Università degli Studi di Padova
Dipartimento di Matematica Pura e Applicata
Corso di laurea in informatica

Anno Accademico 2010-2011

Progetto

Del corso di Tecnologie Web

De Pretto Filippo (referente)
Ferroli Giorgio

Indice

LiberLab

1. Abstract

Il progetto si propone di fornire all'utente un sito internet per la raccolta di libri codificati con la **grammatica TEI**. E' quindi possibile caricare nuovi libri, visualizzare quelli già caricati, ordinarli per numero di visite, per categoria e vedere tutte le categorie con i relativi libri.

L'idea alla base del sito è riassunta da **Fabio Brivio**, uno dei primi autori a pubblicare ebook per Apogeo.

> Manca […] un'applicazione client/server che permetta all'autore di lavorare sul suo libro condividendo, attraverso una rete dedicata, il suo lavoro con l'editore, che in questo modo potrebbe gestire con maggiore elasticità il lavoro di editing, ma non solo. L'applicazione […] dovrebbe salvare l'opera in un formato standard facilmente condivisibile per un suo agevole (ri)utilizzo: creare capitoli sample da distribuire in posta elettronica alle liste dei giornalisti interessati, creare pdf del libro da vendere in anteprima sul Web, o i file di stampa per lo stampatore. […] Il formato di codifica dovrebbe essere su base XML, e in questo senso soluzioni ci sono: TEI, DocBook o la soluzione allo studio dell'International Digital Publishing Forum.

Ovviamente il sito che abbiamo sviluppato è solo una versione base di tutto ciò, una sorta di "demo" per farne capire le potenzialità. L'obiettivo fondamentale del progetto, più che un proliferare di funzionalità, è quello di rendere il sito molto **veloce, accessibile** e in cui la **struttura**, il **contenuto**, il **comportamento** e il **layout** siano **separati** al massimo . Questo rende il sito facilmente traducibile ed è stato infatti **tradotto** anche **in inglese** per mostrare questa funzionalità (il contenuto dei libri è rimasto quello fisso della grammatica TEI in italiano). Queste caratteristiche sono possibili grazie dalla creazione offline della "base" delle pagine tramite lo script createHTML.pl e alla resa dinamica delle pagine permessa da view.cgi, tecniche che saranno illustrate in seguito.

In questo modo è stato possibile creare pagine dinamiche anche complesse che hanno un tempo di caricamento minimale. In particolare si passa dalle pagine praticamente statiche caricate in media in 0.01 secondi, utilizzando un PC fisso con CPU Intel i7 fino alle pagine “complesse", caricate in circa 0.08 secondi. Risultato che riteniamo soddisfacente visto che Google stima il "limite massimo" a circa 1.5 secondi.

Lo stesso progetto, realizzato utilizzando php+mysql per il corso di basi di dati, ha un tempo di caricamento ridotto di un fattore da due a quattro (a seconda della pagina) ~~essendo~~ poichè è necessaria, nel progetto per tecnologie web, l'apertura di più file XML per caricare i dati. Inoltre manca anche anche un servizio di caching efficiente che, non essendo l'obiettivo del progetto, non è stato implementato.

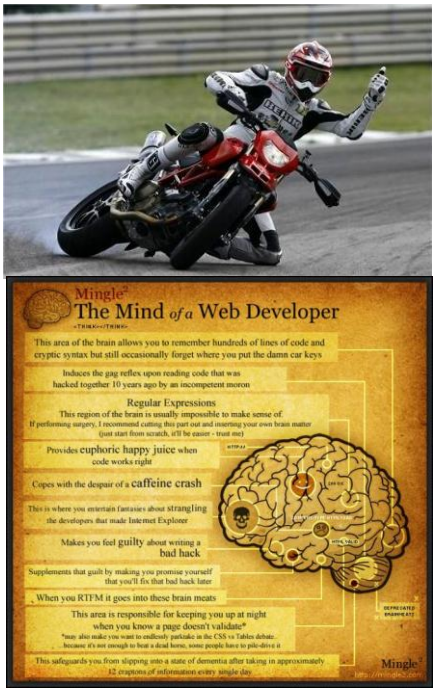
2. Analisi dell'utenza del sito

Come verrà spiegato nella parte dedicata alla Progettazione, il sito inizialmente era stato pensato per uno sviluppo da parte di **4 persone**. Pertanto l'analisi dell'utenza che avrebbe utilizzato il nostro sito era più vasta di quella che invece può utilizzare il sito allo stato attuale.

Infatti, per quanto la sua consultazione sia abbastanza semplice e intuitiva, l'inserimento di nuovi libri richiede la conoscenza della grammatica TEI per codificare correttamente i libri. Esistono anche degli script e delle procedure per convertire file da formati quali .doc e .odt in documenti xml codificati con la grammatica TEI, tuttavia non essendo stati implementati per il sito e non essendo facili da implementare, il "problema" permane.

Pertanto, rimangono principalmente tre fasce d'utenza:

1. In primis, vengono i **ricercatori**. Essendo una grammatica scritta apposta per codificare i documenti delle università e delle biblioteche, può tornare utile un sito che ne permetta l'immediata codifica in HTML e pubblicazione online. Inoltre dovrebbe essere una grammatica già conosciuta per molte società e associazioni quindi non richiederebbe di imparare nulla di nuovo. E' anche possibile organizzare i documenti in categorie (anche se, per semplicità di implementazione, si è supposto che un libro possa essere solo in una categoria) e vedere i più letti e gli ultimi inseriti. Per quanto sia solo una *demo*, il sito potrebbe essere già utilizzabile tranquillamente per questa fascia d'utenza.



2. La seconda fascia d'utenza che potrebbe essere interessata è quella dei **professionisti** e degli scrittori di ebooks. Questa soluzione permette infatti la codifica al volo del libro in HTML, formato convertibile in PDF e/o in epub e così via, senza troppa difficoltà con tool online e offline (es: http://html-pdf-converter.com/) pertanto potrebbe tornare comodo codificare il libro in questa grammatica utilizzando per esempio **Oxygen** (http://www.oxygenxml.com/) e pubblicare il libro online sul sito. Oxygen mette anche a disposizione i tool per le conversioni in pdf e nei formati più diffusi quindi si può usare anche questo strumento per la successiva pubblicazione in altri formati.

3. Può essere utile anche per gli **studenti** che possono imparare questa grammatica per pubblicare velocemente online libri autoprodotti e convertirli con Oxygen negli altri formati se necessario.

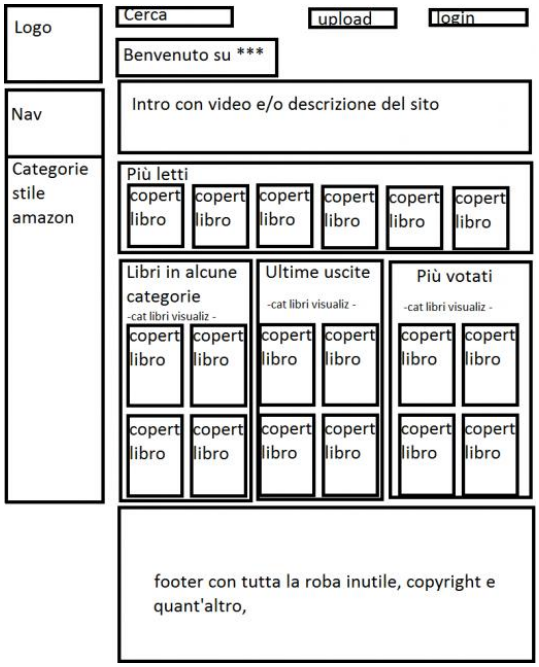
In sostanza, quindi, il sito è pensato per un'**utenza** piuttosto **varia anche se professionale**. Tuttavia è stato costruito utilizzando dei sistemi come il createHTML.pl e views.cgi che potrebbero essere facilmente riconvertiti in modo semplice per la **creazione di altri siti** accessibili, veloci, leggeri e fondamentalmente statici (ma anche dinamici, al costo di un lavoro aggiuntivo).

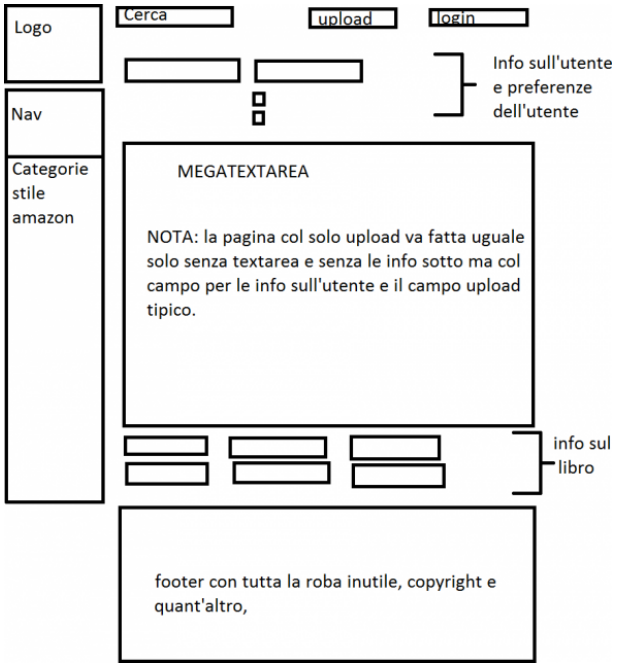
Il codice inoltre è commentato quasi "riga per riga" quindi questo processo può essere svolto tranquillamente anche da persone che non hanno partecipato allo sviluppo del progetto.

3. Progettazione

Il gruppo, costituito inizialmente da quattro persone, è stato suddiviso in due **sottogruppi**, HTML+CSS+JavaScript a Giorgio e Thomas, Perl+XML+XSLT a Filippo e Giuliano. In realtà Giorgio ha modificato e aggiornato l'XML e l'XSLT create da Filippo e viceversa Filippo ha creato delle pagine in HTML più dei CSS di base che è sono stati migliorati e ampliati da Giorgio.

Inizialmente, Filippo ha creato per il primo gruppo delle **work-map** per le pagine fondamentali del sito e ha deciso come impostare la parte Perl del sito, assegnandosi la parte off-line, i filtri per la grammatica TEI e assegnando a Giuliano l'**upload** e la funzione **cerca**.





In fase di progettazione, lo sviluppo del sito era stato pensato per 4 persone (infatti c'erano altri due work-map inizialmente), tuttavia, per i motivi che è possibile approfondire nella pagina "Storia di LiberLab" presente nel progetto, sono rimasti solo Filippo e Giorgio. Ciò ha portato quindi ad una riprogettazione del sito a lavoro in corso. Al momento dell'uscita di Thomas e Giuliano dal gruppo erano infatti già stati realizzati il createHTML, l'home page (statica), una versione base del view.cgi e il bookrotator.

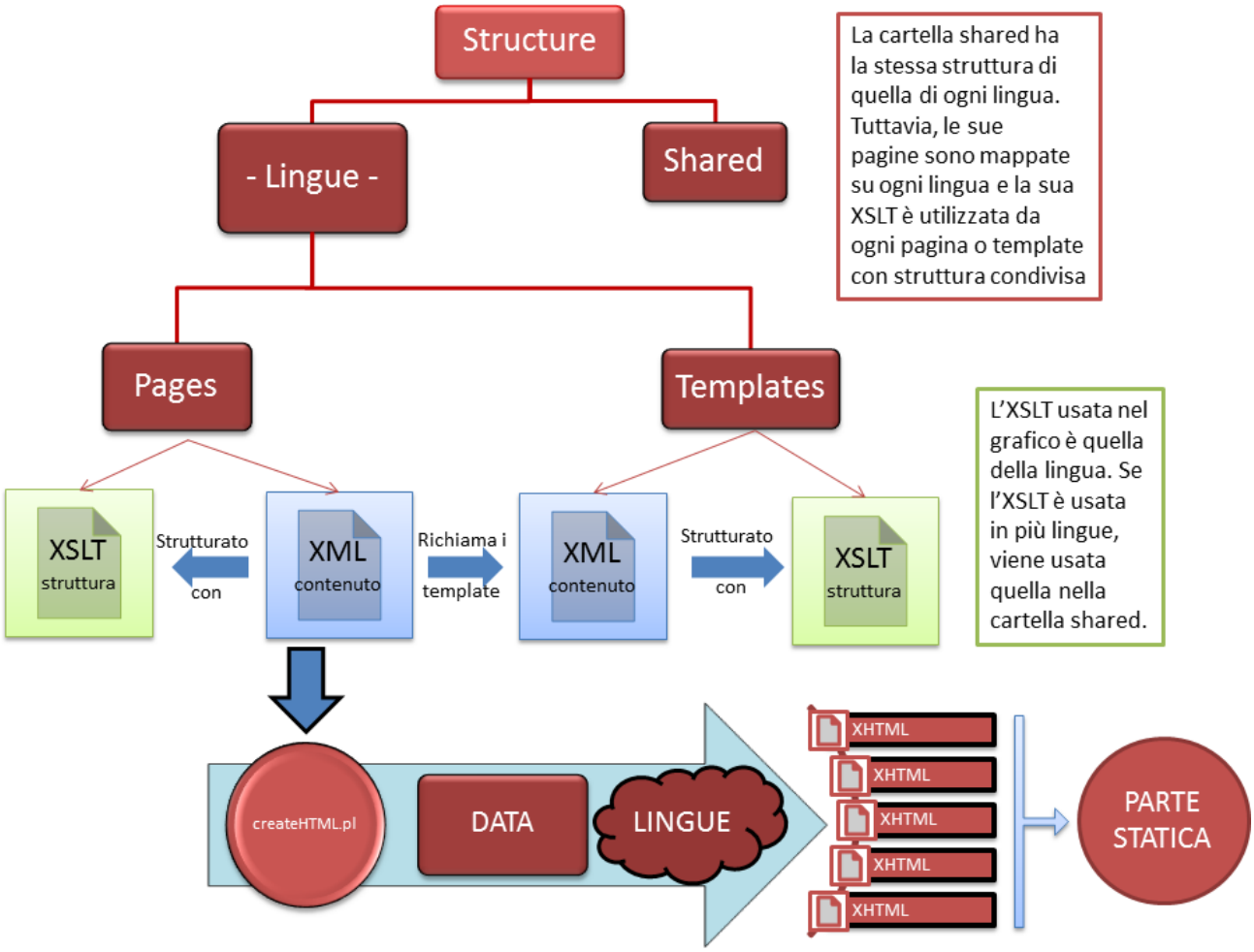
Pertanto, per quanto il bookrotator e il createHTML fossero meno importanti di funzionalità quali la possibilità di modificare i libri, farne l'upload, ricercare i libri già presenti sul sito e così via, abbiamo dovuto **riprogettare** il lavoro partendo da queste componenti già realizzate. Abbiamo quindi scelto di sviluppare l'upload, in quanto necessario, tralasciando l'opzione di modificare i libri già presenti e la funzionalità per cercare i libri. Il login, inoltre, per quanto non necessario per il sito (a riguardo esistono esempi commerciali anche di successo, quali pastebin), non era neanche strettamente richiesto, pertanto si è deciso di non implementarlo. Inoltre l'upload inizialmente prevedeva un **editor** semplificato per caricare i libri in formato TEI, mentre ora l'editor è un semplice **textarea**.

Similmente, per la parte HTML+CSS si era pensato di realizzare una specie di home per ogni categoria, cosa che è stata semplificata rendendo la pagina simile alla pagina **topviewed** sviluppata in seguito.

Riassumiamo quindi ciò che è stato effettivamente realizzato:

4. Lato programmazione

4.1 Creazione della parte statica del sito

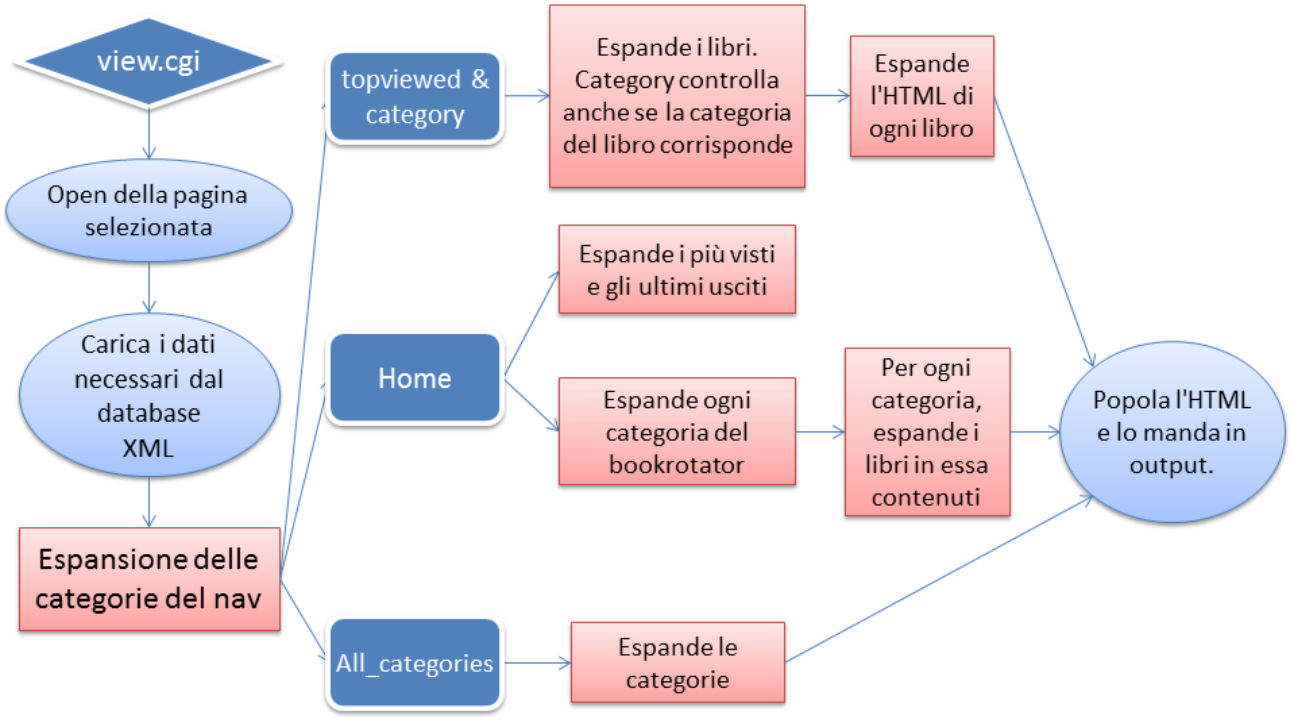


Il sito è diviso in **due parti**, quella **statica** e quella **dinamica**. Lo script createHTML permette di creare la parte statica del sito, esso carica ogni pagina presente nella partella "structure/LINGUA/pages/xml/" e cicla per ogni lingua. Di ogni pagina caricata esso prende i **template** indicati nella pagina come il nav, footer, ecc. e ci applica l'XSLT base e in più ogni XSLT che trova sui template relativi. Il procedimento è illustrato nel grafico all'inizio del paragrafo (l'XML e l'XSLT per semplicità sono rappresentati come un documento solo anche se in realtà sono una serie di documenti nella cartella relativa).

In questo modo la struttura (l'XSLT) è completamente separata dal contenuto (l'XML) tranne che per le pagine *monouso* ovvero quelle presenti in un solo progetto e non in tutte le lingue (almeno teoricamente) dove il contenuto non è fatto con XML+XSLT ma direttamente in HTML per comodità (anche se volendo è possibile trasformare in XML+XSLT anche quelle pagine).

Dopo aver creato ogni pagina per ogni lingua, lo script carica la cartella shared e per ogni pagina contenutavi crea la stessa pagina in tutti i progetti (per esempio si è creata la pagina copyright scritta in inglese ma comune ad ogni lingua.)

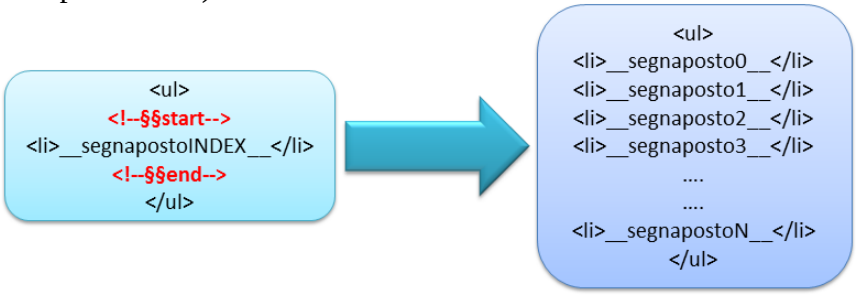
4.2 Creazione della parte dinamica del sito



Così facendo il sito è statico e non può essere aggiornato con i dati di volta in volta modificati. Per fare ciò si sono studiate due soluzioni alternative:

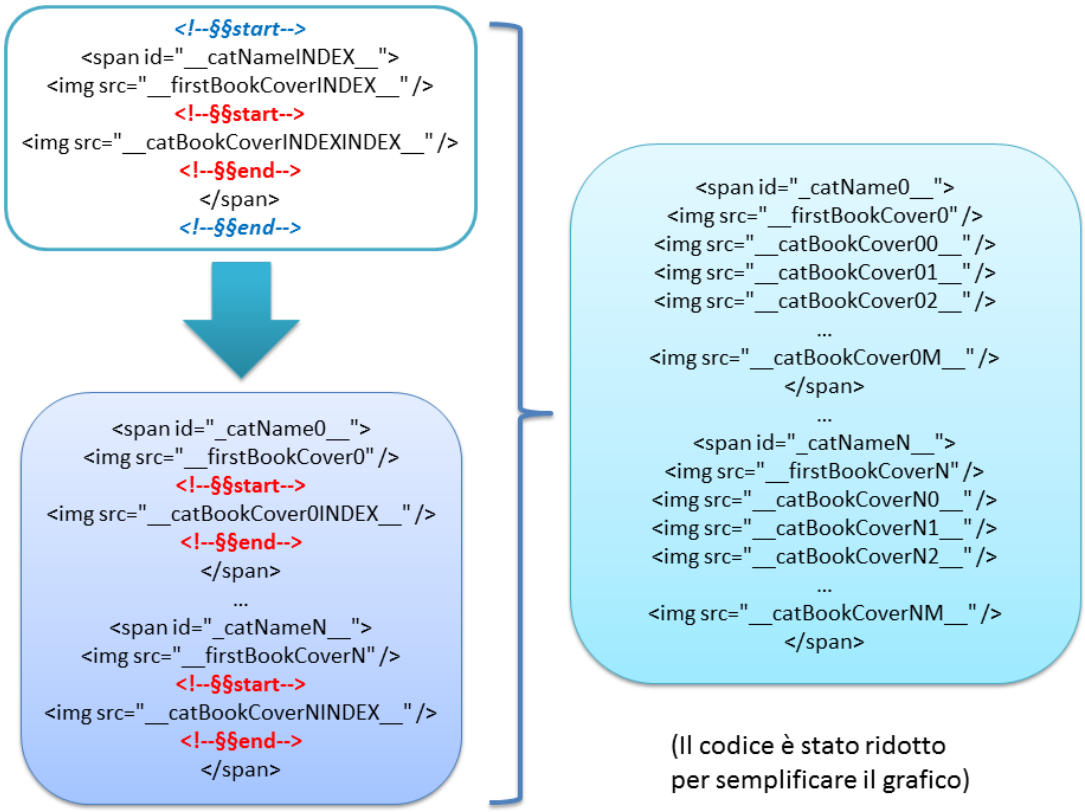
- **Modificare** i file coinvolti **ogni volta** che c'è una modifica nei dati del database. Questo comporterebbe, però, uno sforzo notevole ad ogni visualizzazione di un libro in quanto essa provocherebbe la modifica del contatore delle modifiche, oltre che delle pagine dei topviewed, dell'index e della pagina che visualizza la categoria, ma non è possibile avere tutte queste modifiche per un'azione così banale anche perché la pressione sul server aumenterebbe all'aumentare delle pagine che utilizzano dati dinamici.
- (metodo scelto) Creare uno script **view.cgi** che carica dinamicamente le informazioni dal "database" e modifica le pagine statiche in modo da renderle dinamiche. Per fare ciò sono stati usati dei **segnaposto**, per esempio "__content__" che vengono sostituiti utilizzando delle **espressioni regolari**. In questo modo il codice è completamente indipendente dalla struttura HTML del sito e una modifica dello stesso non comporta alcun problema dal punto di vista del codice.

Il secondo approccio è però complicato da implementare in caso di strutture dinamiche complesse come la pagina topviewed o, peggio, l'home. Per crearle infatti è stato utilizzato un metodo leggermente diverso: si è messa la stringa da *espandere* dentro due **marcatori** `<!--§§start-->` e `<!--§§end-->` con la parola INDEX che viene sostituita di volta in volta con i numeri da 0 al numero massimo utilizzabile (si controlla infatti se nell'hash con i segnaposti da sostituire è presente quello scelto, altrimenti il pezzo HTML non viene espanso oltre).



In questo modo è possibile creare dei pezzi di codice che si duplicano fintanto che è necessario, senza specificare direttamente nel codice Perl la struttura HTML. Così, se si decide di cambiare una lista normale in una lista di definizioni o in una tabella, ciò comporta **solo** una **modifica della parte HTML** e nessuna del codice (se il comportamento della pagina rimane immutato).

Per fare l'homepage si è dovuto **potenziare** questa tecnica in modo che potesse gestire correttamente i segnaposto innestati uno dentro l'altro. Questo è stato possibile con una leggera riscrittura dell'espressione regolare che gestiva i marcatori. Infatti c'era il problema di espandere le categorie e successivamente di espandere per ogni categoria i singoli libri, cosa impossibile da fare utilizzando un indice solo. Nel grafico è possibile vedere come sono stati gestiti i segnaposto innestati con un frammento del codice del bookrotator.

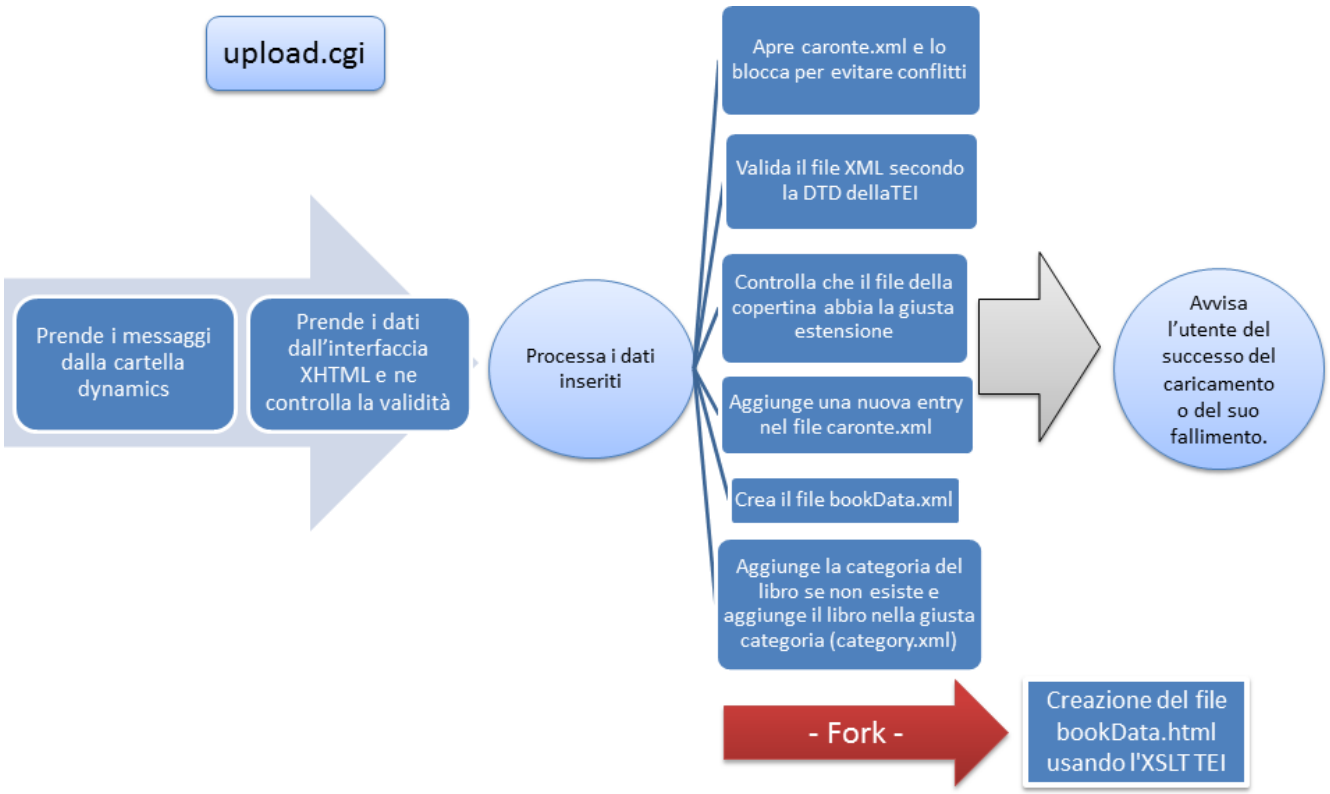


Nota: utilizzare questo sistema di segnaposti potrebbe avere delle ricadute in ambito sicurezza. Studiando attentamente la struttura del sito è infatti possibile fare del code injection per bloccare il caricamento del sito. Teoricamente le espressioni regolari coinvolte sono abbastanza robuste da sopportare l'inserimento del carattere "§" in parti critiche del codice (infatti un punto critico potrebbe essere soltanto tra i tag `<!--§§start-->` e `<!--§§end-->` cosa impossibile da realizzare visto che lo script espande prima il pezzo di codice HTML sostituendo solo alla fine i segnaposto con il loro contenuto e non prima).

La stessa cosa vale se, invece del carattere §, viene inserito un intero marcatore di inizio o fine (o entrambi). Questo non dovrebbe portare a far crashare l'applicazione e neanche alla duplicazione di qualche espansione nel contenuto o in altre parti della pagina, in quanto prima vengono gestiti i segnaposti che si espandono e poi quelli normali. Per questo non si è fatto nessun escaping del carattere § o dei marcatori di inizio e fine.

L'unico comportamento anomalo potrebbe valere per i "segnaposto classici" che semplicemente verrebbero sostituiti con il codice relativo, cosa che può potenzialmente tornare utile e non dovrebbe provocare alcun problema (da un punto di vista di sicurezza).

4.3 Modulo di upload



L'ultimo script Perl presente nel progetto è il modulo dell'**upload**. Esso deve caricare correttamente i dati dei libri effettuando parecchi controlli quali la **validazione** della grammatica, l'esistenza di vari parametri fondamentali per il libro, **senza** che però ciò comporti **inconsistenze** e dati a metà. Inoltre si è pensato di cercare di evitare il più possibile conflitti dovuti alla concorrenza e si è scelto di fare un **fork** alla fine dello script per la parte che pensa alla creazione della versione HTML del libro (perché il modulo un po' lento), non facendo così aspettare inutilmente l'utente. Purtroppo, non avendo a disposizione un DBMS commerciale, tutto ciò è stato gestito manualmente. Per la parte relativa alla cancellazione di procedure incomplete sono stati effettuati test multipli, oltre ad un controllo avanzato del codice, quindi possiamo garantire una **buona affidabilità**. Per la **concorrenza**, non avendo ancora seguito programmazione 3, non ci sentiamo di garantire una serializzazione perfetta delle operazioni (per quanto sia stato il nostro obiettivo).

Nota 1: sempre in ambito sicurezza si è preferito non fare l'escaping dei caratteri inseriti bensì di utilizzare l'opzione CDATA per racchiudere il testo inserito. Ciò, tuttavia, non risolve del tutto i problemi perché la sequenza di caratteri "]]>" potrebbe comunque permettere code injection. Tuttavia, non essendo una problematica veramente complessa da risolvere, molto limite e la cui soluzione non è necessaria viste le specifiche del progetto, non si è trovata una vera soluzione.

Nota 2: il campo short description dell'upload non ha una lunghezza massima. Si suppone che l'utente non inserisca un contenuto superiore alle 300 righe. Avendo HTML5 avremmo usato maxlength="300" nel textarea tuttavia, non potendo, supponiamo un comportamento "logico" da parte dell'utente che ha tutto da guadagnare (in termini di SEO) da una descrizione di ~250 caratteri. Gli esempi, sebbene la descrizione sia breve, superano a volte questo numero in quanto presi da terze parti.

Nota 3: si è presentata tutta la cartella di trasformazione della TEI, anche con materiale *superfluo*, più che altro perché essendo XSLT 2.0 non ne avevamo una piena comprensione e preferivamo non togliere elementi che servivano. L'output, non essendo valido e avendo qualche dettaglio da sistemare, è stato modificato via Perl.

4.4 Bookrotator

Per visualizzare gli ultimi libri caricati nelle varie categorie si è scelto di creare un modulo JavaScript per ciclare i libri delle varie categorie in modo da poter visualizzare nell'area sicura il maggior numero di libri possibile. Se il JavaScript viene disabilitato si visualizza il **primo libro di ogni categoria**. Il modulo, come l'intero sito, presuppone l'esistenza di **almeno un libro** nel database. Infatti, in sua mancanza, sarebbe impossibile l'espansione delle categorie del nav, del contenuto del bookrotator e dei box sottostanti il bookrotator, ma riteniamo che sia un requisito basilare per la creazione stessa del sito.

Nota: nei test di accessibilità si sono riscontrati **problemi** con Internet Explorer di versioni ≤ alla settima. Pertanto si è scelto di disabilitare il bookrotator per queste versioni del sito visualizzandolo come se fosse disabilitato JavaScript.

5. Lato HTML+CSS

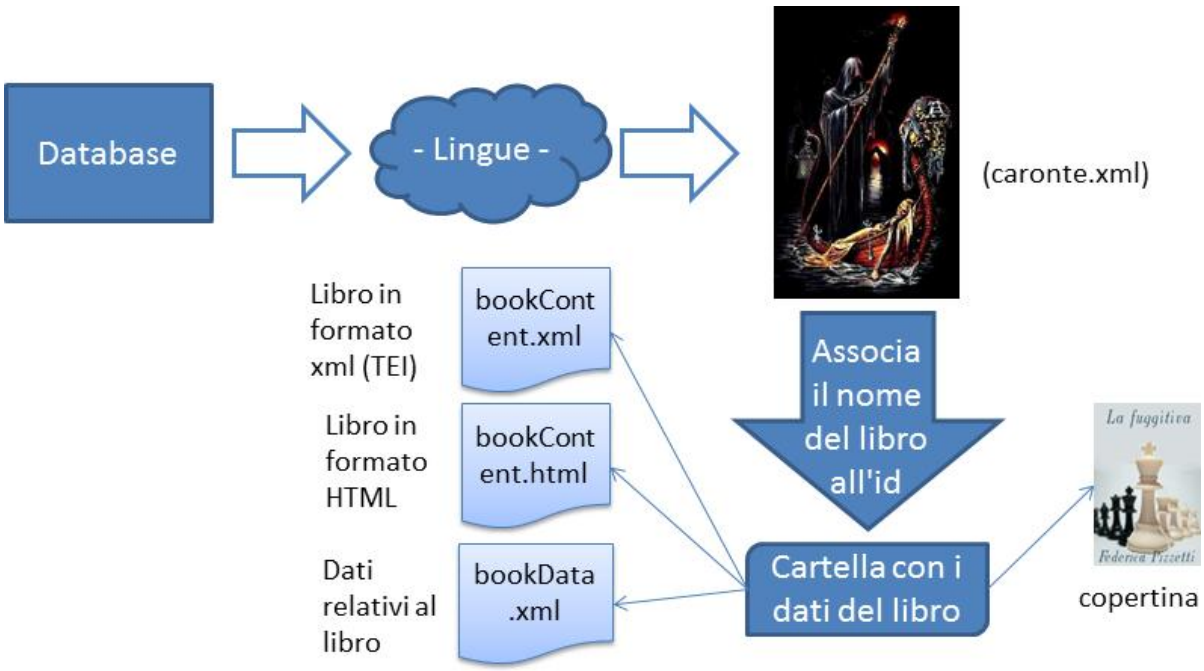
Il sito è stato realizzato seguendo il più rigidamente possibile le richieste fatte in merito a XHTML Strict e CSS puri. E' stato utilizzato un unico selettore CSS 3 per la realizzazione dei bordi arrotondati non essendo una funzionalità critica del sito.

Il codice generato in modo dinamico è leggero ed è valido. Sebbene sia appunto dinamico, il sito non contiene div o span senza senso, per quanto possibile e raggiunge la stessa qualità di un sito creato interamente "a mano". Per riordinare il codice è stato utilizzato il modulo tiny di perl. Teoricamente sarebbe stato possibile lasciare l'output su una sola riga senza spazi, cosa che sarebbe stata più efficiente, ma abbiamo pensato che essendo un sito di valenza prevalentemente dimostrativa/didattica fosse più sensato riordinare il codice in output per facilitarne la correzione. Per lo stesso motivo, sebbene fosse stato detto a lezione che la realizzazione di un unico file css e uno javascript sarebbe stata la scelta migliore per rendere il sito più veloce, si è scelto invece di separare css e javascript in più fogli di stile suddividendoli "semanticamente" per rendere il codice più pulito e comprensibile.

6. Database e Structure

Il progetto consta inoltre di due cartelle fondamentali: database e structure.

6.1 Database



Nel **database** sono presenti i dati dei libri salvati in cartelle "numeriche" con valore crescente. Sono inoltre presenti due file, **caronte.xml** e **category.xml**. Il primo file è stato chiamato così perché viene utilizzato dalle applicazioni per associare il nome di un libro alla corretta cartella, infatti non era possibile salvare direttamente il libro in una directory con il suo nome per le limitazioni imposte dal **filesystem** ai nomi delle directory. Non era neanche possibile realizzare un **megafile** xml contenente tutti i dati, visto che il suo accesso sarebbe stato necessario per ogni minima operazione.

Inoltre ciò avrebbe reso molto critico il file la cui corruzione avrebbe portato ad un blocco irreversibile del sito, mentre in questo modo la corruzione del file sarebbe un problema risolvibile ciclando su ogni cartella e ricostruendolo in tempo lineare mentre la corruzione di un singolo libro ne comporterebbe l'eliminazione senza la perdita di ulteriori dati.

Il file **category.xml** racchiude poi le varie categorie con all'interno ogni libro presente nella categoria. Questo file, per quanto non strettamente necessario (motivo per cui non risulta nel grafico), permette di velocizzare e/o semplificare alcune operazioni sulle categorie delle varie applicazioni che non devono accedere a tutti i file o solo a parte di essi. Gli id dei libri si è preferito separarli da spazi invece che da marcatori ad hoc per evitare un overhead spropositato per singola informazione (per un dato di uno-due caratteri serve un overhead minimo di 6 caratteri, es: `<b>5</b>`).

In ogni cartella sono presenti il file bookData con i dati del libro, la copertina del libro, il file bookContent.xml con il file del libro codificato con la grammatica TEI e il file bookContent.html dov'è codificato in HTML, come mostrato nel grafico.

6.2 Structure

La cartella **structure** (già vista nel grafico dello script createHTML.pl) racchiude tutte le pagine e i template nelle varie lingue codificati interamente utilizzando XML+XSLT. Ogni cartella con la lingua contiene una cartella **dynamics** con i dati dinamici utilizzati dagli script, una cartella **pages** con le pagine in quella lingua e una cartella **templates** con i template utilizzati dal progetto in quella lingua. Le cartelle pages e templates sono quindi suddivise in una cartella **XML** con i dati del **contenuto** relativi a quella lingua e una cartella **XSLT** con la **struttura** degli stessi. La cartella **shared**, strutturata similmente, raccoglie le XSLT condivise tra i vari progetti (in pratica tutte) per cui teoricamente le cartelle XSLT relative ai singoli progetti sono inutili ma sono state aggiunte per completezza (se si volesse infatti fare un'XSLT per il solo progetto italiano sarebbe fattibile senza problemi). La cartella pages di shared contiene come già detto le pagine condivise per ogni progetto.

7. Realizzazione

Come già accennato, la realizzazione del progetto è stata sostanzialmente divisa in due unità non completamente disgiunte tra Filippo e Giorgio. Di seguito si elencano in dettaglio le parti realizzate da ciascun componente del gruppo.

Giorgio

1. Creazione della pagina di index con: layout, nav, footer e content (versioni statiche).
2. Creazione del CSS (completo e funzionante).
3. JavaScript per il cambio lingua senza dover cliccare ok.
4. Rinnovamento del CSS per renderlo fluido fino a piccole dimensioni.
5. Creazione del Logo.
6. Creazione del JavaScript che scorre i libri in home (bookrotator).
7. Creazione del CSS per i cellulari e per la stampa.
8. Sistemazione del CSS per la grammatica TEI.

9. Creazione del javascript "barra fissa" per facilitare la visualizzazione dei libri.
10. Creazione del javascript per togliere il nav per facilitare la lettura dell'utente.
11. Aggiornamento dell'XML+XSLT dell'index già esistente con il nuovo HTML e CSS.
12. Miglioramento del CSS già esistente della pagina di upload e book (con riscrittura e validazione del CSS della TEI)
13. Creazione del JavaScript di controllo per i dati inseriti nella maschera dell'upload.
14. Aggiunta di xml:lang e lang alle pagine: about, copyright, history.

Filippo

1. Leggere modifiche al CSS per sistemare meglio qualche componente nell'index.
2. Script Perl (createHTML) per la trasformazione automatica XML+XSLT → XHTML.
3. Progettazione e creazione della struttura modulare delle pagine del sito (cartella "structure") utilizzando la prima versione del sito come "esempio" per l'XML e l'XSLT.
4. Creazione del file view.pl per la visione dinamica delle pagine (index, category, book, all_categories, topviewed, upload)
5. Formattazione dei libri fittizi all'interno del database.
6. Studiata la TEI Grammar nei suoi aspetti generali.
7. Creazione dello script per la compilazione dei file xml in grammatica TEI.
8. Creazione del contenuto delle pagine 404, about, all_categories, book, category, copyright, error, history, topviewed, upload e upload_results (in xml+xslt e in html tramite script).
9. Creazione dello script perl per l'upload da file e da textarea.
10. Progettazione e creazione del database XML e relativa gestione tramite perl+xpath.
11. Creazione dello script per creare il codice HTML da una pagina strutturata con la grammatica TEI.
12. Sistemata l'XSLT TEI (inizialmente non funzionava e produceva HTML non valido).
13. Stesura della relazione & coordinamento del gruppo.
14. Caricamento della versione in inglese del sito con adattamenti della versione tradotta da Fabrizia Pizzuti, una nostra amica laureanda in lingue.
15. Valutazione dell'accessibilità del sito e conseguenti modifiche (aggiunta di tabindex, accesskey, controllo per il daltonismo e test con i vari browser).

7.1 Difficoltà nella realizzazione

Le difficoltà principali incontrate nella realizzazione del progetto sono dovute all'utilizzazione del linguaggio perl per la costruzione di strutture dati che necessitavano di **hash** e **array innestati,** la complessità nella **programmazione ad oggetti** (che infatti non è stata utilizzata) oltre all'utilizzazione di **file XML** per simulare un database relazionale.

Riguardo al primo problema, non si possono specificare le variabili in input delle funzioni, pertanto si è scelto di passare alle funzioni degli hash anonimi per poter ampliare gli script senza rischiare di dover riscrivere grandi porzioni di codice. Tuttavia, non si possono passare degli hash in questo modo alle funzioni, l'unico modo che si può utilizzare è passare un puntatore. Però, avendo il puntatore, non si può accedere all'array con le chiavi con il comando apposito. Pertanto, nasce la necessità di passare anche l'array tramite puntatore. Peccato che dai test effettuati ciò possa dare dei problemi: per esempio, invece che avere l'intero array spesso passava solo un array di un elemento (senza motivo apparente). In sostanza, il problema è stato risolto facendo un join di tutte le chiavi mettendo "####" come delimitatore e facendo il conseguente split nella funzione separata. Purtroppo non abbiamo trovato una soluzione migliore.

Il terzo problema è già stato sviscerato. Se avessimo potuto decidere come implementare il sito in ogni dettaglio, sarebbe stato più comodo ed efficiente utilizzare un DBMS commerciale per la parte dinamica, mentre avremo continuato ad utilizzare xml+xslt per le pagine che componevano la parte statica del sito e

che permettevano la creazione delle pagine nelle varie lingue oltre alla configurazioni di view.cgi e upload.cgi. Tale soluzione è stata implementata per il progetto di Basi di Dati.

8. Coordinamento

Per il coordinamento del gruppo è stato utilizzato il sito **www.origo.ethz.ch** che permette di creare dei siti internet con forum, wiki, sistema di ticketing e repository svn con sorgente privato. Lo scambio di documenti, le spiegazioni, la progettazione sono avvenuti utilizzando il forum dedicato. Sono poi stati utilizzati i **ticket** per permettere a Filippo di assegnare i lavori e le scadenze ad ogni membro del gruppo. Infine per lo sviluppo del codice condiviso è stato utilizzato il sistema svn (subversion) messo a disposizione dal sito stesso. Per i grafici si è utilizzato **Powerpoint**.

9. Testing

Il sito non è molto ampio e ha solo 3-4 script anche se complessi. Ogni script è stato testato a fondo, il modulo per l'upload è stato testato inserendo 8 libri diversi, il bookrotator è stato tra i primi ad essere realizzato ed è stato testato mentre lavoravamo sulle altre componenti. Lo script topviewed è stato testato per circa una settimana con vari libri, varie categorie (aggiungendone di fittizie) e provando anche a variare sia il numero di categorie che di libri. Lo script per le categorie utilizza (circa) le stesse funzionalità.

Lo script per la visualizzazione del singolo libro è stato testato con solo qualche libro di esempio vista la difficoltà nel reperirne online (e la mancanza di tempo materiale per scriverne alcuni di prova *significativi*). Il testo dei libri è sempre lo stesso (quello della guida alla grammatica TEI).

10. Browser

E’ stato effettuato il test con diversi browser e sistemi operativi.

Internet Explorer 5.5 6 7 8 9. Lo si è provato con i browser “veri” per le versioni IE6 e IE8 e tramite il programma IETester per le altre. Funziona perfettamente con le versioni superiori alla settima, con IE 5.5 carica l’home abbastanza male anche se è navigabile, le pagine con solo testo le carica correttamente mentre il topviewed (e le categorie) le carica non perfettamente ma quasi. IE 6 lo stesso anche se carica la home page in modo migliore ma comunque sfasata. IE 7 dà solo un problema quasi impercettibile coi bordi. Tutte e tre le versioni non riescono a far funzionare correttamente il bookrotator (ma dalla versione 8 funziona correttamente), pertanto è stato disabilitato solo per queste versioni (test effettuato su Windows 7).

Firefox 3.6 4 5, nessun problema (sia Windows 7 che Ubuntu 10.10).

Chrome 12, nessun problema (sia Windows 7 che Ubuntu 10.10).

Opera 11.50, nessun problema (sia Windows 7 che Ubuntu 10.10).

Safari 5.0.5, nessun problema (solo Windows 7).

Lynx 2.8.7 il sito è navigabile tranquillamente anche grazie al link che permette di saltare il nav. Essendoci l’alt su ogni copertina l’utente può comunque fruire del servizio. (Ubuntu 10.10).

Il sito ha anche un CSS per la stampa che è stato testato ed è funzionante e un CSS per i cellulari, testato con HTC Desire HD (**Android**) usando il browser **Dolphin HD** e un **Iphone 4**. Su Android rimane la

visualizzazione minima del desktop (ma è fruibile come se fosse “dedicata”), forse a causa delle dimensioni maggiori dello schermo rispetto all’Iphone. Con l’iphone invece scatta la visualizzazione da cellulare correttamente.

11. Accessibilità

Riguardo l'accessibilità sono stati seguiti i punti visti a lezione e all'esercitazione relativa, in particolare:

Test dei colori

Abbiamo evitato per quanto possibile colori non accessibili per persone affette di daltonismo. I test effettuati con il sito http://www.vischeck.com/ non hanno indicato problemi al riguardo. In seguito si mostrano gli screenshot delle pagine con i diversi gradi di daltonismo. In merito ai singoli libri potrebbero esserci dei leggeri problemi a causa delle copertine, inevitabile se l’upload di queste immagini è lasciato agli utenti, ma il sito di per se risulta dalle immagini accessibile a persone con vari gradi di daltonismo.

Original Image

Deuteranope Simulation

Protanope Simulation

Tritanope Simulation

Criteri Standard

- Utilizzo di tecnologie e raccomandazioni W3C.
- **Separazione** completa tra struttura e contenuto.
- Utilizzo di un attributo **alt** significativo per le immagini.
- Utilizzo di **tabindex** e **accesskey**. Sono stati utilizzati entrambi nel nav, il tabindex è messo correttamente nel modo usuale tramite Perl, l’accesskey invece ha delle lettere significative per le voci del nav “statiche”, per quelle create dinamicamente ha un numero (dopotutto può variare a seconda dei libri inseriti dagli utenti), anche se non “il massimo” questa ci è sembrata la soluzione migliore. Il footer ha solo l’accesskey altrimenti usando il tabindex, finito il nav salterebbe al footer (a meno di cose inutilmente complicate usando il Perl).
- **Tag meta** corretti.

- Traduzione per le parole inglesi con gli attributi **xml:lang e lang.**
- Poter sempre rispondere alle domande: "**dove sei?**" e "**dove puoi andare?**". Per il "dove sei" basta fare riferimento a quello che nell'xml è il "titleContent" oltre al fatto che ogni link se cliccato "perde" il link. Per il dove puoi andare il nav è sempre visibile e nel footer abbiamo "relegato" le pagine di minore importanza.
- **Title** corretto e diverso per ogni pagina. Questo è stato possibile grazie alla soluzione xml+xslt.
- Non ci siano errori di battitura.

Trasformazione

- Trasformazione elegante del sito. Ogni JavaScript se viene disabilitato ha una soluzione alternativa. Per la lingua compare il tasto «invia» (comporta un click in più ma è perfettamente utilizzabile), per il bookrotator, non potendo visualizzare tutti i libri, si visualizzano solo un libro per categoria, mentre la barra fissa per il libro compare solo se c'è il JavaScript e senza di essa il libro è consultabile senza problemi.

Contenuto

- Contenuto semplice, comprensibile e navigabile.
- **Abbreviazioni e acronimi** gestiti con i relativi tag XHTML. Fatto, in particolare nelle pagine descrittive. Nelle descrizioni dei libri, non essendo inserite da noi, non possiamo averne un diretto controllo.

12. Librerie esterne e processore XSLT 2.0

Per il progetto sono state necessarie le seguenti librerie la cui installazione è stata richiesta (ed evasa) dai tecnici.

- libxml-parser-perl
- libxml-libxslt-perl (già presente sul server)
- libtie-ixhash-perl (già presente sul server)
- libhtml-tidy-perl
- libsaxon-java

In particolare la libreria saxon è stata utilizzata come **preprocessore XSLT 2.0** per trasformare i documenti in grammatica TEI in documenti HTML con l'XSLT presente nel sito relativo. Purtroppo il parser presente per il linguaggio perl supporta solo fino all'XSLT 1.0 e non oltre.

Sempre con l'aiuto dei tecnici sono stati creati dei **soft-link** nella cartella public_html per permettere il corretto funzionamento del sito sia su configurazioni dove la **document-root** è "/var/www" (o analogo), configurazione che avevamo noi in locale, sia per una configurazione dove la document-root è "/var/www/public_html" (o analogo), configurazione del server di tecnologie web.

Infine, si era valutato l'utilizzo di **redirect 301** utilizzando il modulo mod_rewrite di apache. Purtroppo il tecnico ci ha detto che il modulo non è abilitato per gli studenti per motivi di sicurezza. Similmente sarebbe stato possibile rimappare gli url generati da view.cgi con url più efficienti e user-friendly (e migliori in ambito SEO). Comunque i redirect sono stati gestiti con semplici file HTML (anche se è una soluzione meno efficiente) e i link di view.cgi sono rimasti così.

13. Ringraziamenti

Si ringraziano:

- Anna Letizia Zocche, per aver trovato dei sample di libri fittizi (le immagini e le descrizioni scritte in formato .odt che sono state poi formattate da noi in dati xml).
- Fabrizia Pizzuti, per la traduzione in inglese di alcune pagine del sito (oltre ad una revisione di altre scritte da noi).