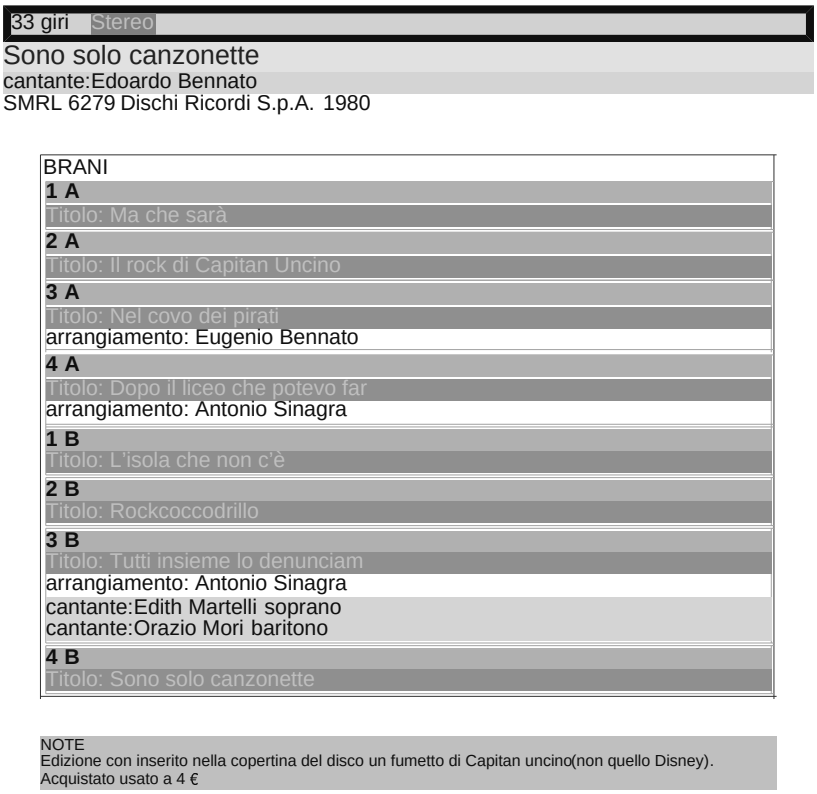


Figura 4.2. Esempio di trasformazione con CSS

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
2 <?xml-stylesheet type="text/css" href="http://127.0.0.1/mike/esem1.css" ?>
3 <!DOCTYPE disco
```

Nel caso di documenti HTML ed XML si usa invece l'elemento `link` per indicare un foglio di stile esterno, e l'elemento `style` per inserire all'interno del *file* le informazioni sullo stile, come nell'esempio che segue. Si possono anche utilizzare gli *header* del protocollo HTTP, nel caso di pagine *web*, o quelli RFC 822 nel caso di messaggi di posta elettronica, ma non tutte le applicazioni sono in grado di gestire correttamente queste intestazioni.

```
<link rel="stylesheet" type="text/css"
      href="/stili/stilebase.css" title="stile base">
<style type="text/css">
body    { font-family: Verdana, Helvetica, sans-serif;
```

```
          color: black;
          background-color: white; }
caption { font-size: 150%;
          font-weight: bold;
          align: center; }
table   { margin: 5%;
          border: 1px solid black;
          text-align: center;
          width: 89%; }
</style>
```

All'interno di un foglio di stile si possono importare altri fogli di stile, utilizzando la direttiva `@import`, oltre ad inserire dei commenti fra "`/*`" e "`*/`", come nell'esempio seguente.

```
/* stili esterni */
@import url(http://127.0.0.1/stili/special1.css);
@import url(http://style.com/basic);
/* modifiche */
pre { border: 2px solid black; }
```

I vari stili vengono dichiarati tramite un selettore seguiti dalle varie proprietà dello stile separate da "`;`" e comprese tra parentesi graffe.

Non è però necessario specificare tutte le proprietà di uno stile per ogni selettore. Le proprietà non dichiarate vengono ereditate dalle proprietà dell'elemento superiore, ed in caso di conflitto esistono delle regole per stabilire quali siano le proprietà valide.

Il selettore serve ad indicare quali elementi devono avere un certo stile, sia semplicemente attraverso il suo nome, sia attraverso la presenza od il valore degli attributi, sia se sono all'interno di altri elementi.

Un CSS adatto ad elaborare l'esempio 4.1 è il seguente, commentato per spiegare i selettori usati. Il risultato grafico che si ottiene con questo CSS è quello della figura 4.2. Altri tipi di selettori sono riportati nell'esempio 4.4, che usa degli elementi HTML.

Esempio 4.3 *Foglio di stile per l'esempio 4.1*

```
1 disco {       display: block;
2         font-family: Verdana, Helvetica, sans-serif;
3         margin: 5px;
4         color: #101010;
5         background-color: white; }
6
7 /* in questo caso tutti gli elementi disco avranno
8 queste caratteristiche */
```

```
9
10 supporto { display: block;
11            border: solid;
12            background-color: #d0d0d0; }
13
14 /* l'elemento supporto acquisisce le proprietà dell'ele-
15 mento superiore (in questo caso disco) e modifica bordo
16 e colore di fondo. */
17
18 stereo:before {        content: "Stereo"; }
19
20 stereo        { display: inline;
21           background-color: gray;
22           color: white;
23           margin-left: 3mm; }
24
25 /* l'elemento stereo modifica le caratteristiche ed aggiunge
26 un testo prima dell'elemento (usando stereo:before) Nel DTD
27 l'elemento stereo è vuoto, ma in questo modo si può fare appa-
28 rire del testo utilizzando l'attributo 'content' */
29
30 brano titolo:before {
31         display: inline;
32         content: "Titolo: "; }
33
34 /* se nel selettore ci sono due o più elementi separati da uno
35 spazio solo gli elementi che seguono la gerarchia usano lo stile
36 indicato */
37
38 disco > titolo { display: block;
39                  color: #202020;
40                  font-size: 125%;
41                  background-color:#e0e0e0; }
42
43 /* se invece esiste il carattere '>' solo gli elementi figli del
44 primo usano questo stile: <disco><titolo>...</titolo></disco>
45 lo usa, <disco><brano><titolo>...</titolo></brano></disco> no.
46 Usando l'esempio precedente in entrambi i casi sarebbe stato
47 usato questo stile */
48
49 brano > titolo { font-size:100%;
50                  display: block;
51                  color: white;
52                  background-color:#909090; }
53
54 contenuto:before { display: block;
55                     font-weight: bolder;
56                   content: "BRANI"; }
57
```

```
contenuto { margin: 5mm;
            display:block;
            border: 1px solid #303030; }

note:before {
        font-style: italic;
        display: block;
        font-weight: bolder;
        content: "NOTE"; }

note {        background-color: #c0c0c0;
        font-style: italic;
        margin: 5mm;
        display:block;
        font-size: 80%; }

cantante:before {
        font-weight: bolder;
        content: "cantante:"; }

cantante { background-color: lightgray;
           display:block; }

cantante:after { font-style: italic;
                 content: " " attr(registro); }

/* in questo caso il testo aggiuntivo viene messo dopo
il contenuto dell'elemento */

arrangiamento:before {
        content: "arrangiamento: "; }
brano:before {
        background-color: #b0b0b0;
        display: block;
        font-weight: bold;
        content: attr(numero) " " attr(lato); }

/* in questo caso con la sintassi attr(attributo) si può
fare apparire il contenuto degli attributi di un elemento */

brano {        display:block;
        margin: 1px;
        border: 1px solid #a0a0a0; };
```

Esempio 4.4 *Altri selettori utilizzabili in CSS2*

```
* { font-style: italic;
    color: black;
    font-family: Bodoni,"New Century Schoolbook","centschbook bt",serif;}
```

```
4
5  /* selettore universale: tutti gli elementi usano questi attributi
6     in questo caso si utilizza un carattere corsivo di colore nero,
7     e come font il sistema cerca uno di quelli disponibili, partendo
8     dal Bodoni e terminando con il serif */
9
10 td[colspan] { font-weight: bold; };
11
12 /* tutte le celle con l'attributo colspn, qualunque sia il valore,
13 appaiono in grassetto */
14
15 [disabled] { background-color: silver ! important; }
16
17 /* tutti gli elementi con attributo disabled presente appaiono
18 con sfondo grigio, ed a questa caratteristica viene dato un peso
19 maggiore */
20
21 form[target="confirm.https"] { background-color: red; }
22
23 /* tutti gli elementi form il cui target è uguale ad confirm.https
24    appaiono con sfondo rosso */
25
26 p.interference { color: lime;
27                  background-color: magenta;
28                  font-family: contdownd, fantasy;
29                  font-size: 50%; }
30
31 /* questo è una abbreviazione per p[class="interference"]
32 in questo caso tutti i parametri con class="interference"
33 avranno scritte verdi su sfondo magenta, con il carattere
34 contownd od in sua assenza un carattere 'fantasy', e grande la metà
35 rispetto alla dimensione normale */
36
37 .bordo { border: 2px solid yellow; }
38
39 /* abbreviazione per [class="bordo"]: qualunqie elemento con
40 classe="bordo" apparirà con un bordo giallo di due pixel */
41
42 ul + li { indent: 3em;}
43
44 /* il primo elemento li che segue un ul verrà indentato di 3 em
45 (cioé l'altezza di un carattere, gli altri no */
46
47 p:first-line { letter-spacing: 1ex; }
48
49 /* la prima riga di un paragrafo appare con le lettere spaziate di
50  1 ex (l'altezza della lettera 'x') */
51
52 p.big * em { word-spacing: 20mm; }
```

```
/* nei paragrafi di classe big, tutti gli elementi em dentro altri
elementi avranno uno spazio extra di 20 mm fra le parole, per es.
<p class="big"><u>attenzione:<em>questo sì</em></u></p> mentre
<p class="big"><em>questo no</em></p> non avrà la spaziatura extra. */

#ref1 { padding: 15pt;
        width: 100%;
        text-align: justify; }

/* l'elemento con ID=ref1 apparirà con un bordo interno di 15 punti,
con una larhezza del 100% e giustificato */
```

Quando ci sono più attributi contrastanti, per decidere quale attributo sia da utilizzare, esistono delle regole ben definite, che permettono di scegliere il comportamento corretto. In generale, più un selettore è specifico per un certo elemento, più ha importanza. L'algoritmo che viene seguito è questo:

1. Cercare tutte le dichiarazioni che si applicano ad un dato elemento. Le dichiarazioni si applicano se il selettore corrisponde all'elemento in questione. Se non si può applicare alcuna dichiarazione, si usa il valore ereditato. Se non esiste un valore ereditato, si utilizza il valore iniziale predefinito.

2. Ordinare le dichiarazioni per importanza: quelle indicate come `! important` pesano di più delle altre.

3. Ordinare le dichiarazioni per provenienza: i fogli di stile del documento pesano di più di quelli utente, che pesano di più dei fogli di stile predefiniti.

4. Ordinare per specificità del selettore: gli attributi ID hanno peso maggiore, seguiti dai selettori di classe e per tipo di attributo, seguiti dai selettori di posizione.

5. Se tutto il resto fallisce la regola specificata per ultima vince.

In questo modo si ottiene una elevata flessibilità nell'utilizzo dei CSS, che risultano, specie per HTML facili da utilizzare.

4.6.2 Limitazioni dei CSS

I fogli di stile sono degli strumenti molto utili e flessibili, in particolare per i documenti HTML ed XHTML. Si può andare a controllare l'aspetto di un sito andando a modificare un solo *file* e non decine di attributi, ed inoltre si riesce a strutturare meglio l'aspetto: il tipo di carattere di un documento per esempio, si può specificare una sola volta.

Esistono alcune limitazioni che rendono i CSS inadatti a produrre certe presentazioni. Gli elementi vengono elaborati nell'ordine in cui appaiono nel documento XML, e quindi non si può ad esempio spostare la posizione di una didascalia e la struttura del risultato è quindi la stessa del documento originale. Infine le regole che permettono di selezionare un elemento sono piuttosto semplici: non si può ad esempio fare apparire in rosso un numero negativo ed in nero uno positivo.

Se si hanno queste esigenze diventa necessario utilizzare altri strumenti. Una delle soluzioni è quella di scrivere un programma che legga un documento XML e generi in *output* un nuovo documento XML, magari con un *markup* diverso, oppure direttamente si occupi di generare i dati necessari ad ottenere una presentazione. L'altra soluzione è quella si andare ad utilizzare XSLT [29] (*Extensible Style Language for Transformation*) per ottenere un nuovo documento XML più adatto per la pubblicazione: una applicazione tipica è quella di andare a generare documenti HTML da un sistema che usa internamente documenti XML. Per i documenti strutturati a pagine è possibile trasformare un documento XML in un documento con un *markup* XSL-FO (*Extensible Stylesheet Language-Formatting Objects*), oltre che in formati più tradizionali come troff o LaTeX.

4.7 Conclusioni

Quanto si è scritto fin qui è solo una minima parte del "mondo" XML. Le applicazioni che utilizzano XML e quindi gli *standard* collegati sono un argomento decisamente vasto e soprattutto in continua evoluzione, grazie agli sforzi del W3C e di altri organizzazioni.

La presentazione non è stata sicuramente esaustiva, limitandosi a trattare degli aspetti fondamentali di XML, ma si può capire facilmente quali siano le potenzialità dello *standard* ma allo stesso tempo di come sia abbastanza facile da usare e da capire.

Capitolo 5

L'implementazione del sistema

In questo capitolo si descriveranno in dettaglio gli aspetti relativi all'implementazione del sito ed alla logica che sta sotto il funzionamento delle varie parti del sistema.

Si spiegherà come il contenuto viene strutturato all'interno del *database* per la sua memorizzazione, si analizzerà il funzionamento del sistema di redazione distribuita e come gli oggetti relativi al contenuto e alla gestione di utenti e flusso di lavoro sono tradotti in entità e relazioni all'interno della base di dati, si spiegherà rapidamente il funzionamento del motore di ricerca ed infine verrà descritta l'interfaccia di amministrazione dal punto di vista degli utenti e verranno descritte le funzioni che vengono svolte dai vari *file* PHP che compongono l'applicazione.

Il metodo utilizzato per costruire quest'applicazione è stato, come si è visto nella sezione 3.6, di creare inizialmente un sistema con funzionalità minime e man mano aggiungere le altre parti necessarie, valutando nel contempo se l'applicazione creata rispondesse ai requisiti richiesti e soprattutto se i requisiti fossero adatti alla situazione organizzativa in cui doveva essere utilizzata l'applicazione.

Nel caso in esame l'uso di un approccio evolutivo, rispetto ad altri sistemi di sviluppo di *software,* risulta vantaggioso per due motivi.

Il primo motivo è che se, come in questo caso, chi richiede il programma non ha le idee molto chiare su cosa abbia effettivamente bisogno, la specifica dettagliata dei requisiti risulta essere un lavoro complesso ed in alcuni casi controproducente: meglio quindi partire da una visione generale del problema e poi avere uno stretto contatto tra sviluppatori ed utilizzatori in modo da poter presentare esempi funzionanti ed avere riscontri che permettano di capire cosa sia effettivamente necessario.

Il secondo motivo è che in questo caso lo sviluppo è stato fatto da una sola persona: l'assenza di documentazione formalizzata non è un grosso problema.

In questa parte ci si occuperà principalmente sull'applicazione attuale e si faranno solamente alcuni rapidi accenni sulle soluzioni esaminate o provate ma successivamente non prese in considerazione.

5.1 Funzionamento del sistema di redazione distribuita

Il sistema di redazione distribuita si basa sul concetto di gruppi, formati da giornalisti e redattori, che creano il contenuto da inserire all'interno delle pagine, e sul concetto di pagine e revisioni.

Come verrà meglio esposto nella sezione 5.3.1, con *pagina* si intende il punto del sito in cui deve apparire una certa informazione: la pagina in sé non ha memorizzato al suo interno nessun contenuto. Con *revisione*[1] si intende il contenuto informativo della pagina: quando un visitatore richiama una data pagina, viene visualizzato il contenuto della revisione *online* associata.

I termini di pagina e di revisione possono risultare un poco fuorvianti e quindi può essere utile fare un paragone con un archivio di documenti cartacei. Si immagini di avere una serie di raccoglitori identificati da un numero di fascicolo progressivo, che contengono al loro interno dei documenti, identificati dal loro numero di protocollo e che sia indicato in qualche modo quale sia il foglio su cui è scritto il documento più aggiornato del fascicolo, ad esempio con una linguetta asportabile od andando a scrivere sul frontespizio del raccoglitore quale sia il numero di protocollo del documento più aggiornato.

Quando si deve recuperare un documento, si prende il raccoglitore identificato dal numero di fascicolo e normalmente si va a vedere qual'è il documento più aggiornato, ma all'interno del raccoglitore sono conservati tutti i documenti precedenti, in modo da poter ricostruire, se necessario, la storia del fascicolo. In questo esempio la pagina corrisponde al raccoglitore e le revisioni corrispondono ai documenti all'interno del raccoglitore.

Il contenuto da pubblicare deve seguire un preciso flusso di lavorazione, che parte dall'inserimento dei testi da parte di un giornalista, prosegue con l'approvazione da parte dei redattori e la pubblicazione del contenuto all'interno del sito, ed infine si conclude con l'archiviazione dei contenuti non più validi. Per coordinare le attività delle varie persone coinvolte esiste un sistema di messaggi che indicano le attività che giornalisti e redattori devono eseguire.

5.1.1 Gruppi, giornalisti e redattori

La gestione del *workflow* del sistema è basata sulla presenza dei gruppi di lavoro (o nodi redazionali). I gruppi di lavoro sono composti da "giornalisti", cioè persone che si occupano di creare ed inserire il contenuto all'interno del CMS. Per ogni

[1] Si è utilizzato il termine "revisione" perché l'uso del termine "contenuto", che sarebbe stato più logico e comprensibile, avrebbe portato ad avere una certa ambiguità: il contenuto è solo una parte dei dati presenti in una revisione.

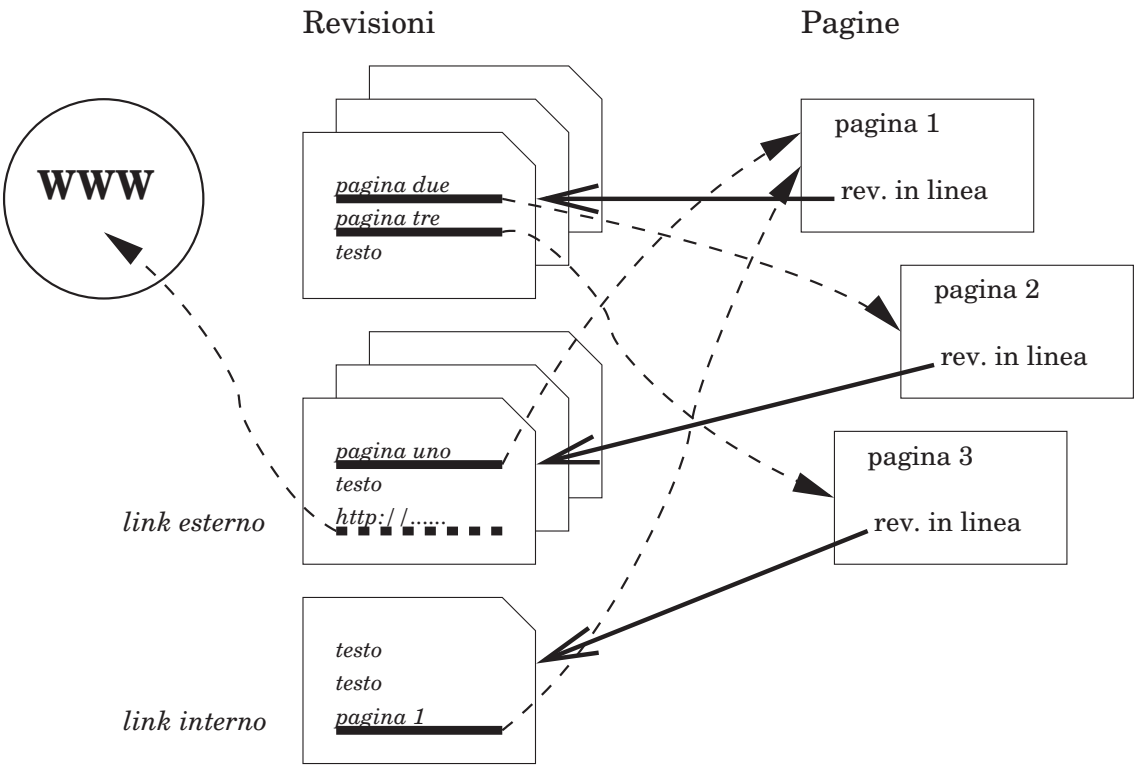


Figura 5.1. Il rapporto tra le pagine e le revisioni.

gruppo di lavoro esiste un "redattore", cioè un giornalista che ha la possibilità di approvare il contenuto creato per la pubblicazione: il redattore può anche assumere il ruolo di giornalista e creare autonomamente del contenuto. Possono esistere gruppi composti da una sola persona che inserisce ed approva da sé i contenuti sotto la sua responsabilità.

I gruppi sono organizzati gerarchicamente in un albero. Esiste un gruppo radice che supervisiona tutto il contenuto del sito, ed una serie di gruppi figli, che hanno in gestione una parte delle pagine del sito e che possono aver ricevuto dal gruppo padre una delega per l'autorizzazione del contenuto, che può essere con fiducia o senza fiducia, come spiegato nel paragrafo 3.4.

Ogni pagina del sito appartiene ad un gruppo: quando un giornalista crea una pagina, viene indicata di proprietà del gruppo di appartenenza del giornalista. Per poter essere pubblicata, una pagina deve essere approvata almeno dal redattore del gruppo proprietario della pagina. Il gruppo di una pagina può essere anche modificato da un redattore tra il proprio gruppo di appartenenza e quelli dei gruppi sottostanti al suo. Un redattore od un giornalista ha visione della parte del sito che

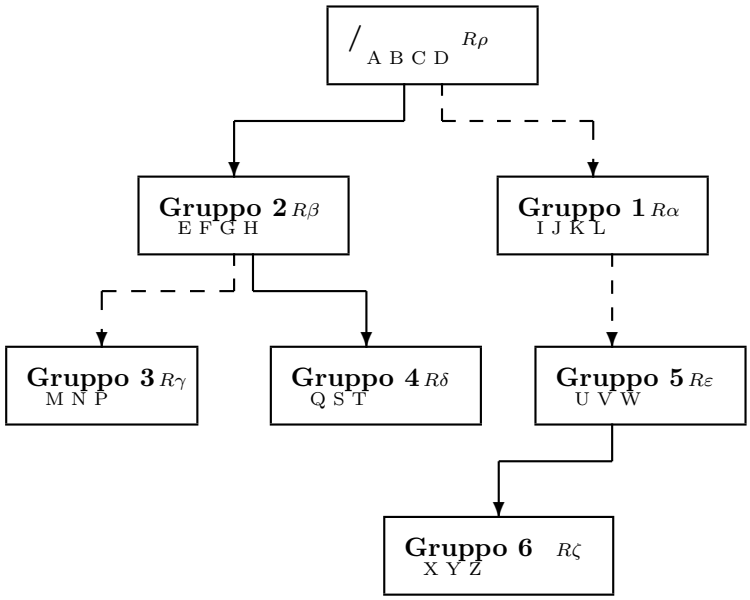


Figura 5.2. Esempio di gerarchia di nodi redazionali (le linee tratteggiate indicano le deleghe senza fiducia, mentre quelle normali indicano le deleghe con fiducia).

è del suo gruppo oppure dei gruppi al di sotto: ad esempio, prendendo come riferimento lo schema di figura 5.2, un giornalista del gruppo 1 può andare a modificare una pagina del gruppo 6, ma non una del gruppo 4 o del gruppo radice.

Questo significa che i redattori ed i giornalisti dei nodi superiori possono modificare tutto il sito, in parallelo con le redazioni dei nodi inferiori. La motivazione è che se una redazione per qualche motivo rimane bloccata, il nodo principale può comunque continuare a modificare tutto il sito.

Un giornalista può anche creare revisioni per pagine di gruppi al di sotto del suo[2], oppure il redattore può cambiare il gruppo a cui appartiene una pagina e riprendersi definitivamente la gestione della pagina.

Le singole revisioni sono invece di proprietà del giornalista che le ha create, e non genericamente del gruppo di lavoro del giornalista. In questo modo si può sempre risalire a chi ha creato un certo contenuto, che come detto può essere anche un giornalista esterno al gruppo.

Possono esistere più revisioni in corso di approvazione od in corso di modifica appartenenti a più giornalisti: solo una delle revisioni naturalmente potrà andare in

[2]L'approvazione viene chiesta al redattore del gruppo del giornalista, in modo da poter operare sul sito anche se un gruppo delegato è fermo. Le notifiche dell'approvazione arrivano anche ai redattori dei gruppi inferiori coinvolti.

linea, ma in questo modo si riesce a gestire il lavoro di creazione del contenuto in maniera più flessibile che con un meccanismo di *check-out/check*-in.

5.1.2 Ciclo di vita del contenuto

In questa parte viene discusso in dettaglio come funziona il sistema di redazione distribuita e di quale sia il flusso di lavoro che porta alla pubblicazione del contenuto. Il contenuto può essere generato sia dai giornalisti che dai redattori: quando un redattore genera del contenuto viene considerato come se fosse un giornalista: un redattore deve comunque approvarsi esplicitamente il contenuto che ha scritto.

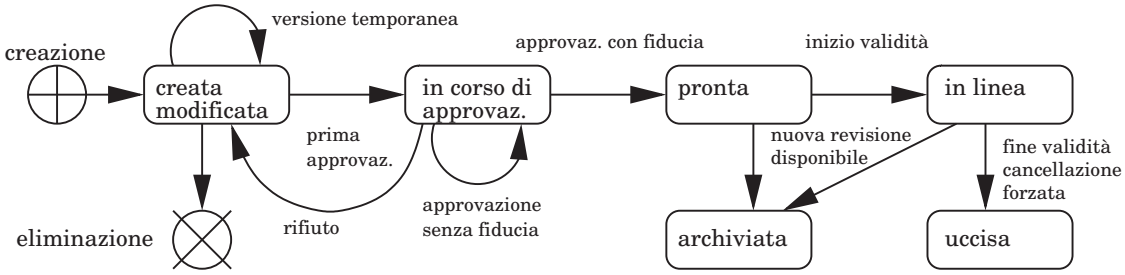


Figura 5.3. Diagramma semplificato degli stati delle revisioni.

1. Quando un giornalista decide di scrivere un nuovo testo, viene creata all'interno del sistema di gestione una nuova pagina ed associata ad essa una revisione, che si trova nello stato di "creata", con all'interno il modello di contenuto scelto.

2. A questo punto è possibile modificare il contenuto della revisione. Al termine della modifica il giornalista ha tre possibilità:

   (a) può salvare il lavoro fatto all'interno del sistema come versione temporanea

   (b) può uscire dall'*editor* senza salvare le modifiche all'interno del sistema di gestione[3]

   (c) può salvare il contenuto e richiedere l'approvazione della revisione.

3. Il giornalista può anche decidere di eliminare la revisione: in questo caso la revisione viene cancellata dal sistema di gestione e non ne rimane traccia. Se la pagina non ha altre revisioni collegate, cosa che normalmente succede

[3]Una volta creata una nuova revisione, questa viene comunque inserita nel sistema di gestione: se non viene salvata alcuna modifica il contenuto della revisione rimane quello del modello.

quando si crea una nuova pagina, anche la pagina viene eliminata dal sistema di gestione[4]

4. Se viene richiesta l'approvazione, inizia l'*iter* di approvazione del contenuto. Il redattore del gruppo a cui appartiene il giornalista controlla la nuova revisione ed ha due possibilità:

   (a) può decidere di rifiutare le modifiche: se non approva il contenuto, viene generato un messaggio di rifiuto al giornalista che può decidere di modificare la revisione oppure di eliminarla (si torna cioè al punto 2)[5]

   (b) può decidere di approvare le modifiche: la revisione passa nello stato di "approvazione in corso", e il giornalista non può modificare ulteriormente il contenuto

5. Se la modifica è approvata ed il gruppo non ha una delega con fiducia, l'approvazione del redattore inferiore provoca una richiesta di approvazione al redattore di livello superiore che di nuovo ha due possibilità:

   (a) può decidere di rifiutare le modifiche: se non approva il contenuto, viene generato un messaggio di rifiuto al giornalista che può decidere di modificare la revisione oppure di eliminarla (si torna cioè al punto 2): la pagina torna nello stato di "creata", in modo che il giornalista possa procedere alla modifica od all'eliminazione

   (b) può decidere di approvare le modifiche: l'*iter* di approvazione prosegue con il redattore di livello superiore al suo

6. Quando si arriva al redattore del nodo radice, di nuovo esistono due possibilità:

   (a) può decidere di rifiutare le modifiche: se non approva il contenuto, come nel caso degli altri redattori, viene generato un messaggio di rifiuto al giornalista che può decidere di modificare la revisione oppure di eliminarla (si torna cioè al punto 2): la pagina torna nello stato di "creata", in modo che il giornalista possa procedere alla modifica od all'eliminazione

   (b) può decidere di approvare le modifiche: in questo caso la pagina si considera definitivamente approvata e passa nello stato di "pronta"

[4] In questo modo se un giornalista crea per errore una nuova pagina, la può eliminare facilmente senza grossi problemi e senza lasciare nel sistema delle pagine inutilizzate e senza alcun contenuto associato.

[5] Esiste anche la possibilità di creare una nuova revisione, appartenente al redattore e non al giornalista, con gli stessi contenuti della revisione in corso di approvazione. L'operazione non elimina la revisione del giornalista che deve essere rifiutata e cancellata dal giornalista.

7. Una volta che la revisione è nello stato di "pronta", non può più essere eliminata dal sistema di gestione: ulteriori modifiche al contenuto provocheranno la creazione di una nuova revisione. Una revisione pronta può essere resa immediatamente disponibile oppure essere messa in linea in un futuro:

   (a) se deve essere immediatamente disponibile passa allo stato "in linea": la pagina eroga il contenuto della revisione ed una eventuale revisione in linea passa nello stato di "archiviata", rimanendo disponibile all'interno del sistema di gestione, ma non più erogabile al pubblico

   (b) se invece deve apparire in un momento nel futuro, rimane nello stato di "pronta", e passerà nello stato "in linea" al momento opportuno, con le stesse modalità del punto precedente

8. Una volta che una revisione è "in linea" può cambiare stato, terminando così il suo ciclo di vita, per tre motivi:

   (a) una nuova revisione viene messa "in linea": la revisione passa nello stato di "archiviata"

   (b) il tempo massimo di pubblicazione della revisione è scaduto, ma non ci sono revisioni disponibili a sostituirla: la revisione passa nello stato di "uccisa" e tentativi di accedere al contenuto della pagina da parte degli utenti daranno come risposta un errore di pagina non trovata

   (c) è stato richiesto forzatamente di mettere fuori linea una pagina: anche in questo caso la revisione passa nello stato di "uccisa"

Quando invece un gruppo ha ricevuto una delega con fiducia, il redattore del gruppo delegante non deve approvare la pagina ma riceve un avviso di avvenuta approvazione: se risalendo nell'albero dei gruppi si arriva al gruppo radice l'approvazione del redattore equivale a quella del redattore del gruppo radice.

Come si è appena visto un giornalista può richiedere la messa fuori linea di una revisione. Il meccanismo di approvazione è simile a quello per l'approvazione di una revisione, salvo il fatto che la revisione viene marcata come "in attesa di cancellazione", rimanendo però comunque nello stato di "pronta" od "in linea", e quindi disponibile all'utenza, fino a quando non si arriva all'approvazione definitiva della cancellazione.

Una revisione, una volta che è stata approvata definitivamente, non può essere più eliminata dalla base di dati: rimane nello stato di "archiviata" od "uccisa", in modo da poter essere eventualmente recuperata. Le revisioni possono essere cancellate dalla base di dati se non sono mai state approvate definitivamente.

Se un giornalista invece decide di modificare una revisione di una pagina già esistente procede con un iter simile a quello della creazione di una pagina nuova.

1. Quando un giornalista decide di modificare il contenuto di una pagina in linea viene creata una nuova revisione, il cui contenuto iniziale può essere di tre tipi:

   (a) può scegliere di partire da un modello, come per una pagina nuova

   (b) può usare il contenuto di una revisione in linea

   (c) può usare il contenuto di una versione archiviata, uccisa, pronta per la pubblicazione od in linea

2. A questo punto è possibile modificare il contenuto della revisione, che si trova nello stato di "modificata"[6]. Al termine della modifica il giornalista ha tre possibilità:

   (a) può salvare il lavoro fatto all'interno del sistema come versione temporanea

   (b) può uscire dall'*editor* senza salvare le modifiche all'interno del sistema di gestione

   (c) può salvare il contenuto e richiedere l'approvazione della revisione.

3. Il giornalista può decidere anche di eliminare la revisione: in questo caso la revisione viene cancellata dal sistema di gestione e non ne rimane traccia.

4. Se richiede l'approvazione della pagina l'*iter* prosegue come per una pagina nuova: la richiesta di approvazione viene fatta al redattore del gruppo a cui il giornalista appartiene e non al gruppo a cui appartiene la pagina.

Un redattore può decidere di modificare una revisione che deve approvare. In questo caso viene creata una nuova revisione, con contenuto uguale a quella da approvare, ma con come proprietario il redattore. La revisione generata dall'altro giornalista rimane memorizzata all'interno del sistema, e quindi deve essere rifiutata dal redattore (e successivamente cancellata dal giornalista).

5.1.3 Ciclo di vita delle pagine

Le pagine rappresentano un sistema per recuperare il contenuto. La cancellazione di una revisione associata ad una pagina, sia che si tratti di una cancellazione del contenuto dalla base di dati, che il passaggio di una revisione nello stato di "archiviata" od "uccisa" non provoca la cancellazione della pagina. Il ciclo di vita di una pagina è più semplice rispetto a quello delle revisioni.

[6]Gli stati di una revisione "creata" e "modificata" sono praticamente equivalenti: l'unica differenza è che una revisione "creata" è generata inizialmente da un modello di contenuto, mentre una versione "modificata" è generata a partire da una revisione esistente.

1. Viene richiesta la creazione di una nuova revisione ed una nuova pagina ad essa associata

2. Inizia il processo di approvazione

   (a) La revisione viene approvata definitivamente, e quindi la pagina non può più essere eliminata dal sistema di gestione

   (b) La revisione viene abbandonata e non esistono più revisioni associate alla pagina: la pagina viene eliminata.

Una pagina viene cancellata se e solo se non ha più revisioni associate ad essa, sia che si tratti di revisioni in corso di approvazione, che revisioni archiviate, che di revisioni in linea. Si può anche andare a forzare la cancellazione incondizionata di una pagina, ma è necessario utilizzare funzioni esterne al sistema normale di gestione del contenuto. Si presume che la cancellazione di pagine già rese disponibili al pubblico sia un evento raro, causato da profonde riorganizzazioni del sito oppure per rimediare ad errori commessi dai redattori.

Poiché non possono esistere revisioni senza una pagina associata, l'eliminazione di una pagina provoca la distruzione di tutte le revisioni ad essa associate.

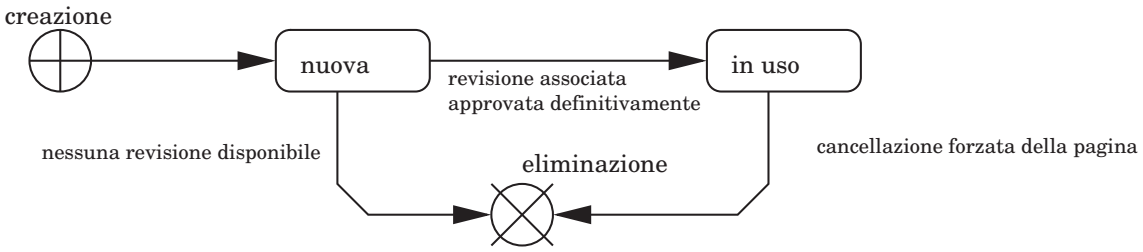


Figura 5.4. Diagramma semplificato degli stati delle pagine.

5.1.4 Gestione dei messaggi

In questo WCMS i redattori ed i giornalisti non sono tutti uguali: appartengono ad un gruppo ben preciso ed i vari gruppi sono in relazione fra loro, e quindi un redattore potrà approvare solo il contenuto generato da alcuni giornalisti. Nel momento in cui un giornalista vuole l'approvazione del contenuto da lui inserito deve chiederlo al redattore del suo gruppo. Una volta che il contenuto è stato approvato da un redattore, deve essere fatta richiesta di approvazione al redattore del gruppo da cui ha ricevuto la delega, oppure nel caso di delega con fiducia il redattore del gruppo delegante deve essere comunque informato delle variazioni avvenute. Per

sincronizzare il flusso di lavoro è stato quindi inserito nel sistema di gestione un meccanismo di generazione e distribuzione di messaggi per giornalisti e redattori.

Per scambiare i messaggi viene utilizzata una tabella che contiene un elenco di attività e di messaggi informativi diretti alle varie persone. La maggior parte dei dati vengono inseriti automaticamente dalle varie funzioni di gestione del sito in seguito alle azioni compiute da redattori e giornalisti, ma è anche possibile creare dei messaggi a mano ed inserire dei commenti ai messaggi automatici. Nel momento in cui un giornalista od un redattore si collega al sistema di gestione delle pagine ed accede alla pagina principale del sistema di gestione delle pagine, avrà nel suo *browser* un elenco dei messaggi, che indicano sia le attività che deve compiere, dei messaggi informativi sullo stato di lavorazione del contenuto e se si tratta di un redattore, delle richieste di approvazione di nuove revisioni.

Se si prende come esempio una struttura redazionale come quella della figura 5.2 (in cui le deleghe con fiducia sono rappresentate da linee continue, e quelle senza fiducia da linee tratteggiate), quando un giornalista del gruppo 6 chiede l'approvazione di una pagina, arriverà un messaggio al redattore del gruppo 6, una volta che questi procede con l'approvazione viene mandato un messaggio informativo al redattore del gruppo 5 (poiché la delega tra 5 e 6 è con fiducia) ed un messaggio con richiesta di approvazione al redattore del gruppo 1, che deve approvarla manualmente. Questo genera un messaggio di richiesta approvazione al redattore del nodo radice. L'approvazione di quest'ultimo manda in linea la pagina. Se invece la pagina è stata creata dal gruppo 4 verranno generati messaggi informativi per il redattore del gruppo 2 e del gruppo radice e la pagina risulta approvata automaticamente.

5.1.5 Esempi di flusso di lavoro

Per maggiore chiarezza si riportano qui di seguito altri possibili scenari su come le pagine e le revisioni vengono gestite dal sistema. In questi esempi i giornalisti, i gruppi ed i redattori sono organizzati seguendo lo schema di figura 5.2. Con $\alpha \ldots \rho$ sono indicati i redattori e con A... Z i giornalisti.

5.1.5.1 Creazione di una nuova pagina

Questo è un flusso di lavoro per la creazione di una nuova pagina in cui vengono richieste delle modifiche da parte dei redattori. Si può notare che le richieste di modifica di una revisione ad un giornalista possono arrivare sia dal suo redattore che da un redattore dei nodi superiori.

1. Z crea una nuova pagina ed inizia a lavorare su una revisione
2. Z salva una versione intermedia della sua revisione e si mette a fare altro

3. Z riprende a lavorare sulla revisione e soddisfatto del lavoro, chiede l'approvazione della revisione
4. ζ riceve una richiesta di approvazione, e rifiuta la revisione con l'indicazione delle modifiche necessarie
5. Z riceve una richiesta di modifica revisione da parte di ζ
6. Z modifica la revisione e ne richiede l'approvazione
7. ζ è soddisfatto ed approva la revisione
8. ε viene notificato dell'approvazione di una nuova revisione, ma non deve fare nulla
9. α riceve una richiesta di approvazione ed approva la revisione
10. Z, ε e ζ ricevono notifica della decisione di α
11. ρ Riceve una richiesta di approvazione, e decide che sono necessarie alcune modifiche
12. Z riceve una richiesta di modifica da parte di ρ
13. Z modifica nuovamente la revisione e richiede l'approvazione
14. ζ approva la revisione
15. ε viene notificato dell'approvazione di una nuova revisione, ma non deve fare nulla
16. α riceve una richiesta di approvazione ed approva la revisione
17. ρ Riceve una richiesta di approvazione, ed approva la revisione
18. La revisione va in linea

5.1.5.2 Nuova revisione di una pagina

Questo è un flusso di lavoro per l'aggiornamento di una pagina, non immediato, ma che dovrà avvenire in un tempo futuro: si noti che il tempo in cui una pagina dovrà apparire è impostato dai giornalisti.

1. Q accede una pagina e crea una nuova revisione che è copia della revisione in linea, impostando un tempo di pubblicazione futuro
2. Q chiede l'approvazione della pagina

3. δ approva la pagina
4. β viene notificato della nuova revisione
5. α viene notificato della nuova revisione
6. La nuova revisione è marcata come pronta ed in attesa di sostituire la revisione in linea
7. Al momento stabilito la nuova revisione sostituisce quella vecchia, che va in archivio

5.1.5.3 Nuova pagina con abbandono

Questo è un flusso di lavoro per la creazione di una pagina, in cui però il giornalista decide di abbandonare la creazione del contenuto prima che venga approvato definitivamente.

1. Q crea una nuova pagina ed inizia a lavorare su una revisione
2. Q salva una versione intermedia della sua revisione e si mette a fare altro
3. Q riprende a lavorare sulla revisione e soddisfatto del lavoro, chiede l'approvazione della revisione
4. Q cambia idea, prima che β abbia visionato la revisione, decide di abbandonare la pagina e la cancella
5. β viene notificato che Q ha cancellato la revisione
6. La revisione viene eliminata dal sistema senza lasciare traccia
7. La pagina creata, non avendo alcuna revisione associata ad essa viene eliminata dal sistema

5.1.5.4 Una revisione viene messa fuori linea

Questo è un esempio di messa fuori linea di una pagina, perché non più raggiungibile. Si noti che il redattore del nodo radice può agire su tutte le pagine del sito per metterle fuori linea.

1. B mette fuori linea una revisione senza che esistano nuove revisioni associate alla pagina, generando una richiesta di approvazione
2. ρ verifica la richiesta e rifiuta la messa fuori linea perché la pagina ha dei collegamenti che puntano ad essa

3. Vengono modificate le pagine con i collegamenti che puntano alla pagina eliminando i *link* in questione
4. B mette di nuovo fuori linea la revisione
5. ρ approva la richiesta
6. La revisione viene messa nello stato di "uccisa" e viene archiviata
7. La pagina rimane senza revisioni associate, ed in caso di accesso da parte degli utenti viene generato un errore di pagina non trovata.

5.1.5.5 Una revisione archiviata viene resa di nuovo accessibile

Un giornalista decide di mettere di nuovo in linea un contenuto che era già stato pubblicato. Viene creata una nuova revisione, con lo stesso contenuto di quella archiviata, che rimane comunque nello stato di "archiviata" od "uccisa". Si noti che a rimettere in linea la pagina può essere un utente diverso da chi aveva fatto la pagina archiviata.

1. E crea una nuova revisione a partire da una pagina archiviata, senza modificarla, e chiede l'approvazione della revisione
2. β approva la nuova revisione (che contiene il contenuto archiviato)
3. La revisione va in linea e ρ viene notificato della variazione

5.2 Il sistema di amministrazione

In questa sezione si esamina come è strutturato dal punto di vista operativo il sistema di gestione del contenuto, di quali sono le operazioni possibili per le varie categorie di utilizzatori e di come è costruita l'interfaccia utente del sistema.

5.2.1 Creazione del contenuto

Quando si accede alla parte del sito che permette la gestione dell'applicazione, appare una schermata principale che permette di entrare nella parte che gestisce le pagine, cioè la parte che permette di inserire e modificare il contenuto oppure nella parte che permette di gestire gli utenti ed i gruppi, oppure ancora di andare alla pagina principale del sito.

Per poter accedere al sistema di gestione del contenuto un giornalista deve per prima cosa autenticarsi, fornendo nome dell'utente e *password*. Se l'autenticazione ha successo viene presentata una pagina informativa, con l'indicazione del nome

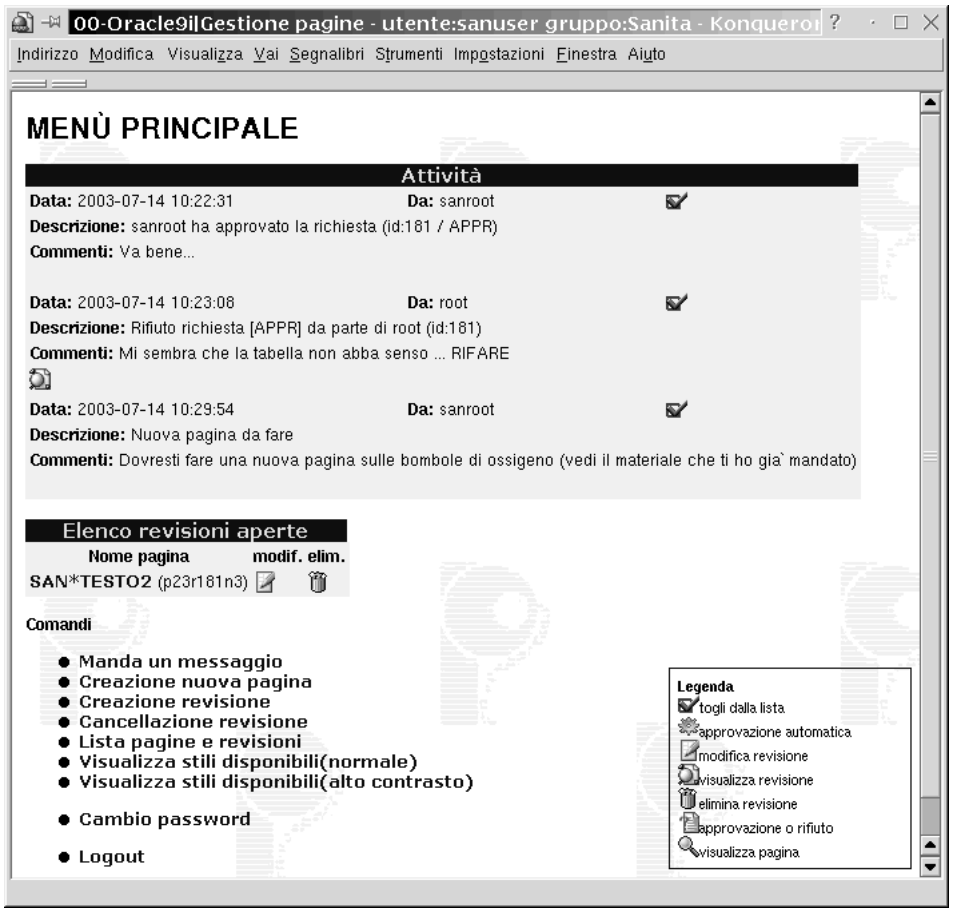


Figura 5.5. Menù principale

dell'utente, del gruppo di appartenenza, dello stato di redattore e della data dell'ultimo collegamento. con un collegamento al menù principale. I messaggi informativi appaiono in verde su sfondo nero, mentre i messaggi di errore appaiono in rosso su sfondo nero[7].

Una volta arrivati al menù principale (vedi figura 5.5) si ha una schermata che si può dividere in tre parti: l'elenco delle attività da fare, l'elenco delle revisioni aperte e l'elenco dei comandi a disposizione. In basso a destra è presente una legenda con la spiegazione delle icone presente nella pagina. Il titolo della pagina indica utente e gruppo collegato.

L'elenco delle attività è composto da blocchi. Nel primo blocco sono indicati la data del messaggio e l'utente di provenienza, sia nel caso che sia stato generato manualmente, che nel caso sia stato generato automaticamente: in quest'ultimo caso

[7]L'aspetto di questi messaggi è controllato da un CSS, e si può quindi modificare facilmente.

appare l'utente che ha richiesto l'approvazione, rifiutato od approvato il contenuto. Nel secondo blocco è indicata una descrizione del messaggio, che può essere generato automaticamente: nel caso dei redattori che hanno al di sotto gruppi con fiducia ai redattori appare anche un'icona che indica l'approvazione automatica. Nel terzo blocco sono presenti i commenti al messaggio eventualmente scritti da chi ha eseguito l'approvazione od il rifiuto, oppure da chi ha generato manualmente il messaggio.

Per ogni attività è presente un'icona che permette di cancellare il messaggio, quando è stata eseguita l'attività collegata oppure se si è letto il messaggio informativo. Possono essere presenti delle icone che permettono di vedere il contenuto in corso di lavorazione, e nel caso si tratti di un redattore che deve approvare il contenuto, due icone che permettono rispettivamente di creare una nuova revisione a partire da quella in corso di approvazione, oppure di accedere alla maschera di approvazione o rifiuto del contenuto. Nella figura 5.5 il primo messaggio è l'approvazione da parte del redattore del gruppo, mentre il secondo è il rifiuto da parte del redattore del nodo principale. Il terzo messaggio è stato invece generato manualmente dal redattore del gruppo.

L'elenco delle revisioni aperte segnala le revisioni su cui un giornalista sta lavorando e che può modificare: possono esserci revisioni di cui non è stata ancora richiesta l'approvazione, revisioni di cui è stata richiesta l'approvazione, ma il redattore del gruppo non ha ancora esaminato ed infine revisioni che nel corso dell'*iter* di approvazione sono state rifiutate e che quindi il giornalista deve modificare. Ogni riga della tabella ha un collegamento ipertestuale con il nome della pagina a cui la revisione appartiene: seguendo il collegamento viene aperta una nuova finestra con il *preview* del contenuto. Alla fine ci sono le operazioni possibili sulla revisione che quindi può essere modificata oppure cancellata, indicate da due icone. La modifica fa entrare nell'*editor* del contenuto, mentre la cancellazione fa apparire una schermata informativa che conferma l'avvenuta eliminazione del contenuto.

Successivamente all'elenco delle revisioni aperte appare l'elenco dei comandi disponibili per la gestione del contenuto e per alcune operazioni di servizio, come il cambiamento delle *password* e l'uscita dal sistema di amministrazione. I comandi disponibili sono descritti nelle sezioni successive.

5.2.1.1 Manda un messaggio

Con questo comando si accede ad una maschera che permette di inviare manualmente un messaggio ad un altro utente del sistema di gestione del contenuto. Appare una maschera con un elenco degli utenti del sistema, un campo per inserire la descrizione del messaggio, ed un'area di testo per inserire i commenti. Si può anche associare una pagina al messaggio. Questo sistema è pensato per inviare dei brevi commenti agli utilizzatori del sistema senza dover utilizzare altre applicazioni e non vuole naturalmente sostituire altri sistemi di comunicazione come ad esempio la posta

elettronica, ma può risultare utile per uno scambio rapido di informazione senza dover attivare altri programmi ed uscire dal *browser*.

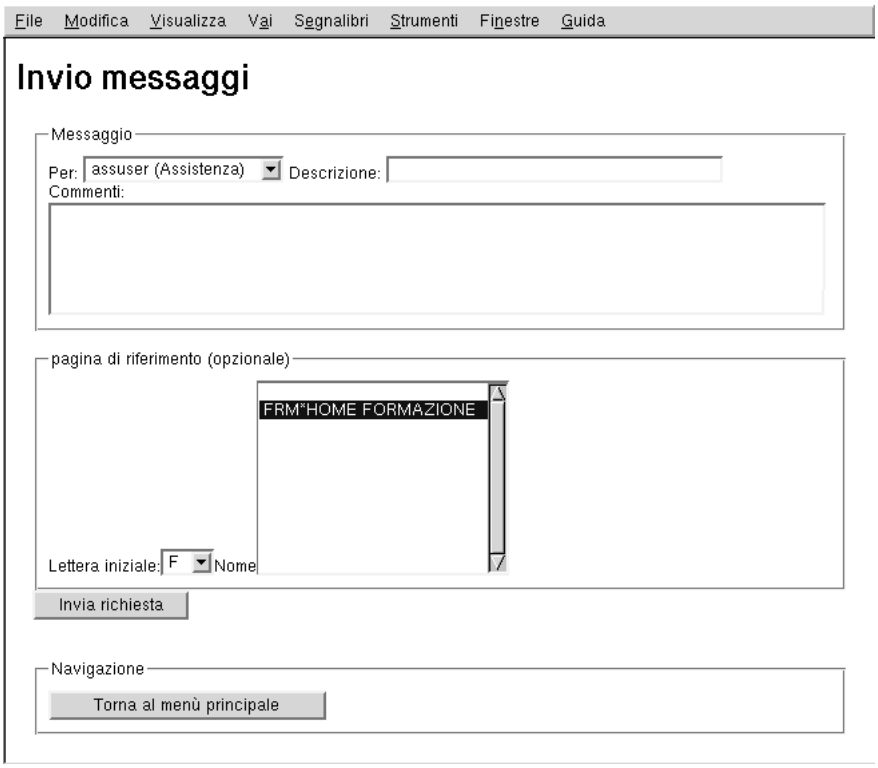


Figura 5.6. Maschera per mandare messaggi

5.2.1.2 Creazione nuova pagina

Con questo comando si può creare una nuova pagina all'interno del sistema di gestione del contenuto e la prima revisione associata ad essa. Appare una maschera in cui si chiede di dare il nome della pagina e da che tipo di modello si deve partire.

Il nome della pagina è composto da una sequenza iniziale di lettere o numeri, da un asterisco e dal nome della pagina che viene utilizzato sia all'interno del sistema di gestione che nella visualizzazione della barra di navigazione. Non è il titolo della pagina HTML generata, che invece si trova all'interno del documento XML.

Premendo il pulsante "crea la pagina", la nuova pagina viene creata ed appare una schermata informativa che informa dell'esecuzione dell'operazione. Se l'operazione non ha successo appare un messaggio informativo che indica quali sono le cause dell'insuccesso. Premendo il pulsante "Modifica la revisione" si può accedere all'*editor*

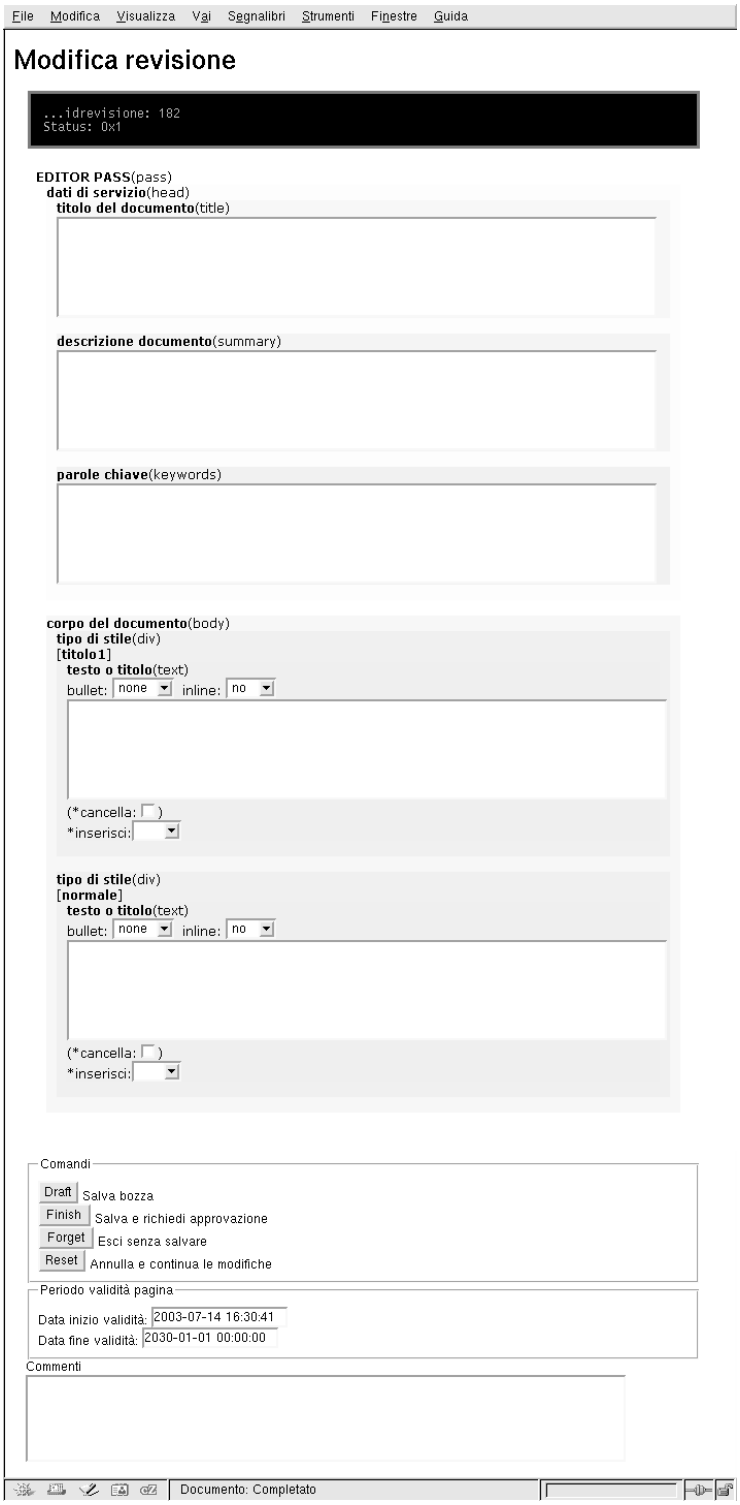


Figura 5.7. *Editor* delle pagine

delle revisioni (figura 5.7). La prima parte dell'*editor* visualizza alcune informazioni di servizio. Segue la parte che permette di inserire e modificare il contenuto.

I vari elementi che compongono il documento XML sono indicati da blocchi con sfondo colorato e da una intestazione che descrive brevemente il tipo di blocco ed indica il suo nome. Seguono i vari attributi degli elementi, con indicato il loro nome ed un campo di testo, oppure una lista di valori che possono essere inseriti nell'elemento. Se l'elemento permette di avere del testo al suo interno, appare anche una *textarea* per l'immissione. Se l'elemento è clonabile appare anche una casella che permette la cancellazione dell'elemento ed una selezione che permette di inserire un nuovo elemento vuoto.

La gestione degli elementi di collegamento ipertestuale è leggermente diversa da quella degli altri elementi: mentre per inserire un *link* esterno bisogna selezionare l'attributo "type" ad "external" ed inserire l'URL della pagina a cui deve puntare il collegamento all'interno di una casella di testo, per i *link* interni si deve selezionare prima la parte iniziale del nome nella pagina tramite un elenco, in modo da fare apparire un secondo elenco delle pagine corrispondenti. Selezionando da qui il nome della pagina, verrà inserito automaticamente il numero della pagina selezionata nella casella di testo, come si può vedere nella figura 5.8.

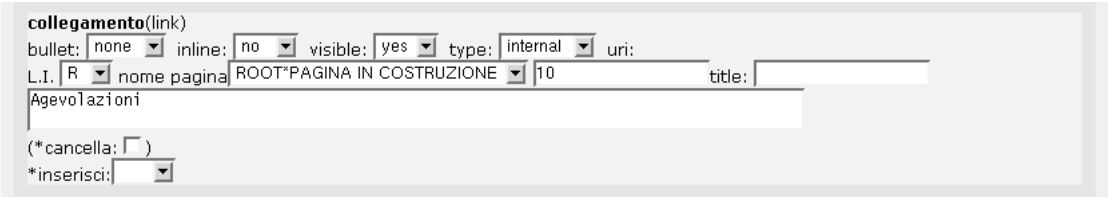


Figura 5.8. Inserimento di collegamenti ipertestuali

Dopo la parte di inserimento e modifica del contenuto ci sono quattro pulsanti che permettono rispettivamente di salvare il documento come bozza, salvare il documento e richiederne l'approvazione, uscire senza salvare ed infine annullare il lavoro fatto e continuare le modifiche. La parte successiva permette di inserire la data e l'ora in cui la revisione deve andare in linea od essere messa fuori linea: se non vengono modificate vengono automaticamente riempite rispettivamente con la data e l'ora corrente e la mezzanotte del primo gennaio 2030[8]. L'ultima parte della maschera contiene un campo per inserire eventuali commenti destinati al redattore

[8]Questa data è stata scelta per risolvere alcune incompatibilità fra Oracle e PHP nella gestione delle date. Oracle permette di gestire date future molto più in avanti, e quindi una volta che verranno risolte questi problemi sarà sufficiente aggiornare opportunamente le date presenti nella tabella ed il valore di *default* per non avere problemi di date.

al momento della richiesta di approvazione. Questi commenti non vengono salvati assieme alle bozze.

5.2.1.3 Creazione revisione

Con questo comando si può creare una nuova revisione di una pagina: si può partire da un modello vuoto, oppure da una revisione esistente. Il primo passo è quello di selezionare una pagina tra quelle disponibili all'interno del sistema di gestione del contenuto. Viene presentata una maschera con due elenchi. Il primo elenco contiene la prima parte del nome della pagina e selezionandone uno dei valori nel secondo appare l'elenco delle pagine corrispondenti, da cui si seleziona la pagina su cui si vuole lavorare.

Le pagine che appaiono qui non sono tutte quelle presenti all'interno del sistema di gestione, ma solo quelle su cui un utente può lavorare. I giornalisti possono lavorare solo sulle pagine del proprio gruppo, mentre i redattori possono operare anche sulle pagine appartenenti ai gruppi al di sotto del proprio. In questo caso, comunque la pagina rimane di proprietà del gruppo originale e non diventa di proprietà del gruppo superiore. Una volta scelta la pagina appare una maschera che permette di scegliere il contenuto che deve avere la nuova revisione. La maschera (vedi figura 5.9) è divisa in due parti. La parte superiore permette di copiare il contenuto di una vecchia revisione in una nuova revisione se il proprietario della revisione è diverso dall'utente che ne ha fatto richiesta oppure se la revisione è stata già approvata, oltre a poter visualizzare il contenuto della revisione.

Per scegliere la revisione su cui lavorare si deve selezionare un *radio button* all'interno della tabella che mostra l'elenco delle revisioni collegate alla pagina selezionata, il loro stato e se si tratta di revisioni aperte o meno. Se la revisione deve essere ancor approvata non viene creata una nuova revisione, ma invece si va a lavorare sulla revisione aperta. La parte inferiore, invece permette di scegliere un modello vuoto da cui partire, in maniera simile a quanto accade per una nuova pagina.

Una volta creata la revisione, si entra nell'*editor* in modo da poter modificare il contenuto della revisione, con le stesse modalità previste per la creazione di una nuova pagina. Si noti che se viene selezionata una revisione esistente non appare la schermata che conferma la creazione della revisione, ma si entra direttamente nell'*editor*, che ovviamente è lo stesso che si usa in caso di creazione di nuova pagina e con le stesse opzioni.

5.2.1.4 Cancellazione revisione

Questo comando permette di generare una richiesta per mettere fuori linea una pagina disponibile, senza che sia presente una nuova revisione da sostituire. Dopo aver scelto la pagina, in maniere simile a quanto si deve fare quando si crea una

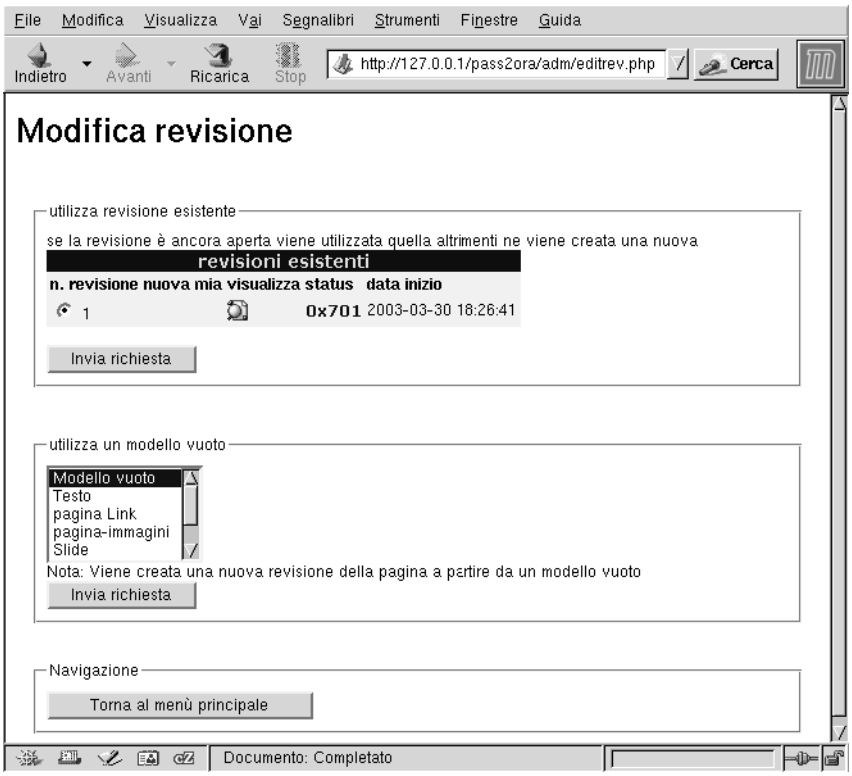


Figura 5.9. Maschera per la creazione della revisione

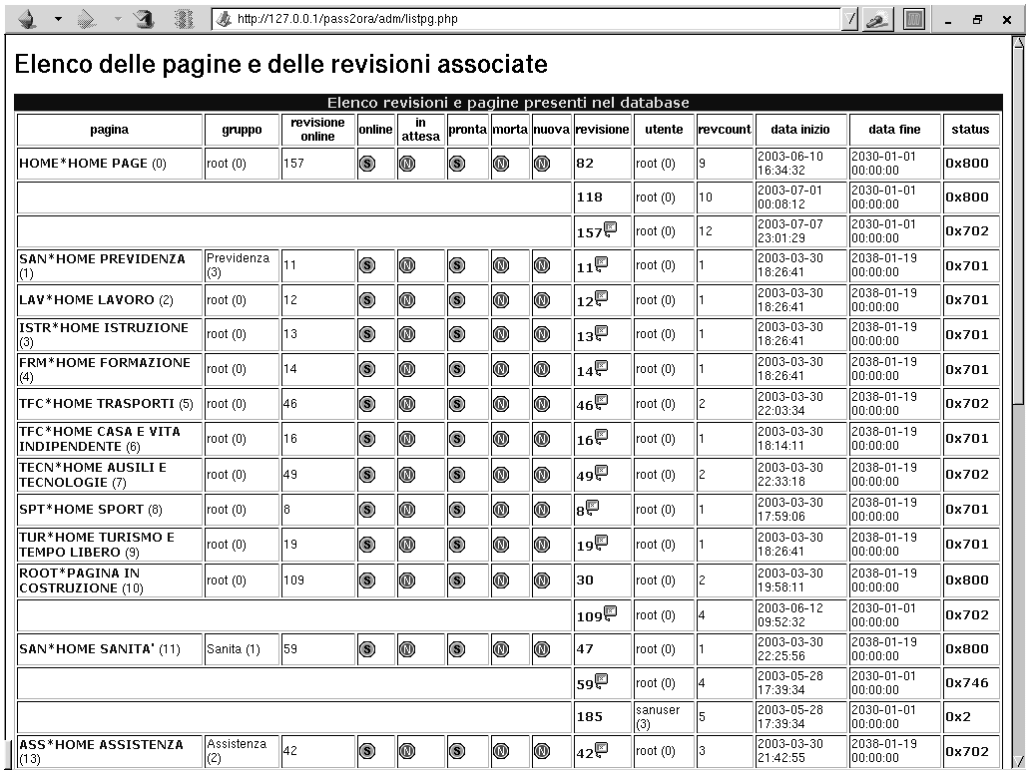
nuova revisione, viene presentata una tabella delle revisioni in linea che possono essere cancellate, in maniera simile a quella vista per la creazione di una revisione a partire dal contenuto di una revisione esistente.

5.2.1.5 Lista pagine e revisioni

Con questo comando è possibile visualizzare una tabella che permette di vedere quali sono le pagine esistenti e le revisioni ad esse associate ed il loro stato, oltre a permettere di visualizzare direttamente le pagine e le revisioni. In questo modo si può avere una visione completa del contenuto presente all'interno del sistema e verificare che non esistano situazioni che possono generare problemi.

Come si può vedere dalla figura 5.10 per ogni pagina sono indicati il gruppo a cui appartiene, il numero di revisione in linea e lo stato in cui si trova, ad esempio se una pagina è nuova, oppure se ha del contenuto in linea, se ha del contenuto approvato definitivamente o se la revisione in linea è stata mandata fuori linea senza che ci fossero altre revisioni disponibili. Per le revisioni associate viene indicato chi ha inserito o modificato il contenuto, la data di inizio validità, la data di fine validità

ed il codice numerico che indica lo stato della revisione. Questa tabella è simile a quella per l'eliminazione incondizionata di pagine e revisioni (figura 5.13), ma senza i collegamenti per la cancellazione.



pagina	gruppo	revisione online	online	in attesa	pronta	morta	nuova	revisione	utente	revcount	data inizio	data fine	status
HOME*HOME PAGE (0)	root (0)	157	S	N	S	N	N	82	root (0)	9	2003-06-10 16:34:32	2030-01-01 00:00:00	0x800
								118	root (0)	10	2003-07-01 00:08:12	2030-01-01 00:00:00	0x800
								157	root (0)	12	2003-07-07 23:01:29	2030-01-01 00:00:00	0x702
SAN*HOME PREVIDENZA (1)	Previdenza (3)	11	S	N	S	N	N	11	root (0)	1	2003-03-30 18:26:41	2038-01-19 00:00:00	0x701
LAV*HOME LAVORO (2)	root (0)	12	S	N	S	N	N	12	root (0)	1	2003-03-30 18:26:41	2038-01-19 00:00:00	0x701
ISTR*HOME ISTRUZIONE (3)	root (0)	13	S	N	S	N	N	13	root (0)	1	2003-03-30 18:26:41	2038-01-19 00:00:00	0x701
FRM*HOME FORMAZIONE (4)	root (0)	14	S	N	S	N	N	14	root (0)	1	2003-03-30 18:26:41	2038-01-19 00:00:00	0x701
TFC*HOME TRASPORTI (5)	root (0)	46	S	N	S	N	N	46	root (0)	2	2003-03-30 22:03:34	2038-01-19 00:00:00	0x702
TFC*HOME CASA E VITA INDIPENDENTE (6)	root (0)	16	S	N	S	N	N	16	root (0)	1	2003-03-30 18:14:11	2038-01-19 00:00:00	0x701
TECN*HOME AUSILI E TECNOLOGIE (7)	root (0)	49	S	N	S	N	N	49	root (0)	2	2003-03-30 22:33:18	2038-01-19 00:00:00	0x702
SPT*HOME SPORT (8)	root (0)	8	S	N	S	N	N	8	root (0)	1	2003-03-30 17:59:06	2038-01-19 00:00:00	0x701
TUR*HOME TURISMO E TEMPO LIBERO (9)	root (0)	19	S	N	S	N	N	19	root (0)	1	2003-03-30 18:26:41	2038-01-19 00:00:00	0x701
ROOT*PAGINA IN COSTRUZIONE (10)	root (0)	109	S	N	S	N	N	30	root (0)	2	2003-03-30 19:58:11	2038-01-19 00:00:00	0x800
								109	root (0)	4	2003-06-12 09:52:32	2030-01-01 00:00:00	0x702
SAN*HOME SANITA' (11)	Sanita (1)	59	S	N	S	N	N	47	root (0)	1	2003-03-30 22:25:56	2038-01-19 00:00:00	0x800
								59	root (0)	4	2003-05-28 17:39:34	2030-01-01 00:00:00	0x746
								185	sanuser (3)	5	2003-05-28 17:39:34	2030-01-01 00:00:00	0x2
ASS*HOME ASSISTENZA (13)	Assistenza (2)	42	S	N	S	N	N	42	root (0)	3	2003-03-30 21:42:55	2038-01-19 00:00:00	0x702

Figura 5.10. Lista pagine e revisioni

5.2.1.6 Visualizza stili disponibili (normale) – Visualizza stili disponibili (alto contrasto)

Questi due comandi fanno apparire una pagina che fa vedere i vari stili disponibili per i vari blocchi di testo e per le tabelle, sia nella versione normale che nella versione ad alto contrasto.

5.2.1.7 Cambio password

Con questo comando permette di accedere ad una pagina che permette al singolo utente di cambiare la propria *password*, dopo aver inserito per conferma la *password* corrente.

5.2.1.8 Logout

Con questo comando si esce dal sistema gi gestione pagine. Dopo essere usciti appare un elenco di opzione che permette di effettuare un nuovo *login* oppure di tornare alla schermata principale del sistema di gestione.

5.2.2 Controllo delle pagine

La differenza principale tra un giornalista ed un redattore è che quest'ultimo può approvare le pagine per la pubblicazione: nella lista delle attività appariranno delle richieste di approvazione e sarà disponibile un'icona che permette di accedere la maschera per l'approvazione od il rifiuto del contenuto. Oltre a questo i redattori hanno anche a disposizione due comandi aggiuntivi, rispetto ai giornalisti, che permettono di modificare il gruppo a cui appartiene una pagina, oppure di cambiarne il nome.

5.2.2.1 Modifica gruppo di una pagina

Questo comando permette di cambiare il gruppo a cui appartiene una pagina, scegliendo tra il proprio gruppo e tutti quelli al di sotto di esso. Selezionando questo comando appare una maschera che permette di scegliere una lista con tutte le pagine sui cui si può operare ed una lista dei gruppi che sono sotto il controllo del redattore.

Si può quindi sia assegnare una pagina del proprio gruppo ad un gruppo delegato, in modo che i giornalisti ed i redattori dell'altro gruppo possano modificare il contenuto associato alla pagina oppure riprendere in gestione una pagina precedentemente di proprietà di un gruppo sottostante, o cambiare la proprietà di una pagina da un gruppo ad un altro.

5.2.2.2 Modifica nome di una pagina

Questo comando permette di modificare il nome di una pagina. Selezionando questo comando appare una maschera che permette di scegliere una lista con tutte le pagine sui cui si può operare, cioè quelle del proprio gruppo e tutti quelli al di sotto e una casella di testo in cui inserire il nuovo nome.

5.2.3 Creazione e modifica dei modelli

Il redattore capo del nodo principale, oltre a tutti i comandi disponibili può anche definire nuovi modelli di pagina. Selezionando il comando "Editor Modelli", appare una maschera con l'elenco dei modelli disponibili all'interno del sistema. Al di sotto dell'elenco appaiono le operazioni possibili sul modello selezionato. Si può cancellare il modello, oppure modificarlo, od ancora creare un nuovo modello a

partire da quello selezionato. Una volta selezionata la modifica oppure la creazione di un nuovo modello, si entra nell'*editor* dei modelli.

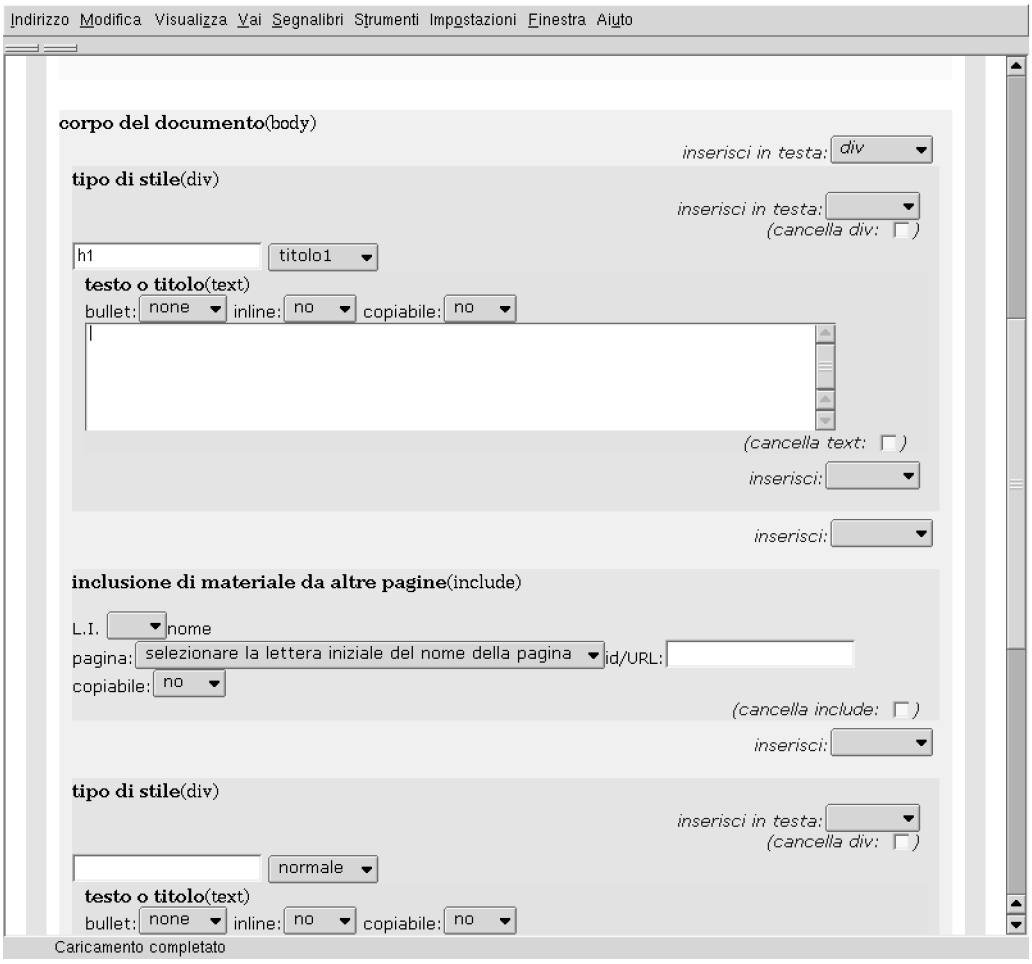


Figura 5.11. *Editor* dei modelli

L'*editor* dei modelli è molto simile all'*editor* delle revisioni, come è possibile vedere nella figura 5.11: nell'implementazione attuale viene utilizzato lo stesso codice per la generazione di entrambe le maschere. Questa scelta deriva logicamente dal fatto che modelli e revisioni hanno come contenuto lo stesso tipo di documento XML.

Le differenza tra una modalità e l'altra sono dovute principalmente allo scopo diverso delle due categorie di documento, come si illustrerà della sezione 5.3.2. Le revisioni servono ad inserire del contenuto da visualizzare, mentre i modelli servono a costruire dei documenti di base per poter creare delle pagine di un certo tipo.

Rispetto all'*editor* delle revisioni, è possibile inserire e cancellare tutti gli elementi, limitati solo dal fatto di ottenere un documento XML valido. Inoltre si possono

modificare tutti gli attributi degli elementi quindi sia il tipo di stile di un elemento `div` che la possibilità di clonare l'elemento nell'*editor* delle revisioni. Per il resto, la parte di inserimento e modifica del contenuto è praticamente uguale: si possono quindi inserire nei modelli testi, collegamenti ipertestuali od immagini in modo da avere modelli parzialmente pronti.

La parte che modifica l'elemento `div` ha una lista degli stili possibili e modifica contemporaneamente gli attributi `style` ed `altstyle`: quest'ultimo viene visualizzato in una casella di testo a sola lettura, mentre per `style` si ha la lista dei valori possibili e scegliendo un valore viene modificato in maniera opportuna anche `altstyle.`

L'altra differenza che si ha è dovuta al fatto che un modello non deve essere autorizzato e diventa immediatamente utilizzabile. Non ci sono i quattro pulsanti presenti nell'*editor* delle revisioni, ma solo un pulsante che permette la memorizzazione ed uno che cancella le modifiche effettuate. Quando le modifiche sono state memorizzate anziché andare direttamente alla pagina principale, si arriva ad una pagina che permette di continuare la modifica sul modello, oppure tornare al menù principale. In questo modo in caso di più modifiche successive, necessarie per inserire una serie di nuovi elementi, si riesce a lavorare più rapidamente che non passando tutte le volte dalla maschera di selezione del modello.

Per quanto riguarda la modifica dei CSS e quindi alla definizione degli stili utilizzabili per l'erogazione dei documenti. Dal sistema di gestione del contenuto non è possibile modificare via *web* il CSS, ma bisogna modificare i *file* presenti sul *web server* che definiscono i CSS e gli stili disponibili. I motivi della scelta sono due. Il primo è che in questo modo si possono utilizzare strumenti per l'*editing* grafico delle pagine HTML per definire i CSS: alcuni di questi prodotti permetto di definire facilmente nuovi stili grafici senza dover scrivere a mano il *file* che definisce il foglio di stile. Il secondo motivo è stato quello di voler evitare la proliferazione degli stili, separando in maniera netta l'impostazione grafica del sito e la gestione del contenuto. Visto il tipo di *audience* del sito, oltretutto risulta importante avere una grafica essenziale e coerente e quindi avere un unico punto in cui si decide la grafica e non permettere modifiche aiuta ad ottenere questi risultati.

5.2.4 Amministrazione degli utenti e dei gruppi

Per inserire, eliminare o modificare gli utenti, per inserire, eliminare o modificare lo stato dei gruppi e per procedere all'eliminazione di pagine o revisioni immediatamente saltando le normali procedure esiste una parte del sistema di gestione non accessibile a redattori ed a giornalisti, in cui può operare l'amministratore del sito. L'amministratore del sito è quindi la persona che si occupa di impostare il

funzionamento del sito, ma non di inserire o di approvare il contenuto[9].

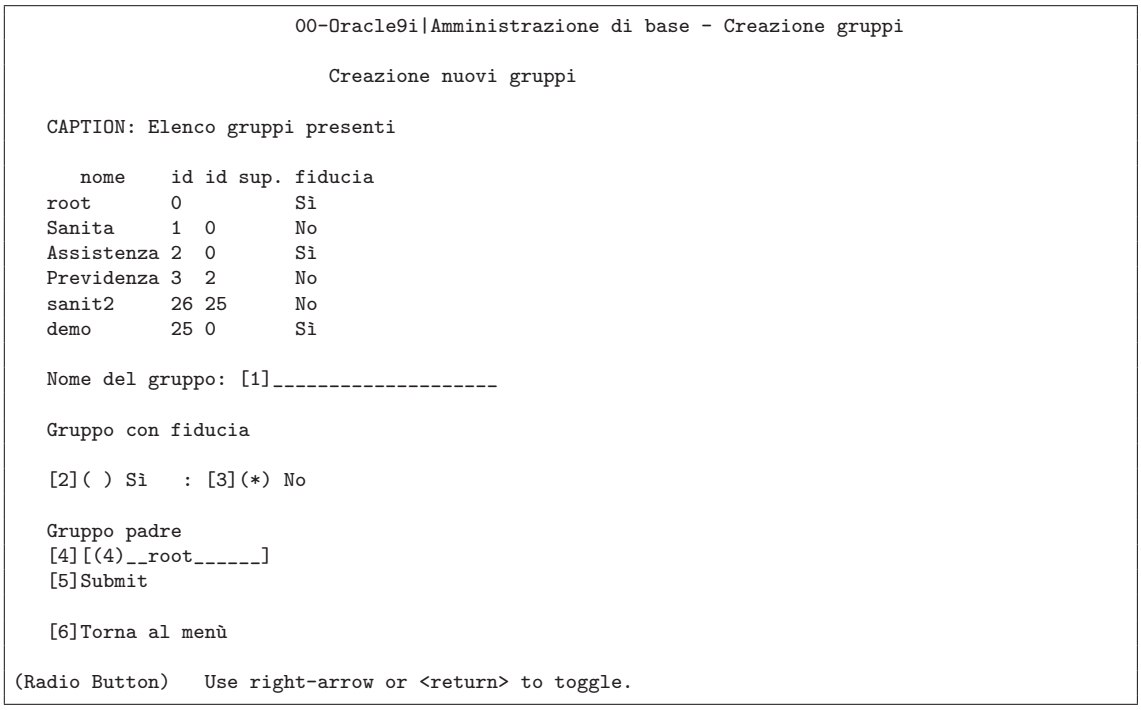


Figura 5.12. Maschera di inserimento nuovi gruppi in Lynx

Il metodo di autenticazione per accedere come amministratore è diverso rispetto a quello per l'autenticazione come giornalista o redattore: viene utilizzata una *directory* sul *web server* protetta da *password*, e viene quindi utilizzata la sicurezza del *web server*[10]. Le ragioni tecniche della scelta sono che se il meccanismo di autenticazione per i giornalisti ha malfunzionamenti od errori di configurazione, è sempre possibile accedere a queste funzioni e rendere di nuovo utilizzabile il sistema di gestione del contenuto. In questo modo si riescono a separare in maniera netta le funzioni di amministrazione, che sono operazioni che vengono effettuate raramente, dal lavoro di redazione che viene fatto tutti i giorni.

Le operazioni possibili sono nove.

Inserimento nuovi gruppi Viene presentata una tabella con l'elenco dei gruppi, se hanno delega con fiducia o meno e chi è il gruppo delegante. Per inserire un

[9] L'amministratore del sito può essere anche fisicamente un redattore od un giornalista, ma per accedere alle diverse funzioni deve accedere in modo diverso.

[10] è possibile configurare il *server* per permettere l'accesso alle pagine mediante una connessione SSL per maggiore sicurezza.

gruppo è necessario dare un nome, al gruppo, indicare il tipo di delega e chi è il gruppo delegante (vedi figura 5.12)

Modifica stato dei gruppi Permette di modificare lo stato di un gruppo, cioè qual'è il gruppo delegante e se la delega è con fiducia o meno. La maschera di inserimento è simile a quella precedente, salvo il fatto che il campo per l'inserimento del nome del gruppo è sostituito da un elenco dei gruppi esistenti nel sistema. Una modifica dello stato dei gruppi è un'operazione che può causare problemi di funzionamento al sistema se eseguita quando il gruppo da modificare od i suoi gruppi figli hanno revisioni in corso di approvazione.

Eliminazione gruppi Permette si eliminare il gruppo selezionato da una lista. Il gruppo non deve avere né pagine né utenti associati ad esso perché l'operazione abbia successo. In caso contrario l'operazione fallisce perché viola l'integrità referenziale del *database*.

Inserimento nuovi utenti Viene presentata una maschera per inserire nome dell'utente, *password*, e gruppo di appartenenza. La *password* deve essere di almeno cinque caratteri.

Abilitazione/disabilitazione utenti Permette di impedire o meno l'accesso di un utente al sistema di gestione del contenuto. Viene presentato un elenco degli utenti, con il gruppo d'appartenenza, e la scelta se abilitare o meno l'utente. Per bloccare l'accesso di un utente è consigliabile utilizzare questa opzione, anzichè eliminare l'utente. Quando un utente viene eliminato non è possibile recuperarlo, anche ricreando un utente con lo stesso nome, che sarà in ogni caso un nuovo utente.

Eliminazione utenti Permette di eliminare un utente dal sistema in maniera definitiva. Viene presentato un elenco degli utenti, con il gruppo d'appartenenza. Una volta eliminato l'utente le sue revisioni diverranno di proprietà del redattore capo del nodo principale.

Cambio password Permette di cambiare la *password* di utente, selezionando il nome da un elenco ed inserendo la nuova *password*.

Modifica stato di redattore Permette di cambiare lo stato di redattore di un giornalista. Viene presentata un elenco dei nomi degli utenti e se l'utente deve essere redattore o meno. Deve esistere un redattore per ogni gruppo, anche se all'interno del gruppo c'è un solo giornalista. Se non ci sono redattori definiti, o se sono più di uno, il sistema di gestione del contenuto può presentare delle anomalie ed impedire l'aggiornamento delle pagine.

Eliminazione incondizionata pagine Permette di eliminare una pagina e le revisioni ad essa associate, oppure di eliminare o mandare *online* una revisione fuori linea, saltando tutto il flusso di lavoro. Viene presentata una tabella con l'elenco di tutte le pagine e di tutte le revisioni presenti nel sistema, con indicazioni di stato ed i comandi per il ripristino delle revisioni (vedi figura 5.13). Questa opzione va utilizzata solo in caso di reale necessità, per rimettere a posto il sito in caso di problemi di funzionamento 0o per grosse riorganizzazioni. Si noti che mettere *online* una revisione a tempo scaduta o che si trova nello stato di "pronta" può non dare l'effetto sperato.

Come appena detto, alcune funzionalità presenti nel sistema di amministrazione possono causare perdita di dati o disfunzioni gravi del sistema di gestione del contenuto. Le operazioni più rischiose sono indicate da un simbolo di avvertimento nel menù di scelta.

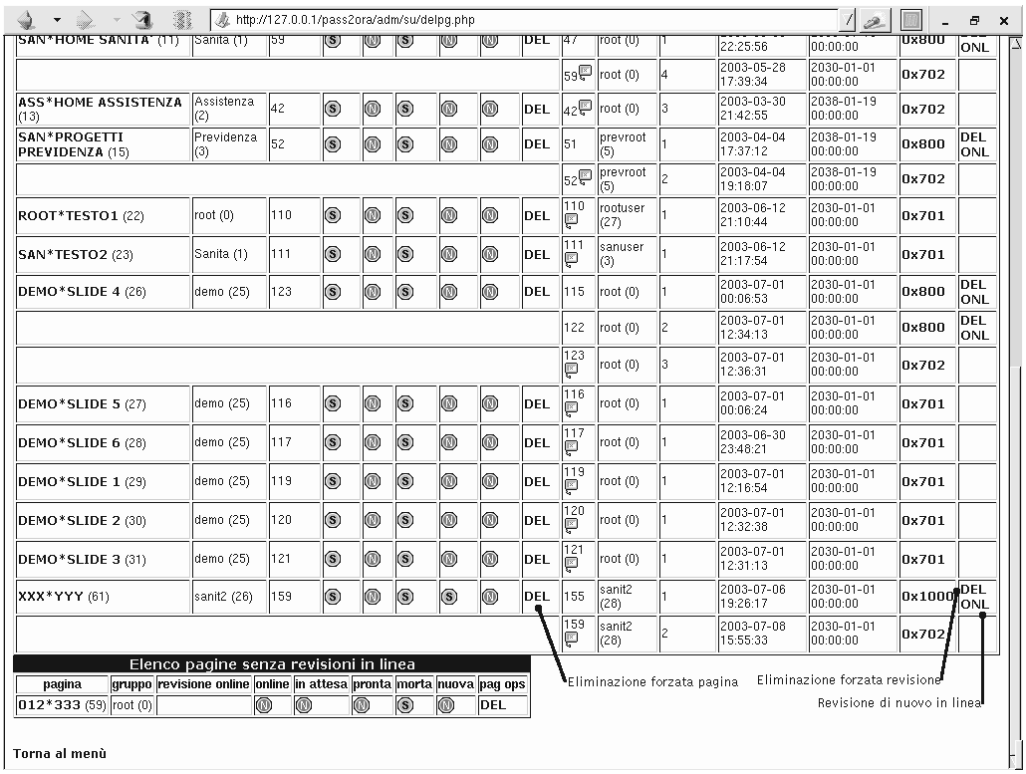


Figura 5.13. Maschera per l'eliminazione incondizionata delle pagine

5.3 Formato di memorizzazione del contenuto

Per memorizzare il contenuto inserito all'interno del WCMS si è utilizzato un *markup* basato su XML. Si sarebbe certamente potuto utilizzare lo stesso metodo di inserimento del contenuto e di generazione delle pagine HTML già sviluppato per la versione già sviluppata del sistema di gestione, evitando così di dover riscrivere le parti relative dell'applicazione e riducendo quindi i tempi di sviluppo.

I vantaggi nell'adottare una codifica basata su XML sono però tali da compensare ampiamente lo sforzo di riscrittura: vengono eliminate tutte le rigidità intrinseche del sistema precedente. Si ha soprattutto un sistema più *standard* per la memorizzazione del contenuto, che rende possibile integrare più facilmente nuove funzionalità, magari fornite da altre applicazioni ed avere comunque una certa semplificazione delle funzioni che si occupano dell'elaborazione e della memorizzazione del contenuto.

Ad ogni pagina da visualizzare nel sito *web* corrisponde un documento XML con il contenuto della pagina. Ogni documento XML viene memorizzato, nella sua interezza e senza spezzare il contenuto in più parti, mediante l'uso di un unico campo CLOB[11] presente all'interno di una tabella del *database*. Nel momento in cui viene richiesta una pagina, il documento XML viene trasformato in un documento XHTML che viene inviato all'utente mediante HTTP.

Le uniche parti dell'applicazione che si devono occupare di gestire XML sono quelle relative all'inserimento e modifica del contenuto, la successiva visualizzazione, oltre alle parti che operano sul contenuto, come ad esempio la parte che si occupa dell'indicizzazione delle pagine per il motore di ricerca. Il motore di ricerca si può limitare all'elaborazione dei *character data*, senza occuparsi della presenza di elementi ed attributi: naturalmente l'elaborazione degli elementi permette di calcolare un peso maggiore per le parole presenti all'interno di certi elementi piuttosto che altri.

Il sistema di gestione utenti e di redazione distribuita è di fatto indipendente dal formato in cui il contenuto è inserito all'interno del sistema: se il *markup* viene esteso e modificato per aggiungere la visualizzazione di altri elementi non si deve modificare il sistema di gestione del *workflow*, ma solamente l'*editor*, la parte che si occupa della trasformazione ed eventualmente la parte del motore di ricerca che indicizza le pagine. Allo stesso modo, modificare l'*editor*, ad esempio utilizzando un sistema WYSIWYG, non va a toccare il resto del sistema, così come utilizzare un diverso sistema di trasformazione. In linea di principio, si può estrarre il contenuto dalla base di dati e trasformarlo in altri formati, ad esempio adatti alla pubblicazione

[11] *Character Large OBject*: questo tipo di colonna permette di memorizzare dati di testo di grande dimensione in modo opaco. La base di dati in questo caso si occupa solo di memorizzare ed estrarre i dati della colonna senza poter fare altre operazioni se non l'eventuale conversione della codifica del carattere.

cartacea o usare per la parte di erogazione delle pagine linguaggi diversi dal PHP per diminuire il carico di lavoro del *web server*.

Per la gestione delle immagini si è deciso di adottare un approccio semplice: vengono memorizzate all'interno di un *web server*. Nei documenti XML viene indicato semplicemente l'URL per recuperare le immagini da inserire nella pagina *web*. Si è preferito risolvere in questa maniera estremamente semplice la gestione delle immagini sopratutto perché nel sito esistente si hanno per la quasi totalità contenuti testuali e le immagini sono assai poche: andare a costruire un sistema di gestione più complesso che permettesse la memorizzazione di immagini all'interno della base di dati non avrebbe portato vantaggi sostanziali nell'utilizzo del sistema.

5.3.1 Pagine e contenuto

Per memorizzare il contenuto e per poterlo recuperare per l'erogazione il sistema utilizza due oggetti, che si traducono come due tabelle all'interno della base di dati, indicati nel rispettivamente come *pagina* e come *revisione*.

La pagina rappresenta il punto del sito in cui deve apparire una certa informazione, in altri termini è l'oggetto a cui ci si riferisce quando bisogna erogare una pagina HTML. Per ogni pagina HTML del sito esiste una sola pagina all'interno del sistema di amministrazione. La pagina in sé non ha memorizzato al suo interno il contenuto da erogare, ma rappresenta un oggetto che può essere recuperato mediante un URI.

La revisione rappresenta il contenuto informativo della pagina: si è preferito utilizzare il termine revisione anziché quello di contenuto per non generare confusione con il contenuto inteso come documento XML. Il contenuto visualizzato in una pagina viene preso da una revisione, ma la revisione non ha associato a sé nessun URI per poter essere recuperata[12], perché questa è una proprietà delle pagine.

Può esistere una sola revisione in linea, cioè visibile al di fuori del sistema di amministrazione, mentre possono esistere più revisioni fuori linea con diversi stati, ad esempio in attesa di approvazione, pronte o scadute. Le revisioni più vecchie vengono comunque mantenute internamente per avere traccia delle modifiche del contenuto di una pagina.

Tutte le revisioni memorizzate nel sistema sono sempre associate ad una pagina. L'associazione viene stabilita al momento della creazione del contenuto e non può essere modificata successivamente. Non possono esistere revisioni non collegate ad una pagina.

In condizioni normali, una pagina deve avere una ed una sola revisione in linea, ma è possibile che in casi particolari questo non sia vero: un tentativo di accesso

[12]In realtà le revisioni hanno un URI associato ad esse, ma questo è accessibile solamente dall'interno del sistema di amministrazione.

alla pagina produrrà la generazione di un messaggio standard di "pagina non disponibile", così come il tentativo di accesso ad una pagina inesistente. Le situazioni che possono portare ad avere una pagina senza revisione in linea associata sono diverse. Quando viene creata una nuova pagina, la revisione associata ad essa non è stata ancora approvata e quindi non può essere visualizzata, ma è disponibile all'interno del sistema di amministrazione: in questo modo altre revisioni possono inserire al loro interno riferimenti ipertestuali alla nuova pagina, anche se, come si può intuire, non è possibile navigare direttamente nella nuova struttura.

Questo metodo permette di costruire una parte del sito creando nuove pagine e preparando tutti gli *hyperlink* necessari, prima di procedere alla richiesta di approvazione, oppure quando viene creato un nuovo gruppo mettere a disposizione una pagina vuota su cui il gruppo possa iniziare a lavorare raggiungibile immediatamente da altre parti del sito. Un'altra situazione in cui una pagina può rimanere senza revisione in linea si verifica se viene richiesta la cancellazione di una revisione e non sono disponibili rimpiazzi. La presenza di pagine senza revisioni in linea associate può essere causata ancora da piccoli errori nella gestione delle pagine a tempo: ad esempio una revisione scade senza che ne esista il rimpiazzo oppure una pagina deve apparire nel futuro ma le pagine che hanno il collegamento verso quella pagina non sono a tempo o non sono correttamente sincronizzate. In questo modo, anche se vengono commesse piccole sviste si ha un comportamento definito del sistema che sia pure non perfetto, non ne pregiudica le funzionalità. Inoltre si risolve il problema di collegamenti a pagine interne del sito da parte di siti esterni, che non essendo controllabili dal sistema di pubblicazione possono continuare a rendere raggiungibili parti di sito che non dovrebbero più esser disponibili.

La tecnica utilizzata per la memorizzazione del contenuto è abbastanza particolare: normalmente vengono utilizzate metodologie di *check-out/check-in* e l'utilizzo di un sito *online* e di uno *offline* che risulta forse più facile da capire rispetto alla tecnica utilizzata. Questo tipo di tecnica è molto adatta quando il sito da gestire si può suddividere in aree ben distinte con responsabili indipendenti l'uno dall'altro e comunque con una struttura ad albero ben definita. Nel nostro caso avendo invece una struttura molto più libera e gruppi di lavoro che possono essere dipendenti l'uno dall'altro, bisogna usare un metodo differente.

Non si può avere una parte del sito *offline* ed una parte *online*, ma ogni pagina diventa un "minisito" che può avere una versione del contenuto visibile all'utenza ed almeno una utilizzabile per l'aggiornamento. Se il flusso di lavoro per l'approvazione diventa più complesso e si vuole avere la possibilità di avere pagine che appaiono automaticamente ad un dato istante di tempo futuro, si devono prevedere altre versioni del contenuto per questo scopo. Usare un insieme fisso di contenuti per ogni pagina (da approvare, approvato, in attesa) è troppo rigido: un approccio dinamico, in cui una pagina può avere diverse revisioni in uno stato diverso permette una maggiore flessibilità nell'uso del sistema e permette anche di archiviare i vecchi

contenuti in modo semplice ed elegante.

La tecnica utilizzata invece in Zope/CMF, cioè di implementare un flusso di lavoro con la definizione di *Access Control List* per definire la possibilità di approvare o meno i contenuti e definendo dei *workflow* personalizzabili per gestire il percorso di approvazione, per essere ricreata in un ambiente diverso da Zope, necessita di scrivere una grande quantità di codice ed è comunque abbastanza complessa da gestire dal punto di vista amministrativo.

5.3.2 Modelli di pagina

Anche in questo sistema di gestione del contenuto, come nella vecchia versione, esistono dei modelli di pagina, che permettono di impostare vari tipi di pagine da utilizzare per le varie sezioni del sito. Il comportamento dei modelli però è profondamente diverso rispetto alla vecchia versione: i modelli di pagina sono documenti XML memorizzati in una apposita tabella del *database*, che contengono uno scheletro del contenuto che deve essere inserito all'interno di una pagina.

Quando viene creata una nuova revisione a partire da un modello viene copiato XML dalla tabella che contiene i modelli a quella che contiene le revisioni. Si può anche creare una nuova revisione a partire da una revisione memorizzata all'interno della base di dati, ad esempio quando si deve aggiornare il contenuto di una pagina in linea. Non esistono più legami tra modello e revisione: il comportamento è molto simile a quello che si ha in un *word processor*, in cui si possono creare modelli di pagina da cui partire per scrivere documenti di una certa categoria.

L'utilizzo di questo sistema di gestione dei modelli porta ad avere una serie di vantaggi ed una maggiore flessibilità di gestione, in particolare nei seguenti punti:

- una volta che una revisione è stata creata il modello non è più necessario per la visualizzazione delle pagine o la modifica di una revisione, rendendo possibile modificare un modello senza che le pagine ricavate dal modello debbano essere fatte intervenire;
- è possibile modificare una revisione, ad esempio per aggiungere righe in una tabella o avere più o meno collegamenti ipertestuali senza dover creare tutta una serie di modelli simili;
- si può utilizzare la stessa interfaccia sia per la modifica delle pagine che per la modifica dei modelli, che possono anche contenere del testo *standard* precedentemente inserito;
- diventa facile gestire il *preview* di una revisione.

5.4 Entità e relazioni

Nella figura 5.14 è rappresentato lo schema semplificato delle entità rappresentate nella base di dati. Non sono state rappresentate le tabelle per la memorizzazione dei dati statistici, quella per i modelli e la tabella per la memorizzazione dei messaggi per non complicare inutilmente lo schema.

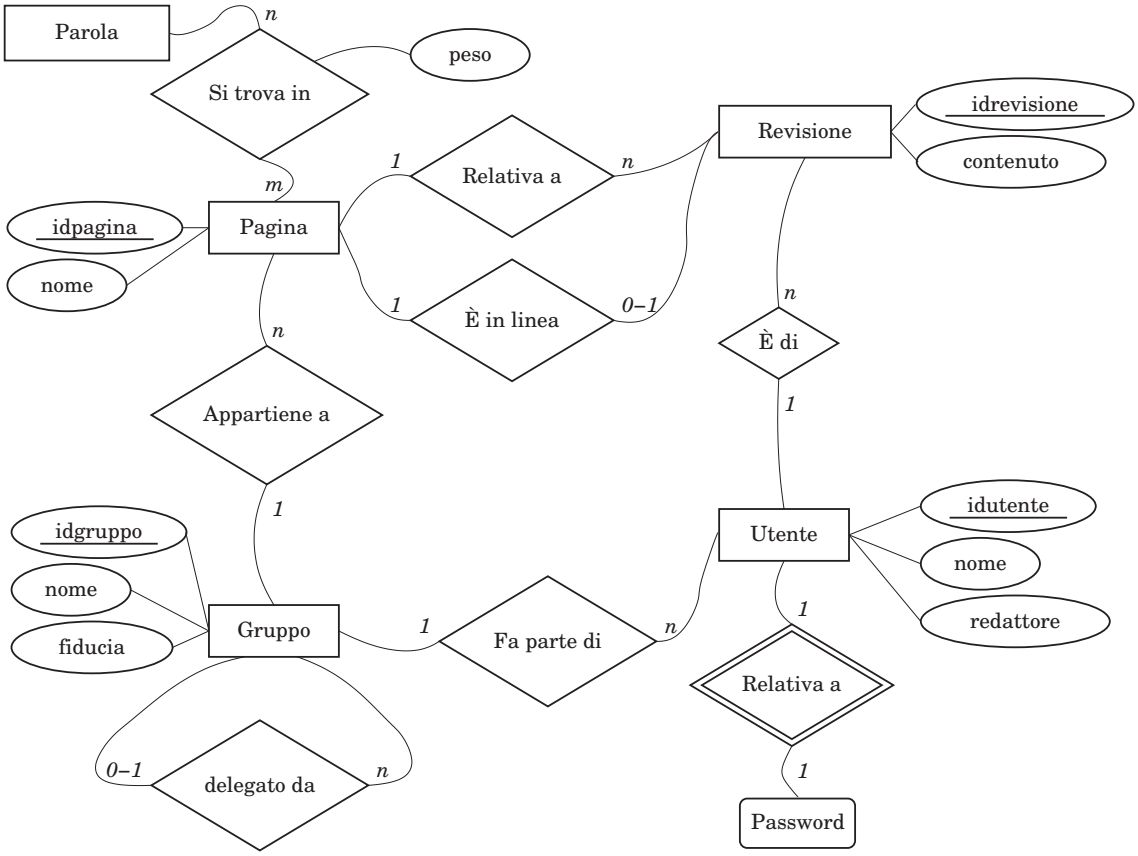


Figura 5.14. Diagramma semplificato entità-relazione della base di dati

Come si può osservare dallo schema, le pagine appartengono ai gruppi, mentre le singole revisioni appartengono agli utenti: non esiste una relazione diretta tra gruppo e singola revisione. Se la pagina appartiene ad un dato gruppo, le revisioni ad essa associate non necessariamente appartengono ad utenti del gruppo. La possibilità per un dato utente di creare una revisione per una pagina non è indicata all'interno della base di dati, ma deve essere stabilita dall'applicazione.

La scelta di strutturare in questo modo la base di dati è dovuta alla necessità di non complicare eccessivamente la base di dati. Si sarebbe potuta inserire una

relazione tra pagina ed utente "può modificare", ma ci sarebbe stato il problema di mantenere questa tabella aggiornata

Deve essere cura dell'applicazione aggiornare il motore di ricerca quando la revisione in linea di una pagina cambia: come si può osservare esiste una relazione tra parole e pagine e non tra parole e revisione. La scelta è stata fatta per avere una maggior efficienza nella fase di ricerca: vengono restituiti *hyperlink* alle pagine e non alle revisioni ed inoltre non ha senso indicizzare i contenuti che non siano *online* e disponibili ai visitatori.

Esiste un'entità "password" in cui sono contenute le informazioni di autenticazione per il singolo utente: logicamente le informazioni di autenticazione servono solo al momento del *login* e non per la gestione delle pagine. Se si dovessero utilizzare informazioni di autenticazione esterne questa struttura permette di poterlo fare, al prezzo di dover gestire una tabella aggiuntiva.

Nell'appendice C viene descritto il *database* SQL ottenuto dalle entità e dalle relazioni di cui si è appena discusso e dalle altre tabelle di supporto, con uno schema delle relazioni fra le varie tabelle e le istruzioni necessarie alla sua creazione.

Le entità presenti sono quindi state rappresentate da tabelle, e le relazioni 1 ad 1 ed 1 ad n sono state rappresentate da *foreign key* all'interno delle tabelle, così come la relazione forte tra utente e *password*. La relazione tra pagina e parola, essendo n ad m e con un parametro è stata rappresentata con una tabella la cui chiave è composta dalle *foreign key* della pagina e della parola.

5.5 Editor di pagine e modelli

La decisione di utilizzare XML per la memorizzazione del contenuto ha reso necessario dover riscrivere le parti del programma che servono a modificare i modelli ed i contenuti, con l'obiettivo di avere un sistema più semplice da utilizzare, rispetto al vecchio sistema.

La strada più interessante per avere un *editor* facile da usare è quella di utilizzare un sistema WYSIWYG, che sia simile ad un *word processor* oppure ad un *editor* grafico HTML. Dare la possibilità di scaricare il documento XML per modificarlo con un *editor* esterno non era una strada praticabile.

L'alternativa sarebbe stata di utilizzare un *applet* Java oppure un ActiveX all'interno della pagina HTML utilizzata per le modifiche del contenuto. Si sono quindi cercati dei sistemi con queste funzionalità, ma si è giunti alla conclusione che non esistono sistemi liberi con le caratteristiche richieste.

Si è quindi deciso di seguire una strada simile a quella della vecchia applicazione, cioè mediante l'uso di un'interfaccia basata su *form* HTML. Per semplificare il programma si è deciso di unificare il più possibile il codice necessario alla modifica dei contenuti e dei modelli. I *form* vengono generati trasformando XML in HTML:

elementi, *character data* ed attributi non vengono trasformati in elementi visuali, ma in *checkbox*, menù a discesa e caselle di testo, con l'aggiunta di campi nascosti che permettono di ricostruire un documento XML con i dati ricevuti dalla connessione HTTP.

In una prima fase la generazione dei *form* per la modifica dei modelli e dei *form* per la modifica dei contenuti era la stessa: si avevano quindi le stesse possibilità di modifica nel caso di una pagina o di un modello. La soluzione adottata permetteva troppa libertà nell'inserimento dei contenuti: di fatto l'uso dei modelli era inutile, perché partendo da un modello base era possibile costruire documenti con qualsiasi struttura.

Si è quindi modificato il codice che genera i *form* in modo che possa essere usato in due modi differenti. Nel caso si debba modificare un modello è possibile intervenire aggiungendo e togliendo qualunque tipo di elemento avendo cioè la possibilità di decidere lo stile di visualizzazione e la struttura del documento. Nel caso si debba modificare una pagina, alcuni tipi di elementi e di attributi sono bloccati, ma viene lasciata la possibilità di cancellare od aggiungere elementi in maniera controllata, ad esempio collegamenti ipertestuali o blocchi di testo. La soluzione adottata permette comunque una certa flessibilità nella creazione delle pagine e quindi permette di evitare la proliferazione di modelli simili. La manutenzione del codice risulta più semplice: non bisogna modificare due funzioni diverse nel caso si debbano gestire nuovi elementi od attributi.

5.6 Motore di ricerca

Il sistema di gestione del contenuto ha al suo interno un motore di ricerca che indicizza il contenuto del sito e permette all'utenza di ricercare le pagine di loro interesse inserendo una serie di parole all'interno di un *form* presente nelle pagine erogate. Il motore di ricerca non utilizza i moduli di *text retrieval* di Oracle: l'uso degli strumenti messi a disposizione dal RDBMS avrebbe portato a rendere l'applicazione fortemente dipendente da Oracle e dalla presenza all'interno della base di dati di queste funzioni.

Si è deciso quindi di scrivere un sistema di indicizzazione di testi che utilizza alcune tabelle all'interno del database e che mediante alcune funzioni scritte in PHP genera le *query* SQL che permettono di indicizzare il contenuto inserito all'interno del sistema. Una pagina PHP permette ai visitatori del sito di fare ricerche testuali: vengono generate le *query* SQL necessarie a recuperare le pagine che contengono le parole cercate ed ad ordinarle per importanza.

Le metodologie per la creazione dei motori di ricerca e dei sistemi di *information retrieval* in generale, sono un campo di ricerca molto vasto: l'implementazione di un motore di ricerca "esteso" è di per sé un progetto piuttosto complesso. Si è

quindi preferito non affrontare questa tematica ed optare per un sistema semplice, che comunque ha il vantaggio di essere facile da utilizzare sia per chi inserisce il contenuto che per l'utenza. Si è lasciata però la possibilità di sostituire questo motore di ricerca, come previsto dalle specifiche, con un altro più complesso senza andare a modificare il resto del sistema di gestione.

Tra i vari tipi di algoritmi utilizzabili per creare un motore di ricerca mediante l'uso di una base di dati redazionale si è preferito implementare un sistema basato sul concetto di *vector space*, sicuramente più facile da utilizzare da parte di un utente inesperto rispetto all'utilizzo di ricerche booleane ma allo stesso tempo leggero e semplice da implementare.

In un motore di ricerca basato sul modello *vector space,* ogni documento indicizzato viene rappresentato come un vettore n-dimensionale $\overline{v}$, con n uguale al numero di parole indicizzate. Il vettore viene calcolato come $\overline{v} = \sum_{i=0}^{n} t_i \, \overline{p_i}$, dove t_i rappresenta il peso della parola all'interno del documento, se questa è presente, e 0 se la parola non è presente mentre p_i è il versore associato alla parola.

Quando viene fatta una ricerca, si crea un vettore $\overline{q} = \sum_{i=0}^{n} t_i \overline{p_i}$, dove in questo caso t_i vale 1 se la parola corrispondente è presente nella stringa di ricerca. Vengono quindi cercati i vettori che rappresentano i documenti più vicini a $\overline{q}$, calcolando il prodotto scalare $\overline{q} \times \overline{v_n}$: le pagine più significative avranno un valore di questo prodotto più alto, e quindi ordinando i risultati ottenuti per questo valore si potranno fare apparire le pagine più significative per la ricerca.

Per l'implementazione di un motore di ricerca di questo tipo all'interno dell'applicativo sono state create tre tabelle all'interno della base di dati. La prima contiene le *stopword*, cioè le parole molto comuni, che non devono essere indicizzate. La seconda contiene un elenco di tutte le parole presenti nei testi indicizzati mediante il motore di ricerca. La terza tabella è un *inverted index*, cioè va ad associare le parole con i testi che le contengono, con un peso che indica l'importanza che ha quella parola all'interno del testo.

Nel momento in cui una pagina viene indicizzata, normalmente al momento della messa in linea della pagina, una funzione PHP estrae i *character data* dal documento XML ed inserisce nella terza tabella il riferimento alla parola, il riferimento alla pagina in cui il contenuto appare ed un peso che è calcolato come il numero di volte che una parola appare nel testo, pesato in maniera diversa se in un elemento od in un altro. I termini presenti nel testo indicizzato ma che non sono presenti nella tabella che contiene l'elenco delle parole, vengono automaticamente aggiunti a quest'ultima, mentre le *stopword* presenti non vengono ovviamente prese in considerazione.

Quando un utente utilizza il motore di ricerca per trovare le pagine di suo interesse, inserisce una stringa contenente un elenco di parole chiave. La stringa viene spezzata ed i singoli termini vengono inseriti in una tabella temporanea. Viene quindi effettuata una *query* SQL che restituisce i puntatori alle pagine che contengono

quei termini, ordinati per la somma dei pesi delle parole che sono memorizzati nella tabella che associa parole a pagine.

Aumentando il numero dei termini nella stringa di ricerca aumenta il numero delle pagine trovate, ma quelle più significative tenderanno ad essere messe sempre più in evidenza: in questo modo anche una ricerca con termini imprecisi o generici tenderà comunque a generare risultati significativi. Se si fosse utilizzato l'approccio di restituire solo le pagine che contengono tutti i termini ordinati secondo il peso delle pagine, una ricerca con un termine inesistente non avrebbe prodotto risultati, cosa che avrebbe potuto confondere gli utenti.

Figura 5.15. L'aspetto del sito per i visitatori

5.7 Conclusioni

Con quanto esposto in questo capitolo risulta abbastanza chiaro capire quali sono i concetti che stanno dietro a questo CMS e riuscire ad utilizzare con successo il sistema. Sono state anche accennate le scelte progettuali che hanno portato a questa implementazione.

Nelle appendici è stato descritto in dettaglio il funzionamento interno del sistema utile per espandere ed a modificare l'applicazione per meglio adattarsi alle proprie esigenze ed a comprendere meglio cosa facciano le varie procedure che compongono il sistema.

Capitolo 6

Considerazioni finali

L'applicazione sin qui descritta appartiene a pieno titolo ai *web content management system* scritti appositamente per risolvere i problemi di una particolare organizzazione per poter soddisfare le esigenze specifiche per la pubblicazione di contenuti sul proprio sito.

La scelta del modo in cui procedere, come si è detto, è stata dettata dall'ambiente operativo piuttosto particolare in cui deve essere erogato il sito, ambiente operativo che ha imposto anche scelte nel tipo di soluzione adottare, che in alcuni casi non sono sicuramente ottimali.

L'utilizzo di un CMS libero già scritto avrebbe sicuramente permesso di avere un'applicazione funzionante in molto meno tempo e certamente in diverse condizioni si sarebbe seguita questa strada.

L'applicazione presenta alcune particolarità interessanti. La separazione tra HTML ed inserimento del contenuto è molto forte, e chi inserisce il contenuto non vede direttamente la pagina *web*, ma deve ragionare in termini di blocchi di informazione e di stili associati. ciò può sembrare una limitazione, ma in realtà aiuta sia ad avere un aspetto grafico coerente del sito generato, sia a permettere a chi scriva a concentrarsi su quello che è veramente importante in una pagina *web*, il contenuto, e non a perdersi nella gestione dell'aspetto della pagina HTML.

Non avendo a che fare con HTML non è necessario conoscerlo per poter scrivere dei documenti: è possibile utilizzare questo sistema anche da parte di persone che non conoscono questo *markup*, senza andare incontro ai problemi tipici che si hanno con l'utilizzo di *editor* grafici. In questo modo si mettono in grado persone con una scarsa cultura informatica di poter pubblicare pagine *web* ben fatte ed in un sito coerente ed aggiornato senza che esse debbano fare grossi sforzi d'apprendimento.

La relativa facilità d'uso di questa applicazione può fare supporre che le evoluzioni future del sistema lo possano trasformare da un sistema costruito *ad hoc* ad un WCMS di utilizzo generale: certamente questo non era un obiettivo del progetto e non sarà sicuramente un risultato raggiungibile in tempi brevi, ma nulla vieta che

questo possa accadere. Le organizzazioni in cui un sistema simile a questo potrebbe risultare utile non si limitano certamente al servizio Passepartout del Comune di Torino.

Alcuni aspetti di quest'applicazione si prestano ad ulteriori sviluppi, perché alcune parti importanti in un ambito generale non sono applicabili o sono inutili nel caso particolare del sito InformaHandicap Piemonte o perché il loro studio e la loro applicazione pratica possono essere, per le loro dimensioni, oggetto di un lavoro separato da integrare successivamente con questo.

Gli aspetti in cui l'applicazione può essere migliorata sono fondamentalmente il motore di ricerca, l'indipendenza dall'uso di un particolare RDBMS, l'uso di XSLT e l'*editor* usato per inserire e modificare il contenuto all'interno del sistema.

Il motore di ricerca attualmente utilizzato è piuttosto semplice. Lo si può però sostituire con uno più complesso, come ad esempio un motore di ricerca semantico, che cioè abbia la conoscenza del significato delle parole inserite e possa quindi consigliare l'utilizzo di sinonimi, oppure che permetta di eseguire richieste più complesse ed articolate.

Questa applicazione, allo stato attuale può funzionare solamente utilizzando l'RDBMS Oracle. Essendo questo un'applicazione proprietaria, chiusa, complessa da configurare correttamente e decisamente costosa, può essere interessante sfruttare meglio le potenzialità delle classi ADODB per arrivare ad una maggiore indipendenza dal *database* utilizzato.

Il prototipo dell'applicazione è stato scritto utilizzando PostgreSQL, quindi andando ad isolare le parti di codice che utilizzano sintassi proprietarie delle *query* SQL e della gestione dei CLOB si può ottenere un sistema di gestione del contenuto che può utilizzare RDBMS liberi come appunto PostgreSQL od Interbase.

La flessibilità nel scegliere il *database* di *backend* tra uno di quelli liberi dà la possibilità di utilizzare un *web content management system* completamente libero, utilizzabile per siti *web* diversi da quello per cui l'applicazione è stata scritta. Per questi motivi, quando sarà disponibile una versione del programma indipendente dalla base di dati, verrà rilasciata al pubblico con la GNU *General Public License*.

Se vengono usate le funzioni di trasformazione XSLT di PHP anzichè andare ad utilizzare un programma PHP per eseguire le trasformazioni dei documenti XML si può arrivare a poter gestire in maniera più flessibile l'aspetto delle pagine generate e del tipo di documento XML che viene memorizzato all'interno del sistema.

Per utilizzare XSLT è necessario che il PHP utilizzato sul *web server* che eroga le pagine sia stato compilato con le opportune opzioni e librerie addizionali per XSLT.

Anzichè utilizzare l'interfaccia basata su *form* HTML attualmente utilizzata si può pensare di andare ad utilizzare *applet* Java, oppure programmi ActiveX (che però limiterebbero all'utilizzo di *client* con sistema operativo Microsoft per l'accesso

al sistema di gestione) per avere una interfaccia di *editing* che presenti, mentre si modifica il contenuto, un documento simile a quello che verrà poi pubblicato.

L'applicazione attualmente è in fase di *test* su una macchina con sistema operativo Solaris-SPARC per l'erogazione delle pagine *web* che colloquia con un database Oracle installato su una macchina GNU/Linux-Intel presso il Politecnico di Torino. In collaborazione con gli operatori del Servizio Passepartout si sta effettuando una fase di *debug* e di miglioramento di alcuni aspetti dell'applicazione. Una volta arrivati ad una versione stabile, ed espletate le formalità burocratiche per l'installazione dell'applicazione sulle macchine del CSI, si passerà ad una fase di *test* dell'ambiente di produzione ed alla conversione dei contenuti del sito dinamico precedente in XML per l'erogazione con questo sistema.

In conclusione un doveroso ringraziamento va a Marco Griva, che con il suo lavoro di tesi ha permesso di avere un sistema di gestione del contenuto funzionante da cui si è potuti partire come base per comprendere i requisiti che doveva avere questa applicazione, fungendo anche come ispirazione per l'attuale sistema e per la sua disponibilità per illustrare in dettaglio il funzionamento interno del sistema da lui sviluppato. Un grazie va anche a Pino Rossetti del servizio Passepartout per la disponibilità dimostrata durante la fase di prova e *debug* del programma e per il *feedback* costruttivo che ha permesso di migliorare alcuni aspetti e definire meglio i requisiti del sistema di gestione del contenuto.

Appendice A

Il DTD utilizzato

In questa parte verrà descritta la struttura del *markup* utilizzata per memorizzare il contenuto all'interno del sistema.

Listato A.1 *Il DTD utilizzato per il WCMS*

```
<!ELEMENT pass (head,body)>

<!ELEMENT head (title,summary?,keywords?)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT summary  (#PCDATA)>

<!ELEMENT keywords (#PCDATA)>

<!ELEMENT body ((div|include)*)>

<!ELEMENT include EMPTY>

<!ATTLIST include uri CDATA #REQUIRED
                  clone (yes|no)  "no">

<!ELEMENT div ((text|image|link|table|hr)*)>

<!ATTLIST div
          style               NMTOKEN                #REQUIRED
          altstyle        (h1|h2|h3|h4|h5|blockquote|plain|address)
                                                     #IMPLIED>

<!ENTITY % decorations
        "bullet  (circle|cross|dot|none)  #IMPLIED
         clone   (no|yes)                 #IMPLIED
         inline         (no|yes)                 #REQUIRED">
```

```
29
30 <!ELEMENT text (#PCDATA)>
31
32 <!ATTLIST text %decorations;>
33
34 <!ELEMENT image        EMPTY>
35
36 <!ATTLIST image        %decorations;
37           uri        CDATA #REQUIRED
38           alt   CDATA #IMPLIED>
39
40 <!ELEMENT link         (#PCDATA)>
41
42 <!ATTLIST link         %decorations;
43           type         (internal|external)        #REQUIRED
44           visible (yes|no)              "yes"
45           shurl (yes|no)                #IMPLIED
46           uri   CDATA                   #REQUIRED
47           title CDATA                   #IMPLIED>
48
49 <!ELEMENT table        ((left,right)+)>
50
51 <!ELEMENT left         (#PCDATA)>
52
53 <!ELEMENT right        (#PCDATA)>
54
55 <!ATTLIST right clone (yes|no)          #IMPLIED>
56
57 <!ATTLIST table %decorations;
58           summary CDATA                 #REQUIRED
59           caption CDATA                 #IMPLIED>
60
61 <!ELEMENT hr        EMPTY>
```

L'elemento radice è l'elemento `pass` che rappresenta il contenuto di ogni singola pagina del sito (riga 1). All'interno dell'elemento `pass` sono presenti obbligatoriamente gli elementi `head` e `body` (riga 3). L'elemento `head` contiene i metadati della pagina, mentre l'elemento `body` contiene i dati della pagina. All'interno di `head` sono contenuti nell'ordine gli elementi `title`, `summary` e `keywords`.

`title` Contiene il titolo della pagina, e può essere rappresentato nel codice HTML generato con l'elemento `<title>`. Questo elemento è obbligatorio. (riga 5)

`summary` Contiene un riassunto del contenuto della pagina, e può essere rappresentato con l'elemento `meta name="description"` in HTML, ed utilizzato dal motore di ricerca per avere dati più significativi sul contenuto del documento. Questo elemento è opzionale. (riga 7)

`keywords` Contiene un elenco di parole chiave, che possono essere rappresentate con l'elemento `meta name="keywords"` in HTML, ed utilizzato dal motore di ricerca per sapere quali sono le parole chiave del documento. Questo elemento è opzionale (riga 9)

All'interno dell'elemento `body` (riga 11) sono contenuti una serie di elementi `div` e `include`, che vanno a comporre il contenuto della pagina.

`div` Contiene un blocco di altri elementi e definisce lo stile degli elementi in esso contenuti. Lo stile influenza sia il tipo di elemento utilizzato in HTML che lo stile ad esso associato. I parametri `dell'attributo` sono `style` (obbligatorio) che indica che stile usare (ovvero l'attributo `class` di HTML),mentre l'attributo `altsyle` indica che tipo di elemento HTML deve essere utilizzato per il blocco: si possono quindi generare intestazioni con `<h1>`, `<h2>`, `<h3>`, ecc... oppure se l'attributo è assente oppure uguale a `plain`, indica che deve essere generato un blocco `<div>`. (righe 18-22)

`include` Contiene l'indicazione per includere il contenuto di un'altra pagina del sito in questa[1]. Inserito per compatibilità con la vecchia versione del sistema di gestione del sito. I parametri sono `uri` che rappresenta la chiave primaria della riga che contiene la pagina da includere e `clone` che permette di duplicare l'elemento nell'*editor* delle pagine. (righe 13-16)

All'interno dell'elemento `div` sono contenuti le parti del documento che devono essere visualizzate. Gli elementi all'interno hanno tre attributi comuni `bullet`, `clone` ed `inline`. L'attributo `bullet` serve per inserire un richiamo opzionale prima del blocco. L'attributo `clone` permette di duplicare l'elemento nell'*editor* delle pagine. L'attributo `inline` modifica il comportamento per il ritorno a capo[2]: per i testi (gli elementi `text` e `table`) modifica il modo in cui i ritorni a capo inseriti nell'*editor* vengono riportati nella pagina HTML generata, mentre per gli altri elementi indica se devono essere inseriti nel flusso del testo o meno.

Gli elementi che possono essere inseriti all'interno di un elemento `div` sono quelli che seguono.

`text` Questo elemento permette di inserire il testo contenuto al suo interno nella pagina. I blocchi di testo vengono resi in HTML con `<p>` se ci si trova all'interno di un `<div>` , mentre all'interno di altri elementi vengono inseriti dei `<br>`. Il parametro `inline` modifica la gestione dei ritorni a capo. (righe 30-32)

[1] Si è utilizzato un elemento anziché utilizzare una *external general entity* ed il suo riferimento nel documento per rendere più semplice l'*editing* del sito e perché l'inclusione di una pagina in un'altra richiede comunque una trasformazione del contenuto della pagina inclusa, che estragga solo la parte contenuta all'interno di `body`.

[2] Il significato di `inline` qui è simile a quello che si incontra nei CSS e non a quello dato nella versione vecchia del sito. Per ottenere questo effetto bisogna utilizzare l'elemento `include`.

`hr` Questo elemento permette di inserire una riga orizzontale nella pagina. È equivalente all'elemento `<hr>` di HTML. (riga 61)

`image` Questo elemento permette di inserire un'immagine nella pagina HTML. Gli attributi aggiuntivi che usa sono `uri` che permette di definire l'URI dove recuperare l'immagine, ed `alt` che corrisponde alla descrizione dell'immagine. Questi elementi sono entrambi obbligatori. (righe 34-38)

`link` Questo elemento permette di inserire un collegamento ipertestuale, cioè l'attributo HTML `<a>`, il cui testo è contenuto come *character data* all'interno dell'elemento. Gli attributi aggiuntivi sono `type`, che se uguale ad `external` indica che il collegamento è verso l'esterno, e l'attributo `uri` in questo caso rappresenta un URI, mentre se è uguale ad `internal` rappresenta un collegamento interno, e quindi l'attributo `uri` rappresenta il numero di pagina interno. Sia `uri` che `type` sono obbligatori. L'attributo `shurl` se ha come valore `yes` modifica l'aspetto del collegamento nella pagina HTML: il testo del *link* ipertestuale è l'attributo `uri`, mentre il *character data* diventa del testo normale che segue il collegamento ipertestuale.

Gli altri attributi aggiuntivi sono `visible` che permette se ha il valore uguale a `no` di non fare apparire il collegamento nella pagina visualizzata, cosa utile se la pagina puntata dal collegamento non è momentaneamente disponibile e l'attributo `title` che se presente inserisce un attributo `title` nell' `<a>` generato in HTML. (righe 42-47)

`table` Questo elemento permette di costruire una tabella a due colonne. Gli elementi aggiuntivi sono `summary` e `caption`, che corrispondono ai relativi attributi ed elementi di HTML 4.0. All'interno della tabella esiste una serie di elementi che devono essere a coppie, `left` e `right`, che contengono il testo da visualizzare rispettivamente nella colonna sinistra ed in quella di destra della tabella. L'elemento `right` ha l'attributo `clone`, che permette di duplicare le righe della tabella (sia l'elemento `right` che l'elemento `left`). La prima riga della tabella contiene il titolo delle colonne (ovvero è trasformata in un elemento `<th>` anziché in un elemento `<tr>`). (righe 43-52)

Per scrivere il DTD è stata analizzata la struttura tipica delle pagine HTML generate dal vecchio sistema e si è giunti a scrivere un linguaggio di *markup* che permettesse di ottenere pagine HTML di questo tipo, quindi con paragrafi con stili diversi, immagini e tabelle. Per aumentare la praticità del sistema e facilitare la generazione di HTML si è deciso di separare le informazioni sullo stile dei blocchi dai blocchi di testo stessi.

Nulla vieta di modificare ed adattare il *markup* utilizzato: anzi bisogna dire che anche il DTD ha subito un processo evolutivo che ha portato ad aggiungere attributi ed elementi a seguito di richieste specifiche degli operatori del servizio Passepartout.

Il *markup* utilizzato permette la creazione di pagine con aspetto molto simile a quello delle pagine generate con il vecchio sistema di gestione, con una maggiore flessibilità nella scelta della resa grafica dei vari elementi.

L'indicazione di come viene trasformato XML in HTML è quella attualmente utilizzata nel WCMS, ed è fortemente legata alla rappresentazione HTML della pagina: dato che l'unico formato di pubblicazione del sistema è HTML, la scelta fatta non pregiudica le funzionalità del sistema.

Appendice B

Struttura dell'applicazione

In questa appendice si esamina lo scopo dei vari *file* PHP che servono al funzionamento dell'applicazione, suddivisi per aree funzionali

B.1 Erogazione del contenuto

Per l'erogazione del contenuto ai visitatori del sito i *file* che vengono utilizzati sono `pagina.php` e `sitemap.php`, che a loro volta utilizzano `mydef.php` ed `xmlrender.php`. Per le informazioni sullo stile vengono utilizzati tre CSS. Il primo, `stili.css`, è lo stile utilizzato per la visualizzazione normale, il secondo `ipo.css` è quello utilizzato per la visualizzazione ad alto contrasto mente il terzo, `stiliadm.css` viene utilizzato nella parte di gestione del sistema.

In questa parte si descriveranno in dettaglio le funzioni che hanno questi *file*.

`pagina.php` Serve a visualizzare le pagine richieste dai visitatori del sito. Accetta due parametri: `pag` che rappresenta il numero di pagina necessario ad identificare la pagina richiesta (se questo parametro manca viene visualizzata la pagina principale, cioè la pagina 0), e `modo` che se uguale a 0 commuta la visualizzazione nella versione ad alto contrasto, mentre se uguale ad 1 commuta la visualizzazione nella versione normale (se questo parametro è assente il tipo di visualizzazione rimane quello precedente).

Per visualizzare una pagina per prima cosa si verifica se esiste già una sessione associata al *browser* che ha mandato la richiesta HTTP e se non esiste viene creata con una barra di navigazione iniziale, altrimenti si recuperano i dati delle pagine già visitate e si crea una struttura dati per creare la barra di navigazione. Viene quindi inserito in una tabella l'ora e la pagina richiesta, insieme ad altri dati utili per ricavare le statistiche di accesso del sito. Successivamente si tenta di recuperare il contenuto della revisione *online* associata alla pagina

richiesta: in caso di errore viene generata una pagina con l'errore di "pagina non trovata". Se invece non ci sono errori, viene creato un oggetto della classe `lameparser` che si occupa di trasformare il contenuto XML in XHTML per l'erogazione.

`xmlrender.php` Serve a definire la classe `lameparser`, che utilizza il *parser* SAX disponibile in PHP per leggere il documento XML e richiamare le funzioni che permettono di generare XHTML corrispondente agli elementi del documento XML. Il costruttore di `lameparser` accetta tre parametri: il primo è il documento XML da trasformare, il secondo indica il numero di pagina da visualizzare ed il terzo, se uguale a `"full"` trasforma la pagina XML in una pagina XHTML completa, mentre se uguale ad `"half"` genera solo gli elementi presenti nell'elemento `<body>`, sia per poter permettere l'inclusione di sottopagine all'interno di una pagina, che per l'uso nella funzione di visualizzazione revisione. La visualizzazione vera e propria del documento XHTML avviene chiamando la procedura `parse` della classe. Questa procedura accetta tre parametri: il primo è un testo XHTML che viene aggiunto all'inizio del `<body>` (utilizzato per la barra di navigazione), il secondo indica quale CSS deve essere utilizzato, ed il terzo è un testo XHTML che viene aggiunto alla fine del `<body>`. Modificando `lameparser` si può utilizzare una diversa tecnica di trasformazione, ad esempio basata su XSLT, per ottenere una diversa struttura della pagina visualizzata.

`sitemap.php` Chiamando questa pagina viene visualizzata una mappa del sito, che rappresenta i primi tre livelli del sito partendo dalla pagina principale. Partendo dalla pagina principale, vengono estratti gli elementi di tipo `<link>` contenuti nel documento utilizzando sempre il *parser* SAX di PHP, e se questi sono di tipo interno vengono visitate in modo ricorsivo le pagine a cui questi elementi puntano, fermandosi al terzo livello (questo parametro si può configurare modificando il *file*). Da questi dati viene ricostruito un elenco puntato dei primi livelli del sito.

`mydef.php` Questo *file* si occupa di aprire una connessione con il *database*, con i parametri indicati al sui interno ed a modificare il formato di rappresentazione delle date. Sono anche definite al suo interno alcune funzioni ed alcune costanti utilizzate dalle pagine del sistema di gestione e per la visualizzazione all'utenza.

`cerca.php` Questo file permette di rispondere alle *query* verso il motore di ricerca testuale. Se richiamato con il parametro `ts` definito, procede alla ricerca delle pagine più significative rispetto alla stringa `cerca`, come spiegato nella sezione 5.6, se necessario suddividendo i risultati ottenuti in pagine ed aggiungendo dei bottoni per poter andare avanti od indietro all'interno dei risultati. Se

viene utilizzato un motore di ricerca diverso da quello attualmente disponibile questa procedura deve essere riscritta o modificata in modo opportuno.

Nella *directory* `img/` sono contenute le immagini utilizzate per la visualizzazione delle pagine da parte del sistema di gestione ed eventualmente le immagini da visualizzare all'interno delle pagine. Per queste ultime comunque si può specificare un URL completo in modo da andarle a recuperare da un'altra *directory* od anche da un altro *web server*.

B.2 Gestione del contenuto

I *file* necessari alla gestione del contenuto sono tutti contenuti all'interno della *directory* `adm/`, tranne `view_rev.php` contenuto nella *directory* di base. In alcuni casi è stata utilizzata una struttura a coppie, in cui un *file* PHP contiene le istruzioni necessarie a generare la maschera di inserimento dati, ed un secondo *file* PHP si occupa di andare a modificare i dati all'interno della base di dati, mentre in altri casi si è utilizzato un unico *file* che modifica la base di dati o mostra una maschera di inserimento in dipendenza dei parametri con cui è chiamato.

I *file* utilizzati dal sistema di gestione, oltre a `mydef.php`, fanno uso di altri *file* PHP che definiscono delle funzioni di utilità, sia perché utilizzate in più punti diversi, che per permettere una loro più facile sostituzione ed aggiornamento.

`insetxt.php` Questo *file* deve rendere disponibile una funzione `update_sengine` per l'indicizzazione di una pagina all'interno del motore di ricerca. Questa funzione accetta tre parametri. Il primo indica il tipo di azione da compiere: se è uguale a "`d`" i riferimenti all'interno del motore di ricerca ad una pagina vengono eliminati, mentre se è uguale ad "`i`", la pagina viene indicizzata. Il secondo è il contenuto, sotto forma di documento XML da indicizzare (che è nullo, se l'operazione è di eliminazione pagina). Il terzo è il numero identificativo della pagina. La funzione viene chiamata sempre all'interno di una transazione, e restituisce 0 in caso di successo ed 1 in caso di errore: in questo caso effettua anche un *rollback* dell'intera transazione. Per cambiare il motore di ricerca basta riscrivere `update_sengine` che abbia le stesse caratteristiche di questa funzione e modificare la parte che si occupa di generare le *query* al motore di ricerca, come accennato in B.1.

`myfunct.php` Questo *file* mette a disposizione alcune funzioni utilizzate per gestire la selezione delle pagine all'interno del sistema di gestione, generando le varie liste di selezione ed il *javascript* necessario al loro funzionamento.

`writeJS` genera il javascript necessario ad avere l'elenco delle pagine suddiviso in vari blocchi per la parte iniziale

`mkpagesel` genera la maschera di selezione pagina di utilizzo generale

`mkpagesel2` genera la maschera di selezione pagina utilizzata all'interno di `xmledit.php`

`getgroupsunder` restituisce due *array* contenti l'elenco ed il nome dei gruppi discendenti da un dato gruppo

`styles.php` Questo *file* mette a disposizione alcune funzioni necessarie per la gestione degli stili all'interno dell'*editor* e per la generazione delle pagine che visualizzano gli stili disponibili. All'inizio del *file* viene definito un *array*, `stili`, che deve contenere gli stili definiti all'interno dei CSS e l'elemento HTML a loro associato. Per poter selezionare correttamente gli stili, aggiunte od eliminazioni degli stili definiti nel CSS devono corrispondere modifiche ed eliminazioni a `stili`.

`xmledit.php` Questo *file* deve mettere a disposizione due funzioni che servono a gestire l'*editor* per le pagine e per i modelli.

`xmlprepare` serve a generare del codice javascript necessario al funzionamento dell'*editor*

`xmledit` genera la parte del *form* che serve per inserire e modificare il contenuto XML. Accetta due parametri: il primo è il documento XML che deve essere modificato, mentre il secondo, se uguale a 0 genera i campi del *form* per l'editor delle pagine, se uguale ad 1 genera i campi del *form* per l'*editor* dei modelli. Questa funzione deve essere chiamata all'interno del *form* in quanto genera solo la parte strettamente necessaria alla modifica del contenuto.

`xmlrebuild.php` Definisce una classe `xmlrebuild` che serve a rigenerare un documento XML dai campi del *form* generato da `xmledit`. Il costruttore della classe inizializza alcune variabili interne: per ottenere il documento XML bisogna prima chiamare il metodo `xform`, senza parametri, che serve a creare il documento XML, e poi il metodo `retxml`, senza parametri, che restituisce il documento generato.

`view_rev.php` Permette di visualizzare la revisione `idrevisione` senza dover passare dalla visualizzazione delle pagine. In questo modo si possono visualizzare le revisioni in corso di lavorazione. Rispetto alla visualizzazione di una pagina completa la barra di navigazione è sostituita da una finestra informativa, il titolo e l'intestazione della pagina sono diversi ed ovviamente non si può accedere al motore di ricerca.

Gli altri *file* PHP sono quelli che vengono richiamati dal *browser* quando un giornalista deve eseguire delle operazioni all'interno del sistema del contenuto. Per accedere a questi *file* è necessario che l'utilizzatore si sia autenticato con successo, e che quindi le variabili di sessione utilizzate siano presenti. In caso contrario viene generato l'errore HTTP di "accesso negato".

`login.php` Si occupa di autenticare l'utente e quindi di permettergli l'accesso al sistema di gestione. Viene creata una sessione, ed in caso di *login* e *password* corretti, provvede a visualizzare una schermata informativa, aggiornare la data dell'ultimo collegamento all'interno di una tabella della base di dati ed a inserire i valori corretti all'interno delle variabili di sessione. Se chiamato con una sessione attiva e corretta provvede invece a distruggere la sessione, ad aggiornare la data in cui il collegamento è terminato all'interno della base di dati, ed a presentare un menù che consente di effettuare di nuovo l'autenticazione.

`main.php` Serve a generare la schermata principale del sistema di gestione, visualizzando quindi l'elenco delle attività, le revisioni aperte ed i comandi disponibili.

`passwd.php` Serve a poter modificare da parte dell'utente la *password* con cui accede al sistema di gestione. Se richiamata senza parametri genera la maschera di cambio *password*, se invece sono presenti i valori inseriti nella maschera provvede ad aggiornare *l'hash value* della *password* memorizzato nella base di dati.

`msg.php` Serve a poter generare manualmente i messaggi inseriti nella lista delle attività degli utenti. Se richiamato senza parametri genera la maschera per l'inserimento del messaggio, mentre se ci sono i parametri generati dalla maschera provvede ad inserire il messaggio all'interno della base di dati.

`newpage1.php` Serve per creare una nuova pagina oppure una nuova revisione a partire da un modello. Se chiamato senza parametri genera la maschera di creazione nuova pagina. Se chiamato con il parametro `ts` uguale ad 1 (cioè se chiamata dalla maschera di creazione nuova pagina di `newpage1.php`) genera una nuova pagina e la revisione associata con il contenuto copiato dal modello selezionato. Se `ts` è invece uguale a 2 (cioè se chiamata da `editrev.php`) invece crea solamente una revisione con il contenuto del modello selezionato e l'associa con la pagina data con il parametro `idp`.

`editrev.php` Questo *file* viene utilizzato per diverse funzioni legate all'inserimento ed alla modifica del contenuto. Se non è presente il numero della revisione nel parametro `idrevisione` e nemmeno il numero di pagina nel parametro `idpagina`, viene generata la maschera di selezione pagina. Se è presente il solo

parametro `idpagina` viene generata la maschera di selezione della revisione. Se invece è presente il parametro `idrevisione` viene generata la maschera di *editing* della revisione. In questo caso, se la revisione selezionata non è dell'utente che sta effettuando l'operazione, oppure è stata già approvata da qualcuno viene creata una nuova revisione, copia di quella selezionata. In caso contrario si va a modificare la revisione selezionata.

`edit_upd.php` Questo file viene richiamato da `editrev.php` una volta terminata la modifica del contenuto. Se il parametro `sub` è uguale a "`finish`" viene generato un messaggio di richiesta approvazione pagina e viene aggiornato il contenuto della revisione all'interno del *database*. Se è uguale a "`draft`" viene solamente aggiornato il contenuto della revisione. Se invece è uguale a "`forget`" invece non fa nulla.

`delrev.php` Questo file viene utilizzato per generare una richiesta di cancellazione pagina. Il funzionamento è simile a quello di `editrev.php`, nel caso non sia presente il parametro `idrevisione`. Se questo parametro è presente il comportamento varia se la revisione selezionata è ancora aperta o meno. Se è ancora aperta, viene cancellata la revisione dalla base di dati: se la pagina a cui era associata la revisione non ha nessun'altra revisione viene eliminata. Se la revisione è stata già approvata viene invece generata una richiesta per mandare *offline* la revisione.

`listpg.php` Questo *file* genera la tabella che contiene l'elenco delle pagine e delle revisioni presenti nel sistema.

`showstyles.php` Questo stile genera una pagina di esempio con tutti gli stili disponibili.

Altri *file* PHP sono utilizzati per le funzioni relative all'approvazione delle pagine ed ai comandi disponibili solo ai redattori. Se si tenta l'accesso come giornalista normale alla pagine riservate ai redattori oppure se si tenta di accedere alle pagine di modifica dei modelli se non si è il redattore del nodo principale, viene generato un errore HTTP di "accesso negato" e la sessione corrente viene distrutta, e quindi il giornalista deve comunque effettuare nuovamente il *login* per continuare ad utilizzare il sistema di gestione.

`chgrp.php` Genera la maschera necessaria alla modifica del gruppo di appartenenza di una pagina.

`chgrp2.php` Va a modificare il gruppo di una pagina all'interno della base di dati utilizzando i dati forniti da `chgrp.php` .

`namepage.php` Genera la maschera necessaria alla modifica del nome di una pagina.

`namepage2.php` Verifica che il nome della pagina inserito con `namepage.php` sia unico, ed in caso positivo va a modificare il nome della pagina all'interno della base di dati.

`modelsel.php` Genera la maschera necessaria alla selezione di un modello per la sua creazione, modifica od eliminazione.

`modeledit.php` Serve per gestire creazione, modifica ed eliminazione dei modelli. Il parametro `nuovo` seleziona le varie operazioni possibili. Se è uguale a "`D`" elimina il modello specificato da `idmodello`, se è uguale a "`N`" crea un modello a partire da quello specificato con `idmodello` e genera la maschera di modifica per il modello appena creato, mentre se è uguale a "`E`" genera la maschera di modifica del modello specificato con `idmodello`. Il modello 0, cioè il modello vuoto non si può né cancellare né modificare.

`model_upd.php` Serve per memorizzare le modifiche ad un modello effettuate con `modeledit.php` all'interno della base di dati.

`approve.php` Questa pagina genera la maschera di approvazione o rifiuto di una revisione o di una cancellazione per i redattori. Una volta deciso se rifiutare od approvare una pagina, viene chiamato `approve2.php` per le operazioni effettive sulla base di dati.

`approve2.php` Questa pagina gestisce tutta la logica del sistema di approvazione distribuita. Data una revisione da approvare va a costruirsi un *array* con l'elenco di tutti i gruppi coinvolti nel percorso di approvazione e se il redattore che ha approvato la richiesta appartiene ad uno di questi gruppi. Se l'operazione è stata rifiutata viene generato un messaggio per il giornalista della pagina. Se l'operazione è invece stata approvata, si va a cercare quale sia il gruppo superiore che non ha una delega con fiducia e si manda al redattore di questo gruppo una richiesta di approvazione, mentre ai redattori dei gruppi inferiori viene mandato un messaggio informativo. Ai redattori dei gruppi che hanno dato una delega con fiducia viene mandato un messaggio informativo sull'avvenuta approvazione. Se il redattore che ha effettuato l'approvazione è quello del gruppo radice, oppure tutte le deleghe tra il suo gruppo e quello radice sono con fiducia si procede alle operazioni di aggiornamento delle pagine, in modo da mandare in linea la revisione approvata o di mandare *offline* la revisione precedente, oltre ad aggiornare gli indici del motore di ricerca. Se la revisione approvata deve apparire in un tempo futuro lo stato della revisione viene opportunamente modificato. Questo *file* PHP è di fatto il cuore del sistema che gestisce il flusso di lavoro all'interno di questo CMS: è una procedura

abbastanza complessa sia per la necessità di gestire gli eventuali errori senza avere uno stato incoerente all'interno della base di dati, sia per l'intrinseca complessità del sistema di redazione distribuita.

`clock-ok.php` Questo *file* non è richiamato direttamente da sistema di gestione, ma serve per l'attivazione delle pagine a tempo e per effettuare alcune operazioni di manutenzione. Restituisce inoltre il contenuto delle tabelle che raccolgono informazioni sugli accessi e sulle richieste al motore di ricerca. In un sistema di produzione questa pagina deve essere richiamata da un comando eseguito ad intervalli regolari di tempo che scarichi il contenuto della pagina e lo memorizzi esternamente alla base di dati, oppure lo mandi via posta elettronica a chi si occupa di gestire le statistiche del sito.

B.3 Amministrazione del sistema

All'interno della *directory* `adm/su/` sono presenti i *file* PHP necessari all'amministrazione del sistema. Questa *directory* deve essere protetta utilizzando il meccanismo di autenticazione del *web server*.

`addgroup.php` Serve a generare la maschera per la creazione di un nuovo gruppo e successivamente di modificare il contenuto della base di dati.

`adduser.php` Serve a generare la maschera per la creazione di un nuovo utente ed inserire la sua *password* e successivamente di modificare il contenuto della base di dati.

`delgroup.php` Serve a generare la maschera per l'eliminazione di un gruppo e successivamente di modificare il contenuto della base di dati. Se viene violata l'integrità referenziale del *database* non sarà possibile eliminare il gruppo.

`delpg.php` Genera una tabella con l'elenco delle pagine e delle revisioni, simile alla tabella generata da `listpg.php`, ma dà la possibilità di andare a cancellare forzatamente pagine e revisioni.

`pgops.php` Chiamata da `delpg.php`, elimina la pagina indicata da `idpagina` e le revisioni ad essa associate.

`revops.php` Chiamata da `delpg.php`, se il parametro `mode` è uguale a "`DEL`", elimina la revisione indicata da `idrevisione`, mentre se parametro `mode` è uguale a "`ONL`", la revisione indicata da `idrevisione` viene rimessa forzatamente *online*.

`deluser.php` Serve a generare la maschera per l'eliminazione definitiva di un utente dal sistema di gestione. Successivamente viene eliminato l'utente e tutti i suoi dati dal sistema di gestione e le pagine dell'utente eliminato diventano di proprietà del redattore del nodo principale.

`disabuser.php` Serve a generare la maschera di disabilitazione o di abilitazione temporanea all'accesso del sistema di un utente e successivamente di modificare il contenuto della base di dati.

`makered.php` Serve a generare la maschera per modificare lo stato di redattore di un utente e successivamente di modificare il contenuto della base di dati. Non vengono effettuate verifiche sul numero di redattori presenti in un gruppo: deve essere cura dell'amministratore del sistema usare correttamente questa procedura.

`modgroup.php` Serve a generare la maschera per modificare il tipo di delega di un gruppo e quale sia il suo gruppo superiore. Successivamente va a modificare opportunamente il contenuto della base di dati.

`passwd.php` Serve a generare la maschera per l'inserimento di una nuova *password* e successivamente di modificare opportunamente la base di dati.

Appendice C

Struttura della base di dati

In questa parte verrà data una descrizione sintetica della struttura delle tabelle ed i loro campi presenti nel *database*, raggruppate per la loro funzione, cioè per la gestione di gruppi ed utenti, per la gestione dei messaggi, per la memorizzazione dei documenti XML che rappresentano il contenuto inserito, per il motore di ricerca e per le statistiche di accesso. Nella figura C.1 è riportato lo schema del *database*, escludendo le tabelle temporanee. Nella sezione C.6 vengono riportati i comandi necessari per creare la base di dati.

C.1 Tabelle per la gestione di gruppi ed utenti

Queste tabelle servono per definire gruppi ed utenti che possono creare e modificare contenuto e per gestire la procedura di autenticazione e di accesso.

`gruppo` Questa tabella permette di definire i vari gruppi presenti nella base di dati.

- `idgruppo` Numero del gruppo ed è la chiave primaria. Viene assegnato automaticamente. Il gruppo principale ha `idgruppo` uguale a zero.
- `idgruppo_r` Numero del gruppo padre. Per il gruppo principale è `NULL`.
- `fiducia` Vale "`t`" se il gruppo ha ricevuto una delega con fiducia, "`f`" altrimenti.
- `nome` Nome del gruppo.

`utente` Questa tabella contiene un elenco degli utilizzatori del sistema, cioè i redattori ed i giornalisti e memorizza i dati necessari al funzionamento del sistema di redazione distribuita.

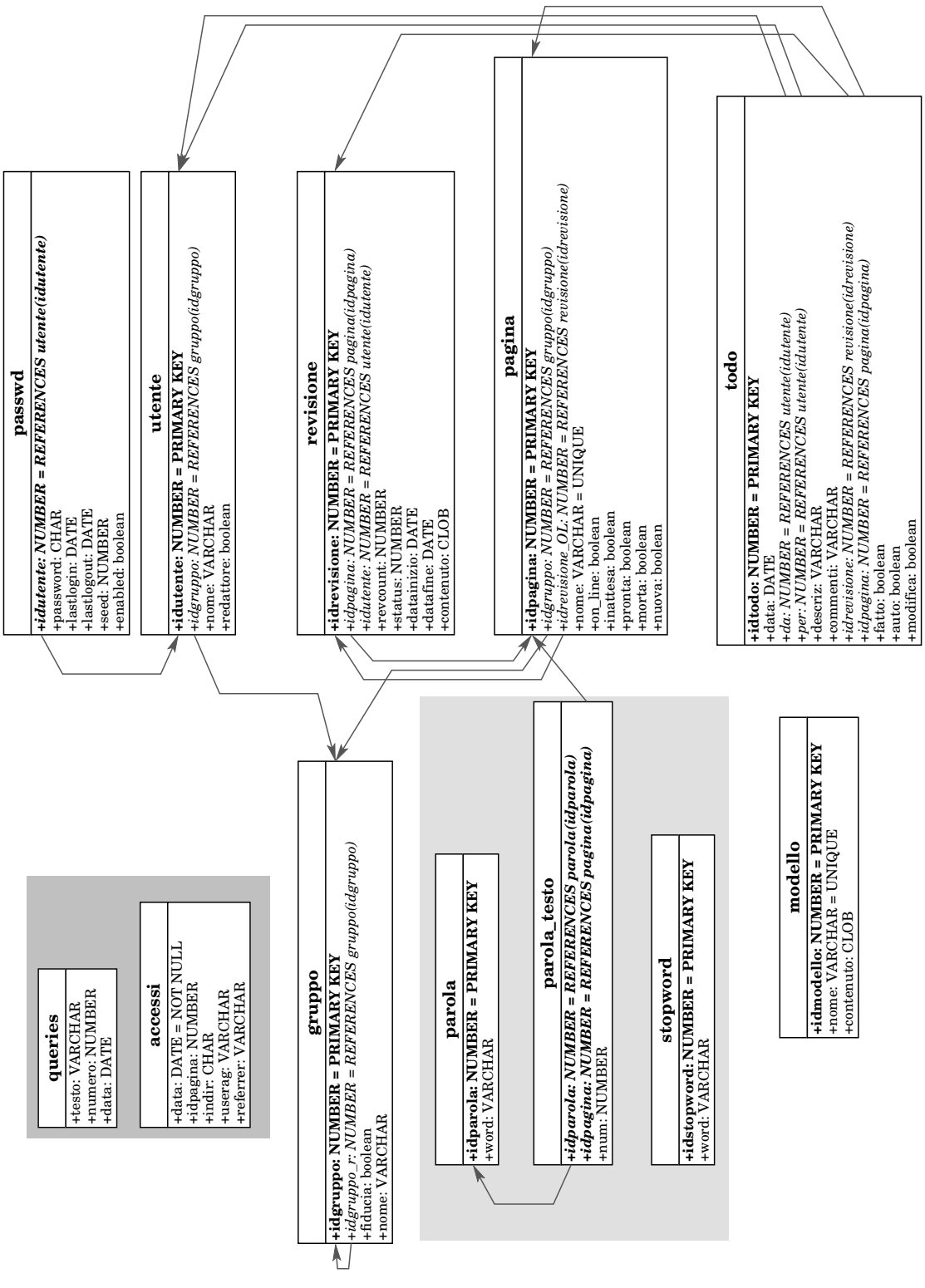


Figura C.1. Schema della base di dati

`idutente` Numero dell'utente, chiave primaria della tabella. Viene assegnato automaticamente. Il redattore del gruppo principale ha `idutente` uguale a 0.

`idgruppo` Numero del gruppo a cui l'utente appartiene.

`redattore` Vale "`t`" se l'utente è un redattore, "`f`" altrimenti.

`nome` Nome dell'utente.

`passwd` Questa tabella contiene i dati che permettono l'autenticazione e l'accesso degli utenti al sistema di gestione del contenuto. Questa tabella si potrebbe normalizzare ed unire con la tabella `utente`, ma in questo modo è possibile modificare questa tabella ed utilizzare sistemi più complessi di autenticazione senza andare a toccare l'altra tabella, utilizzata dal sistema di redazione distribuita.

`idutente` Numero dell'utente a cui si riferisce la riga della tabella: è la chiave debole della tabella

`password` Valore di *hash* MD5 della *password* dell'utente.

`lastlogin` Data di inizio dell'ultimo collegamento dell'utente. Viene visualizzata quando un utente si autentica con successo, in modo da poter avere una rapida verifica se sono stati effettuati accessi non autorizzati.

`lastlogout` Data della fine dell'ultimo collegamento dell'utente. Si noti che se si chiude il *browser* senza prima effettuare esplicitamente il *logout* oppure per qualche motivo la sessione scade, questo campo non viene aggiornato.

`seed` Numero che indica se un utente risulta collegato o meno.

`enabled` Abilitazione all'accesso: se l'utente può accedere al sistema di gestione vale "`t`" altrimenti vale "`f`".

C.2 Tabella per la gestione dei messaggi

Questa tabella serve per memorizzare i messaggi generati in maniera automatica o manuale che vengono visualizzati nell'elenco delle attività degli utenti.

`todo` È l'unica tabella che appartiene a questo gruppo.

`idtodo` Chiave primaria della tabella è una sequenza assegnata automaticamente

`data` Data del messaggio.

`da` Numero dell'utente che ha generato il messaggio

`per` Numero dell'utente a cui è destinato il messaggio

`descriz` Descrizione del messaggio.

`act` Codice a quattro lettere che indica il tipo di messaggio, in caso di messaggio generato automaticamente. "`CANC`" indica una richiesta di cancellazione, "`APPR`" indica una richiesta di approvazione, "`NO  `" se si tratta di un messaggio di rifiuto, "`AUTO`" se si tratta di una notifica di approvazione automatica.

`commenti` Commenti opzionali al messaggio

`idrevisione` In caso di richiesta di approvazione o di rifiuto contiene il numero di revisione a cui il messaggio si riferisce.

`idpagina` In caso di messaggio generato manualmente può contenere il numero di pagina a cui il messaggio si riferisce.

`fatto` Se è uguale a "t" l'attività deve essere ancora eseguita, e quindi viene visualizzato il messaggio.

`auto` Se è uguale a "t" significa che il messaggio è una notifica di approvazione automatica. Viene utilizzato un campo specifico, anziché andare ad utilizzare il campo `act` perché in questo modo si possono definire più tipi di messaggi relativi all'approvazione automatica.

`modifica` Se uguale a "`t`" appare ed `idrevisione` non è nullo, l'icona di modifica revisione all'interno delle attività.

C.3 Tabelle per la memorizzazione del contenuto

Queste tabelle servono a memorizzare il contenuto, sotto forma di documenti XML, all'interno del *database*.

`modello` Questa tabella è utilizzata per memorizzare i modelli di documento. Si noti che questa tabella non ha *foreign key* verso le altre tabelle perché non ha legami forti con altre tabelle.

`idmodello` Numero sequenziale del modello, chiave primaria della tabella. Viene generato automaticamente dal sistema.

`nome` Nome utilizzato per identificare il modello da parte degli utilizzatori. Due modelli diversi non possono avere lo stesso nome.

`contenuto` In questo CLOB è memorizzato il documento XML che rappresenta il modello di documento.

`pagina` Questa tabella viene utilizzata per memorizzare le pagine disponibili all'interno del sistema.

- `idpagina` Numero della pagina, assegnato automaticamente. Chiave primaria della tabella.
- `idgruppo` Numero del gruppo a cui appartiene la pagina.
- `idrevisione_ol` Numero della revisione il cui contenuto deve essere visualizzato quando viene richiesta l'erogazione della pagina. Se non c'è alcuna revisione da visualizzare questo campo contiene un `NULL`.
- `on_line` Vale "`t`" se la pagina ha una revisione disponibile da visualizzare, "`f`" altrimenti: se `idrevisione_ol` è `NULL` questo campo deve essere ad "`f`".
- `inattesa` Vale "`t`" se la pagina non ha una revisione disponibile da visualizzare, ma ne esiste una che è stata approvata definitivamente con un tempo di inizio validità futuro, "`f`" altrimenti.
- `pronta` Vale "`t`" se esiste almeno una revisione appartenete alla pagina che è stata approvata definitivamente e non è stata mandata *offline*, "`f`" altrimenti.
- `morta` Vale "`t`" se la pagina non ha revisioni disponibili, perché sono state mandate *offline*, "`f`" altrimenti.
- `nuova` Vale "`t`" se la pagina è stata creata e nessuna revisione è stata approvata definitivamente, "`f`" altrimenti.
- `nome` Nome della pagina. Due pagine diverse non possono avere lo stesso nome.

`revisione` Questa tabella memorizza il contenuto inserito all'interno del sistema di gestione.

- `idrevisione` Numero progressivo assegnato progressivamente in maniera automatica che identifica la revisione. Chiave primaria.
- `idpagina` Numero della pagina a cui appartiene la revisione
- `idutente` Numero dell'utente che ha creato la revisione.
- `revcount` Versione della revisione: la versione creata per ultima ha un valore maggiore, mentre quando viene creata la revisione associata ad una pagina nuova a questo campo viene assegnto il valore 1.
- `datainizio` Data di inizio validità del contenuto della revisione.
- `datafine` Data di fine validità della revisione.
- `contenuto` In questo campo viene memorizzato il documento XML che rappresenta il contenuto della revisione.

valore (hex)	stato	descrizione
1	CREATED	La revisione è stata creata da un modello
2	MODIFIED	La revisione è stata creata da una revisione
4	DELETED	La revisione ha ricevuto una richiesta di cancellazione
40	REWREQ	È stato richiesto di approvare la revisione
80	REDO	La revisione (o la sua cancellazione) è stata rifiutata
100	APPROVED	La revisione è stata approvata almeno una volta
200	READY	La revisione è stata approvata definitivamente
400	ONLINE	La revisione è disponibile ai visitatori del sito
800	ARCHIVED	La revisione è stata sostituita da una più recente
1000	KILLED	La revisione è stata mandata fuori linea, od è scaduta

Tabella C.1. Valore dei *flag* per una revisione

`status` Stato della revisione. Lo stato è rappresentato da un numero intero, dato dalla somma dei valori indicati nella tabella C.1. Una revisione creata da un modello che è stata approvata definitivamente ed è in linea, ha ad esempio lo stato 701 (esadecimale).

Si noti che lo stato delle pagine e delle revisioni associate è legato: ad esempio la revisione da visualizzare quando viene richiamata la pagina (indicata da `idrevisione_ol`) deve avere lo stato 701 o 702 esadecimale. La coerenza di questi valori deve essere garantita dall'applicazione: andare ad inserire dei *trigger* o delle *foreign key* avrebbe reso troppo complesso e rigido da gestire il *database*, rendendo molto difficoltose eventuali modifiche al flusso di lavoro.

C.4 Tabelle per il motore di ricerca

Queste tabelle vengono utilizzate per la memorizzazione delle parole e dell'*inverted index* che servono al funzionamento del motore di ricerca.

`parola` In questa tabella viene memorizzato l'elenco completo delle parole indicizzate.

`idparola` Chiave primaria della tabella, numero sequenziale assegnato automaticamente.

`word` Parola memorizzata.

`parola_testo` Questa tabella viene utilizzata per memorizzare gli *inverted index* per la ricerca testuale.

- `idparola` Riferimento alla parola presente nella tabella `parola`.
- `idpagina` Riferimento al numero di pagina in cui è contenuta la parola. Insieme ad `idparola` costituisce la *weak key* della tabella.
- `num` Peso della parola a cui si riferisce `idparola` all'interno della pagina a cui si riferisce `idpagina`.

`stopword` Questa tabella contiene un elenco di parole comuni che non vengono indicizzate dal motore di ricerca.

- `idstopword` Chiave primaria della tabella, numero sequenziale assegnato automaticamente.
- `word` Parola da non indicizzare.

Durante le fasi di ricerca e di indicizzazione vengono utilizzate due tabelle temporanee per la memorizzazione dei dati intermedi.

`tquery` Serve a memorizzare le parole inserite nella stringa che viene utilizzata per l'interrogazione al motore di ricerca.

`tw` Serve a memorizzare le parole trovate all'interno di una pagina ed il loro peso durante la fase di indicizzazione.

C.5 Tabelle per le statistiche di accesso

Queste tabelle servono a tenere traccia per scopi statistici degli accessi alle varie pagine e delle richieste effettuate al motore di ricerca. Non sono legate alle altre tabelle e non hanno una chiave primaria perché il loro scopo è quello di essere elaborate successivamente con altri strumenti e quindi si limitano a memorizzare dati grezzi.

`queries` Questa tabella memorizza le richieste effettuate dagli utenti al motore di ricerca.

`testo` In questa colonna viene memorizzato il testo della richiesta al motore di ricerca.

`numero` Rappresenta il numero di pagine totali restituite dal motore di ricerca.

`data` Data in cui è stata fatta la richiesta.

`accessi` Questa tabella memorizza i singoli accessi effettuati dai visitatori alle pagine.

`data` Data della richiesta HTTP.

`idpagina` Numero della pagina richiesta.

`indir` Indirizzo IP della macchina che ha richiesto la pagina.

`userag` Campo che memorizza la stringa che identifica il *browser* che ha generato la richiesta HTTP, se quest'ultimo ha inviato il campo `User_Agent`.

`referrer` Campo che memorizza la pagina da cui l'utilizzatore è arrivato, se il *browser* ha inviato nella richiesta il campo `Referrer`.

C.6 Definizione della base di dati in SQL

Quello che segue è la definizione, per l'RDBMS Oracle, di tabelle, *trigger* e funzioni della base di dati che viene utilizzata dal sistema di gestione del contenuto, oltre ai comandi per inserire un insieme minimo di dati necessario al funzionamento del sistema. Viene creato il gruppo principale, `root`, l'utente `root`, con *password* `root`, che è il redattore del gruppo principale, la pagina principale, il modello vuoto e due modelli di documento di esempio. Viene anche popolata la tabella `stopword` con alcuni termini comuni.

```
1  -- Definizione delle sequenze
2
3  CREATE SEQUENCE gruppo_idgruppo_seq;
4
5  CREATE SEQUENCE modello_idmodello_seq;
6
7  CREATE SEQUENCE pagina_idpagina_seq;
8
9  CREATE SEQUENCE parola_idparola_seq;
10
11 CREATE SEQUENCE revisione_idrevisione_seq;
12
13 CREATE SEQUENCE stopword_idstopword_seq;
14
15 CREATE SEQUENCE todo_idtodo_seq;
16
17 CREATE SEQUENCE utente_idutente_seq;
18
19 CREATE TABLE accessi
20 (
21         data       DATE            NOT NULL,
```

```
22          idpagina    NUMBER,
23          indir       CHAR(15),
24          userag      VARCHAR2(127),
25          referrer    VARCHAR2(255)
26  );
27
28  CREATE TRIGGER accessi_time
29  BEFORE INSERT ON accessi
30  FOR EACH ROW
31  begin
32   if :new.data is null then
33     select sysdate() into :new.data from dual;
34   end if;
35  end;
36  /
37
38  CREATE TABLE modello
39  (
40          idmodello   NUMBER          CONSTRAINT modello_pkey
41                                        PRIMARY KEY,
42          nome        VARCHAR2(64)    CONSTRAINT modello_nome
43                                        UNIQUE
44                                      NOT NULL,
45          contenuto   CLOB
46  );
47
48  CREATE TRIGGER idmodello_ser
49  BEFORE INSERT ON modello
50  FOR EACH ROW
51  begin
52   if :new.idmodello is null then
53     select modello_idmodello_seq.nextval into :new.idmodello from dual;
54   end if;
55  end;
56  /
57
58  CREATE TABLE gruppo
59  (
60          idgruppo     NUMBER          CONSTRAINT gruppo_pkey
61                                         PRIMARY KEY,
62          idgruppo_r   NUMBER          CONSTRAINT gruppo_gruppo
63                                         REFERENCES gruppo(idgruppo),
64          fiducia      CHAR(1)         DEFAULT 'f'
65                                       NOT NULL,
66          nome         VARCHAR2(64)
67  );
68
69  --  La tabella 'utente' ha una CONSTRAINT chiamata 'utente_gruppo'
70  --  che si riferisce alla tabella precedente
```

```
71  --  La tabella 'gruppo' ha una CONSTRAINT chiamata 'gruppo_gruppo'
72  --  che si riferisce alla tabella precedente
73  --  La tabella 'pagina' ha una CONSTRAINT chiamata 'pagina_gruppo'
74  --  che si riferisce alla tabella precedente
75
76  CREATE TABLE utente
77  (
78          idutente    NUMBER          CONSTRAINT utente_pkey
79                                        PRIMARY KEY,
80          idgruppo    NUMBER          CONSTRAINT utente_gruppo
81                                        REFERENCES gruppo(idgruppo),
82          redattore   CHAR(1)         DEFAULT 'f'
83                                      NOT NULL,
84          nome        VARCHAR2(255)
85  );
86
87  --  La tabella 'revisione' ha una CONSTRAINT chiamata 'revisione_utente'
88  --  che si riferisce alla tabella precedente
89  --  La tabella 'todo' ha una CONSTRAINT chiamata 'todo_utente_da'
90  --  che si riferisce alla tabella precedente
91  --  La tabella 'todo' ha una CONSTRAINT chiamata 'todo_utente_per'
92  --  che si riferisce alla tabella precedente
93
94  CREATE TRIGGER idutente_ser
95  BEFORE INSERT ON utente
96  FOR EACH ROW
97  begin
98   if :new.idutente is null then
99    select utente_idutente_seq.nextval into :new.idutente from dual;
100  end if;
101 end;
102 /
103
104
105 CREATE TRIGGER idgruppo_ser
106 BEFORE INSERT ON gruppo
107 FOR EACH ROW
108 begin
109  if :new.idgruppo is null then
110    select gruppo_idgruppo_seq.nextval into :new.idgruppo from dual;
111  end if;
112 end;
113 /
114
115 CREATE TABLE pagina
116 (
117         idpagina         NUMBER         CONSTRAINT pagina_pkey
118                                           PRIMARY KEY,
119         idgruppo         NUMBER         CONSTRAINT pagina_gruppo
```

```
120                                             REFERENCES gruppo(idgruppo),
121         idrevisione_ol   NUMBER,
122         on_line          CHAR(1),
123         inattesa         CHAR(1),
124         pronta           CHAR(1),
125         morta            CHAR(1),
126         nuova            CHAR(1),
127         nome             VARCHAR2(64)   CONSTRAINT pagina_nome
128                                           UNIQUE
129 );
130
131 --  La tabella 'revisione' ha una CONSTRAINT chiamata 'revisione_pagina'
132 --  che si riferisce alla tabella precedente
133 --  La tabella 'todo' ha una CONSTRAINT chiamata 'todo_pagina'
134 --  che si riferisce alla tabella precedente
135 --  La tabella 'parola_testo' ha una CONSTRAINT chiamata 'parola_testo_pagina'
136 --  che si riferisce alla tabella precedente
137
138 CREATE TRIGGER idpagina_ser
139 BEFORE INSERT ON pagina
140 FOR EACH ROW
141 begin
142  if :new.idpagina is null then
143    select pagina_idpagina_seq.nextval into :new.idpagina from dual;
144  end if;
145 end;
146 /
147
148 CREATE TABLE revisione
149 (
150         idrevisione   NUMBER   CONSTRAINT idrevisione_pkey
151                                  PRIMARY KEY,
152         idpagina      NUMBER   CONSTRAINT revisione_pagina
153                                  REFERENCES pagina(idpagina)
154                                    ON DELETE CASCADE,
155         idutente      NUMBER   CONSTRAINT revisione_utente
156                                  REFERENCES utente(idutente)
157                                    ON DELETE SET NULL,
158         revcount      NUMBER   DEFAULT 0
159                                NOT NULL,
160         datainizio    DATE,
161         datafine      DATE,
162         contenuto     CLOB,
163         status        NUMBER   DEFAULT 0
164                                NOT NULL
165 );
166
167 --  La tabella 'todo' ha una CONSTRAINT chiamata 'todo_revisione'
168 --  che si riferisce alla tabella precedente
```

```
169 --  La tabella 'pagina' ha una CONSTRAINT chiamata 'pagina_revisione'
170 --  che si riferisce alla tabella precedente
171
172 ALTER TABLE pagina ADD CONSTRAINT pagina_revisione
173                       FOREIGN KEY (idrevisione_ol) REFERENCES
174                       revisione(idrevisione) ON DELETE SET NULL;
175
176 CREATE TABLE parola
177 (
178         idparola   NUMBER          CONSTRAINT parola_pkey
179                                      PRIMARY KEY
180                                    NOT NULL,
181         word       VARCHAR2(255)   NOT NULL
182 );
183
184 CREATE TRIGGER idparola_ser
185 BEFORE INSERT ON parola
186 FOR EACH ROW
187 begin
188  if  :new.idparola is null then
189    select parola_idparola_seq.nextval into :new.idparola from dual;
190  end if;
191 end;
192 /
193
194 CREATE TABLE parola_testo
195 (
196         idparola   NUMBER   CONSTRAINT parola_testo_parola
197                               REFERENCES parola(idparola)
198                                 ON DELETE CASCADE
199                                 DEFERRABLE
200                                 INITIALLY DEFERRED,
201         idpagina   NUMBER   CONSTRAINT parola_testo_pagina
202                               REFERENCES pagina(idpagina)
203                                 ON DELETE CASCADE
204                                 DEFERRABLE
205                                 INITIALLY DEFERRED
206                             NOT NULL,
207         num        NUMBER   DEFAULT 1
208 );
209
210 CREATE TABLE passwd
211 (
212         idutente     NUMBER     CONSTRAINT passwd_utente
213                                   REFERENCES utente(idutente)
214                                     ON DELETE CASCADE,
215         password     CHAR(32)   NOT NULL,
216         lastlogin    DATE,
217         lastlogout   DATE,
```

```
218         seed         NUMBER,
219         enabled      CHAR(1)    DEFAULT 't'
220                                 NOT NULL
221 );
222
223 CREATE TABLE queries
224 (
225         testo    VARCHAR2(255),
226         numero   NUMBER,
227         data     DATE            NOT NULL
228 );
229
230 CREATE TRIGGER queries_time
231 BEFORE INSERT ON queries
232 FOR EACH ROW
233 begin
234  if :new.data is null then
235    select sysdate() into :new.data from dual;
236  end if;
237 end;
238 /
239
240 CREATE TRIGGER idrevisione_ser
241 BEFORE INSERT ON revisione
242 FOR EACH ROW
243 begin
244  if  :new.idrevisione is null then
245    select revisione_idrevisione_seq.nextval into :new.idrevisione from dual;
246  end if;
247 end;
248 /
249
250 CREATE TABLE stopword
251 (
252         idstopword   NUMBER          CONSTRAINT stopword_pkey
253                                        PRIMARY KEY
254                                      NOT NULL,
255         word         VARCHAR2(255)
256 );
257
258 CREATE TRIGGER idstopword_ser
259 BEFORE INSERT ON stopword
260 FOR EACH ROW
261 begin
262  if  :new.idstopword is null then
263    select stopword_idstopword_seq.nextval into :new.idstopword from dual;
264  end if;
265 end;
266 /
```

```
267
268 CREATE TABLE todo
269 (
270         idtodo        NUMBER           CONSTRAINT idtodo_pkey
271                                          PRIMARY KEY,
272         data          DATE,
273         da            NUMBER           CONSTRAINT todo_utente_da
274                                          REFERENCES utente(idutente)
275                                            ON DELETE SET NULL,
276         per           NUMBER           CONSTRAINT todo_utente_per
277                                          REFERENCES utente(idutente)
278                                            ON DELETE SET NULL,
279         descriz       VARCHAR2(255),
280         act           CHAR(4),
281         commenti      VARCHAR2(2047),
282         idrevisione   NUMBER           CONSTRAINT todo_revisione
283                                          REFERENCES revisione(idrevisione)
284                                            ON DELETE SET NULL,
285         idpagina      NUMBER           CONSTRAINT todo_pagina
286                                          REFERENCES pagina(idpagina)
287                                            ON DELETE SET NULL,
288         fatto         CHAR(1)          DEFAULT 'f'
289                                        NOT NULL,
290         auto          CHAR(1)          DEFAULT 'f'
291                                        NOT NULL,
292         modifica      CHAR(1)
293 );
294
295 CREATE TRIGGER idtodo_ser
296 BEFORE INSERT ON todo
297 FOR EACH ROW
298 begin
299  if  :new.idtodo is null then
300    select todo_idtodo_seq.nextval into :new.idtodo from dual;
301  end if;
302 end;
303 /
304
305 CREATE GLOBAL TEMPORARY TABLE tquery
306 (
307         idparola   NUMBER,
308         word       VARCHAR2(255),
309         tf         NUMBER          DEFAULT 1
310 );
311
312 CREATE GLOBAL TEMPORARY TABLE tw
313 (
314         a      VARCHAR2(255),
315         peso   NUMBER          DEFAULT 1
```

```
316 );
317
318
319 CREATE TRIGGER accessi_time
320 BEFORE INSERT ON accessi
321 FOR EACH ROW
322 begin
323  if :new.data is null then
324    select sysdate() into :new.data from dual;
325  end if;
326 end;
327 /
328
329 CREATE TRIGGER idgruppo_ser
330 BEFORE INSERT ON gruppo
331 FOR EACH ROW
332 begin
333  if :new.idgruppo is null then
334    select gruppo_idgruppo_seq.nextval into :new.idgruppo from dual;
335  end if;
336 end;
337 /
338
339 CREATE TRIGGER idmodello_ser
340 BEFORE INSERT ON modello
341 FOR EACH ROW
342 begin
343  if :new.idmodello is null then
344    select modello_idmodello_seq.nextval into :new.idmodello from dual;
345  end if;
346 end;
347 /
348
349 CREATE TRIGGER idpagina_ser
350 BEFORE INSERT ON pagina
351 FOR EACH ROW
352 begin
353  if :new.idpagina is null then
354    select pagina_idpagina_seq.nextval into :new.idpagina from dual;
355  end if;
356 end;
357 /
358
359 CREATE TRIGGER idparola_ser
360 BEFORE INSERT ON parola
361 FOR EACH ROW
362 begin
363  if  :new.idparola is null then
364    select parola_idparola_seq.nextval into :new.idparola from dual;
```

```
365  end if;
366 end;
367 /
368
369 CREATE TRIGGER idrevisione_ser
370 BEFORE INSERT ON revisione
371 FOR EACH ROW
372 begin
373  if  :new.idrevisione is null then
374    select revisione_idrevisione_seq.nextval into :new.idrevisione from dual;
375  end if;
376 end;
377 /
378
379 CREATE TRIGGER idstopword_ser
380 BEFORE INSERT ON stopword
381 FOR EACH ROW
382 begin
383  if  :new.idstopword is null then
384    select stopword_idstopword_seq.nextval into :new.idstopword from dual;
385  end if;
386 end;
387 /
388
389 CREATE TRIGGER idtodo_ser
390 BEFORE INSERT ON todo
391 FOR EACH ROW
392 begin
393  if  :new.idtodo is null then
394    select todo_idtodo_seq.nextval into :new.idtodo from dual;
395  end if;
396 end;
397 /
398
399 CREATE TRIGGER idutente_ser
400 BEFORE INSERT ON utente
401 FOR EACH ROW
402 begin
403  if :new.idutente is null then
404   select utente_idutente_seq.nextval into :new.idutente from dual;
405  end if;
406 end;
407 /
408
409 CREATE TRIGGER queries_time
410 BEFORE INSERT ON queries
411 FOR EACH ROW
412 begin
413  if :new.data is null then
```

```
414     select sysdate() into :new.data from dual;
415   end if;
416 end;
417 /
418
419
420 CREATE FUNCTION bitor( x IN NUMBER, y IN NUMBER ) RETURN NUMBER  AS
421 BEGIN
422     RETURN x + y - bitand(x,y);
423 END;
424 /
425
426 CREATE FUNCTION bitxor( x IN NUMBER, y IN NUMBER ) RETURN NUMBER  AS
427 BEGIN
428     RETURN bitor(x,y) - bitand(x,y);
429 END;
430 /
431
432 -- Fine descrisione del database
433 -- Inserimento dati iniziali
434
435 INSERT INTO gruppo
436         (idgruppo,idgruppo_r,fiducia,nome) VALUES
437         (0,NULL,'t','root');
438 UPDATE gruppo SET idgruppo=0 WHERE nome='root';
439
440 INSERT INTO utente
441         (idutente,idgruppo,redattore,nome) VALUES
442         (0,0,'t','root');
443 UPDATE utente SET idutente=0 WHERE nome='root';
444
445 INSERT INTO passwd
446         (idutente,password,enabled) VALUES
447         (0,'63a9f0ea7bb98050796b649e85481845','t');
448
449 INSERT INTO pagina
450         (idpagina,idgruppo,on_line,inattesa,pronta,morta,nuova,nome)
451         VALUES (0,0, 'f', 'f', 'f', 'f', 'f','HOME*HOME PAGE');
452 UPDATE pagina SET idpagina=0 WHERE nome='HOME*HOME PAGE';
453
454 INSERT INTO modello
455 VALUES (0,'Modello vuoto','<?xml version="1.0" ?>
456 <!DOCTYPE pass SYSTEM "pass.dtd">
457 <pass>
458 <head><title></title><summary></summary><keywords></keywords></head>
459 <body><div style="titolo1" altstyle="h1">
460 <text bullet="none" clone="no" inline="no"></text></div></body>
461 </pass>');
462 UPDATE modello SET idmodello=0 WHERE nome='Modello vuoto';
```

```
463
464 INSERT INTO modello (nome,contenuto)
465 VALUES ('Testo','<?xml version="1.0" ?>
466 <!DOCTYPE pass SYSTEM "pass.dtd">
467 <pass>
468 <head><title></title><summary></summary><keywords></keywords>
469 </head>
470 <body>
471 <div style="titolo1" altstyle="h1">
472 <text bullet="none" clone="no" inline="no"></text>
473 </div>
474 <div style="normale" altstyle="h1">
475 <text bullet="none" clone="yes" inline="no"></text>
476 </div>
477 </body>
478 </pass>');
479
480 INSERT INTO modello (nome,contenuto)
481  VALUES ('pagina Link','<?xml version = "1.0" encoding="ISO-8859-1" ?>
482 <!DOCTYPE pass SYSTEM "pass.dtd">
483 <pass>
484 <head>
485 <title></title><summary></summary><keywords></keywords>
486 </head>
487 <body>
488 <div style="titolo1" altstyle="h1">
489 <image bullet="none" inline="no" clone="no"
490 uri="img/infohand.jpg" alt="InfoHandicap" />
491 </div>
492 <div style="normale" altstyle="">
493 <link bullet="none" inline="no" clone="yes" visible="no"
494  type="external" uri=""></link>
495 <link bullet="none" inline="no" clone="yes" visible="no"
496  type="external" uri=""></link>
497 <link bullet="none" inline="no" clone="yes" visible="no"
498  type="external" uri=""> </link>
499 <link bullet="none" inline="no" clone="yes" visible="no"
500  type="external" uri=""></link>
501 <link bullet="none" inline="no" clone="yes" visible="no"
502  type="external" uri=""></link>
503 <link bullet="none" inline="no" clone="yes" visible="no"
504  type="external" uri=""></link>
505 <link bullet="none" inline="no" clone="yes" visible="no"
506  type="external" uri=""></link>
507 <link bullet="none" inline="no" clone="yes" visible="no"
508  type="external" uri=""></link>
509 <link bullet="none" inline="no" clone="yes" visible="no"
510  type="external" uri=""></link>
511 <link bullet="none" inline="no" clone="yes" visible="no"
```

```
 type="external" uri=""></link>
<link bullet="none" inline="no" clone="yes" visible="no"
 type="external" uri=""></link>
<link bullet="none" inline="no" clone="yes" visible="no"
 type="external" uri=""></link>
<link bullet="none" inline="no" clone="yes" visible="no"
 type="external" uri=""></link>
<link bullet="none" inline="no" clone="yes" visible="no"
 type="external" uri=""></link>
</div>
</body>
</pass>');

INSERT INTO stopword (word)  VALUES ('a');
INSERT INTO stopword (word)  VALUES ('ad');
INSERT INTO stopword (word)  VALUES ('agli');
INSERT INTO stopword (word)  VALUES ('al');
INSERT INTO stopword (word)  VALUES ('all');
INSERT INTO stopword (word)  VALUES ('alla');
INSERT INTO stopword (word)  VALUES ('altra');
INSERT INTO stopword (word)  VALUES ('altri');
INSERT INTO stopword (word)  VALUES ('altro');
INSERT INTO stopword (word)  VALUES ('anche');
INSERT INTO stopword (word)  VALUES ('anzi');
INSERT INTO stopword (word)  VALUES ('ben');
INSERT INTO stopword (word)  VALUES ('che');
INSERT INTO stopword (word)  VALUES ('chi');
INSERT INTO stopword (word)  VALUES ('col');
INSERT INTO stopword (word)  VALUES ('coll');
INSERT INTO stopword (word)  VALUES ('come');
INSERT INTO stopword (word)  VALUES ('con');
INSERT INTO stopword (word)  VALUES ('cosicché');
INSERT INTO stopword (word)  VALUES ('da');
INSERT INTO stopword (word)  VALUES ('dai');
INSERT INTO stopword (word)  VALUES ('dall');
INSERT INTO stopword (word)  VALUES ('degli');
INSERT INTO stopword (word)  VALUES ('dei');
INSERT INTO stopword (word)  VALUES ('del');
INSERT INTO stopword (word)  VALUES ('dell');
INSERT INTO stopword (word)  VALUES ('delle');
INSERT INTO stopword (word)  VALUES ('deve');
INSERT INTO stopword (word)  VALUES ('di');
INSERT INTO stopword (word)  VALUES ('e');
INSERT INTO stopword (word)  VALUES ('ed');
INSERT INTO stopword (word)  VALUES ('era');
INSERT INTO stopword (word)  VALUES ('essi');
INSERT INTO stopword (word)  VALUES ('fra');
INSERT INTO stopword (word)  VALUES ('fuori');
INSERT INTO stopword (word)  VALUES ('gli');
```

```
561 INSERT INTO stopword (word)  VALUES ('ha');
562 INSERT INTO stopword (word)  VALUES ('hai');
563 INSERT INTO stopword (word)  VALUES ('ho');
564 INSERT INTO stopword (word)  VALUES ('i');
565 INSERT INTO stopword (word)  VALUES ('il');
566 INSERT INTO stopword (word)  VALUES ('in');
567 INSERT INTO stopword (word)  VALUES ('io');
568 INSERT INTO stopword (word)  VALUES ('l');
569 INSERT INTO stopword (word)  VALUES ('la');
570 INSERT INTO stopword (word)  VALUES ('le');
571 INSERT INTO stopword (word)  VALUES ('lei');
572 INSERT INTO stopword (word)  VALUES ('lo');
573 INSERT INTO stopword (word)  VALUES ('loro');
574 INSERT INTO stopword (word)  VALUES ('lui');
575 INSERT INTO stopword (word)  VALUES ('modalità');
576 INSERT INTO stopword (word)  VALUES ('ne');
577 INSERT INTO stopword (word)  VALUES ('necessità');
578 INSERT INTO stopword (word)  VALUES ('nei');
579 INSERT INTO stopword (word)  VALUES ('nel');
580 INSERT INTO stopword (word)  VALUES ('nell');
581 INSERT INTO stopword (word)  VALUES ('nella');
582 INSERT INTO stopword (word)  VALUES ('no');
583 INSERT INTO stopword (word)  VALUES ('noi');
584 INSERT INTO stopword (word)  VALUES ('non');
585 INSERT INTO stopword (word)  VALUES ('nè');
586 INSERT INTO stopword (word)  VALUES ('o');
587 INSERT INTO stopword (word)  VALUES ('ore');
588 INSERT INTO stopword (word)  VALUES ('parità');
589 INSERT INTO stopword (word)  VALUES ('per');
590 INSERT INTO stopword (word)  VALUES ('percui');
591 INSERT INTO stopword (word)  VALUES ('perche');
592 INSERT INTO stopword (word)  VALUES ('perchè');
593 INSERT INTO stopword (word)  VALUES ('perché');
594 INSERT INTO stopword (word)  VALUES ('pi');
595 INSERT INTO stopword (word)  VALUES ('piu');
596 INSERT INTO stopword (word)  VALUES ('più');
597 INSERT INTO stopword (word)  VALUES ('poiché');
598 INSERT INTO stopword (word)  VALUES ('possibilità');
599 INSERT INTO stopword (word)  VALUES ('poter');
600 INSERT INTO stopword (word)  VALUES ('pu');
601 INSERT INTO stopword (word)  VALUES ('quantità');
602 INSERT INTO stopword (word)  VALUES ('quella');
603 INSERT INTO stopword (word)  VALUES ('quelle');
604 INSERT INTO stopword (word)  VALUES ('quelli');
605 INSERT INTO stopword (word)  VALUES ('quello');
606 INSERT INTO stopword (word)  VALUES ('questa');
607 INSERT INTO stopword (word)  VALUES ('queste');
608 INSERT INTO stopword (word)  VALUES ('questo');
609 INSERT INTO stopword (word)  VALUES ('qui');
```

```
610 INSERT INTO stopword (word)  VALUES ('se');
611 INSERT INTO stopword (word)  VALUES ('si');
612 INSERT INTO stopword (word)  VALUES ('solo');
613 INSERT INTO stopword (word)  VALUES ('sono');
614 INSERT INTO stopword (word)  VALUES ('sta');
615 INSERT INTO stopword (word)  VALUES ('su');
616 INSERT INTO stopword (word)  VALUES ('sul');
617 INSERT INTO stopword (word)  VALUES ('sull');
618 INSERT INTO stopword (word)  VALUES ('sulla');
619 INSERT INTO stopword (word)  VALUES ('suoi');
620 INSERT INTO stopword (word)  VALUES ('sè');
621 INSERT INTO stopword (word)  VALUES ('sé');
622 INSERT INTO stopword (word)  VALUES ('sì');
623 INSERT INTO stopword (word)  VALUES ('tra');
624 INSERT INTO stopword (word)  VALUES ('tu');
625 INSERT INTO stopword (word)  VALUES ('un');
626 INSERT INTO stopword (word)  VALUES ('una');
627 INSERT INTO stopword (word)  VALUES ('uno');
628 INSERT INTO stopword (word)  VALUES ('uso');
629 INSERT INTO stopword (word)  VALUES ('validità');
630 INSERT INTO stopword (word)  VALUES ('versatilità');
631 INSERT INTO stopword (word)  VALUES ('voi');
632 INSERT INTO stopword (word)  VALUES ('è');
633
634 COMMIT;
```

Appendice D

Procedura di installazione

L'installazione del sistema di gestione del contenuto richiede alcune operazioni preliminari per il corretto funzionamento, che verrando descritte qui di seguito.

Il sistema di gestione del contenuto è scritto interamente in PHP. Per poter utilizzare correttamente il sistema è necessario che il *web server* su cui si vanno ad installare i *file* utilizzi Apache (versione 1.3.26 o superiore) con il supporto per il PHP (versione 4.1.2 o superiore), e che il PHP sia compilato con le funzioni per il collegamento al *database* Oracle8 ed abbia le funzioni relative al *parser* SAX attivate. Per l'accesso alla base di dati si utilizzano le classi fornite dalla libreria ADODB (che si trova nella *directory* `adodb/`).

Bisogna impostare alcune variabili di ambiente prima dell'esecuzione di Apache, per fare in modo che PHP riesca a collegarsi al *server* di *database* Oracle correttamente. Si suppone che la configurazione per l'accesso al *database* sia stata già effettuata. Sul *web server* deve essere presente e funzionante la parte *client* per il *database*, in particolare `TNSNAMES.ORA` deve essere correttamente configurato.

Per quanto riguarda invece il *server* che ospita il *database* Oracle, deve esistere un utente che possa essere utilizzato dal *web server* e devono essere create le tabelle ed i *trigger* necessari: si può sia utilizzare `SQL*PLUS` inserendo le istruzioni SQL indicate nell'appendice C.6, che utilizzare l'utility `imp` per importare il *database* di esempio.

Un esempio di quali sono le variabili di ambiente necessarie è questo:

```
export ORACLE_BASE=/oracle
export ORACLE_HOME=/oracle/product/9.2.0
export ORACLE_SID=test
export ORACLE_TERM=xterm
export NLS_LANG=ITALIAN.WE8ISO8859P1
export ORA_NLS33=$ORACLE_HOME/ocommon/nls/admin/data
LD_LIBRARY_PATH=$ORACLE_HOME/lib
export LD_LIBRARY_PATH
export PATH=$PATH:$ORACLE_HOME/bin
```

È necessario anche aver configurato correttamente `httpd.conf`. Si presti attenzione che se viene utilizzato una versione di PHP compilata come CGI è necessario passare le variabili d'ambiente anche all'eseguibile. Se si utilizza l'autenticazione semplice di HTTP bisogna configurarla qui, oppure utilizzare correttamente la direttiva `AllowOverride`, come nell'esempio che segue.

```
<Directory /var/www/pass2ora>
  Options Indexes Includes FollowSymLinks MultiViews
  AllowOverride AuthConfig
  PassEnv ORACLE_BASE ORACLE_HOME ORACLE_SID NLS_LANG ORA_NLS33 LD_LIBRARY_PATH \
          ORACLE_TERM
  Order allow,deny
  Allow from all
  Action php-script /cgi-bin/php
  AddHandler php-script .php
</Directory>
```

Infine è necessario andare a configurare le variabili all'interno di `mydef.php` in modo che si adattino alla struttura del sistema.

`host` è l'indirizzo della macchina che ospita il *database*

`version` è una stringa che appare nel titolo delle pagine per la gestione del sito, nel motore di ricerca e nella mappa del sito

`webhost` è l'indirizzo della *home directory* dove è stato installato il sistema di gestione

`ext_edit` se a 1 permette al redattore del nodo radice di usare l'*editor* dei modelli anche per modificare le revisioni

`username` nome dell'utente del database

`password` password dell'utente del database

`aliasdb` nome del database a cui collegarsi

Un esempio di una configurazione corretta del sistema è questo:

```
$host = "oracleserver";
$version = "00-Oracle9i";
$build = 45;
$webhost = "http://webserver/pass/";
$ext_edit = 0;
$username = "webuser";
$password = "1234567";
$aliasdb = "pass2";
```

Se tutto è stato fatto correttamente, chiamando pagina.php dovrebbe apparire un messaggio di "pagina non trovata" se le tabelle sono vuote, oppure la *home page* del sito di esempio.

Nel caso che il collegamento verso il *database* non riesca viene invece visualizzato un messaggio di errore del tipo: "`Warning: _oci_open_server: ORA-12541: TNS:no listener`".

Le cause che possono fare fallire il collegamento possono essere diverse: le variabili `host` e `aliasdb` possono essere sbagliate, oppure il problema può essere che `TNSNAMES.ORA` non esista o non sia scritto correttamente. Se le variabili d'ambiente sono sbagliate, e quindi il PHP non riesce a leggere `TNSNAMES.ORA`, si ha questo messaggio di errore. Se la macchina su cui è installato Oracle è diversa da quella su cui è installato Apache, può essersi verificato un problema di rete a più basso livello che impedisce ai due elaboratori di stabilire una connessione TCP/IP.

Bibliografia

[1] Robert Cailliau
A Little History of the World Wide Web
http://www.w3.org/History.html

[2] Gregory R. Gromov
History of Internet and WWW: The Roads and Crossroads of Internet History
http://www.netvalley.com/intval1.html

[3] http://www.metatorial.com

[4] Web accessibility Initiative
http://www.w3.org/WAI

[5] Bob Boiko
Why content management? — A CM Domain White Paper
Metatorial Services Inc./HungryMinds Inc.,2002
http://www.metatorial.com/Papers/iqpc_singapore.pdf

[6] Content Management Review of Market
Esprit Soutron Partnership, 2001
www.espritsoutronpartnership.com/support/downloads/pdf/articles/ContentManagev1.pdf

[7] Amos Latteiner, Mike Pelletier
Zope
Mc Graw-Hill Italia, 2002
ISBN 88-386-4235-4
Edizione Originale: The Zope Book — New Riders Publishing, 2002

[8] http://www.cmswatch.com/ContentManagement/Products/

[9] http://www.zope.org

[10] http://www.zope.com

[11] http://cmf.zope.org

[12] http://plone.org

[13] http://www.icoya.com

[14] http://www.opencms.org

[15] http://www.midgard-project.org

[16] http://www.cofax.org

[17] Erik T. Ray
Learning XML
O'Reilly, 2002
ISBN 0-596-00046-4

[18] Dan Livingston
XML
Techniche Nuove, 2002
ISBN 88-481-1389-3
Edizione Originale: Essential XML for Web Professionals — Perentice Hall, 2002

[19] `http://xml.coverpages.org/xml.html`
Per quanto riguarda le applicazioni disponibili `http://xml.coverpages.org/xml.html#techDesign`

[20] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen
Extensible Markup Language (XML) 1.0 — W3C Recommendation 10 February 1998
`http://www.w3.org/TR/1998/REC-xml-19980210`
Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen
Extensible Markup Language (XML) 1.0 (Second Edition) — W3C Recommendation 6 October 2000
`http://www.w3.org/TR/2000/REC-xml-20001006`

[21] S. Pemberton et al.
XHTML 1.0: The Extensible HyperText Markup Language — W3C Recommendation 26 January 2000.
`http://www.w3.org/TR/2000/REC-xhtml1-20000126`

[22] Murray Altheim, Frank Boumphrey et al.
Modularization of XHTML — W3C Recommendation 10 April 2001
`http://www.w3.org/TR/xhtml-modularization`

[23] Murray Altheim and Shane McCarron
XHTML 1.1 - Module-based XHTML — W3C Recommendation 31 May 2001
`http://www.w3.org/TR/xhtml11`

[24] Tim Bray
The Annotated XML Specification
`http://www.xml.com/axml/axml.html`

[25] David C. Fallside
XML Schema Part 0: Primer — W3C Recommendation, 2 May 2001
`http://www.w3.org/TR/xmlschema-0/`

[26] James Clark, MURATA Makoto
RELAX NG Tutorial — Committee Specification 3 December 2001
The Organization for the Advancement of Structured Information Standards [OASIS]

http://www.oasis-open.org/committees/relax-ng/tutorial-20011203.html

[27] Håkon Wium Lie,Bert Bos
Cascading Style Sheets, level 1 — W3C Recommendation 17 Dec 1996, revised 11 Jan 1999
http://www.w3.org/TR/REC-CSS1

[28] Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs
Cascading Style Sheets, level 2 CSS2 Specification — W3C Recommendation 12-May-1998
http://www.w3.org/TR/REC-CSS2

[29] James Clark
XSL Transformations (XSLT) Version 1.0 — W3C Recommendation 16 November 1999
http://www.w3.org/TR/xslt

[30] Marco Griva
Progetto e realizzazione di siti internet informativi: il Servizio Passepartout di informazione sull'handicap
Tesi di laurea, luglio 2001
http://elite.polito.it/tesi/griva.pdf