UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

TESI DI LAUREA TRIENNALE

Collaborazione al progetto *Edition Visualization Technology*: sviluppo di un software di visualizzazione di edizioni digitali

Relatore:
Prof. Roberto Rosselli Del Turco

Candidato:
Julia Kenny

Correlatore:
Prof. Andrea Marchetti

Anno Accademico 2011-2012

Indice

Introduzione

Questa tesi è stata scritta al fine di spiegare come si è svolta la mia collaborazione con il progetto EVT - *Edition Visualization Technology*.

Il progetto ha come obiettivo la realizzazione di un software per la visualizzazione dell'edizione digitale del *Codex Vercellensis* ed è parte del progetto VBD - *Vercelli Book Digitale*.

Il rapporto di collaborazione con questo progetto inizia al termine della mia attività di tirocinio, durante la quale ho avuto la possibilità di applicare le conoscenze apprese nel corso dei miei studi alla realizzazione di siti web commerciali. Al termine di questa attività ho deciso di partecipare al progetto *Edition Visualization Technology* per avere la possibilità di continuare ad approfondire e arricchire le mie conoscenze, questa volta tuttavia, lavorando all'interno di un contesto culturale.

Nel primo capitolo traccio una panoramica sulla codifica digitale dei testi e l'importanza delle edizioni digitali; poi introduco brevemente il progetto *Vercelli Book Digitale* parlando del manoscritto, dei motivi che hanno spinto alla realizzazione di una sua edizione digitale e delle componenti necessarie alla sua realizzazione; infine introduco il progetto *Edition Visualization Technology* ponendo l'accento sulle esigenze del visualizzatore, sulle soluzioni adottate per soddisfarle attraverso la realizzazione del software *EVT Builder* e sui risultati raggiunti al momento dell'inizio della mia collaborazione.

Nel secondo capitolo approfondisco il funzionamento del software illustrando le tecnologie utilizzate per il suo sviluppo e analizzando il suo funzionamento. Successivamente spiego perché, in seguito allo studio del software, ho deciso di modificarlo, in che modo sono intervenuta sul sistema per migliorarlo e i vantaggi apportati.

Nel terzo capitolo illustro come ho curato la realizzazione dell'edizione facsimile e dell'edizione diplomatica del testo e come ho sviluppato gli strumenti *collegamento testo-immagine* e *lente di ingrandimento* per arricchire il visualizzatore. Per ciascuno di questi strumenti vengono fornite informazioni sul funzionamento e su cosa è stato fatto per inserirli nel sistema.

Nel quarto capitolo esprimo delle osservazioni di carattere generale maturate durante e dopo la realizzazione del progetto.

UNO STRUMENTO PER LA FILOLOGIA DIGITALE: EVT

1.1 La filologia digitale

Un testo codificato rappresenta un modello concettuale del testo originale. La codifica di un testo richiede da parte del curatore un'accurata selezione degli elementi da impiegare; durante la marcatura di un testo, il filologo mette in risalto le caratteristiche principali del testo e la sua struttura.
Se viene eseguita una marcatura approfondita, un testo codificato può contenere molte informazioni; questo dipende anche dalle scelte del filologo che, durante la codifica, può limitare o no gli interventi editoriali: ad esempio, un testo può essere ricco di errori, abbreviazioni e altri fenomeni che un filologo può riportare così come sono oppure, per ciascuno di essi, può fornire la forma secondo lui corretta o l'espansione dell'abbreviazione o, ancora, può indicare entrambe le forme. Un'edizione diplomatica più o meno "rigida" e un'edizione critica richiedono un diverso tipo di interventi editoriali.
Le informazioni contenute in un testo codificato non devono essere necessariamente visualizzate tutte contemporaneamente nell'edizione finale del testo: stabilendo delle convenzioni, è possibile creare più livelli di edizione del testo contenenti ciascuno una selezione delle informazioni descritte nel testo codificato e corrispondenti ai diversi tipi di edizione che un filologo può preparare.
Diversi livelli di edizione di un testo rispecchiano diversi tipi di interessi di ricerca; ad esempio, per uno studio linguistico del testo può essere preferibile una versione molto vicina all'originale mentre per uno studio letterario del testo può essere preferita una versione a cui è stato aggiunto un livello maggiore di interpretazione da parte del curatore.

Uno dei vantaggi garantiti dall'applicazione di tecnologie informatiche alla filologia è il fatto che non è più necessario creare tante versioni del testo quante sono i tipi di edizione desiderati: un documento codificato in formato digitale può contenere contemporaneamente tutte le informazioni riguardanti i vari livelli di edizione ed

essere sottoposto a più trasformazioni, ciascuna volta alla produzione di un tipo diverso di edizione del testo che riporti solo una selezione delle informazioni contenute nel file iniziale.

La dimensione dinamica e interattiva che solo lo strumento informatico può offrire costituisce il valore aggiunto delle edizioni digitali rispetto a quelle cartacee, soprattutto nel caso di edizioni digitali di testi antichi: la rappresentazione del testo tramite costrutti ipertestuali rende possibile mostrarne dinamicamente la storia della tradizione testuale e permettere collegamenti a testi aggiuntivi ai quali il testo principale fa riferimento o ad altre informazioni utili per la comprensione del testo da parte del lettore moderno.
Un'edizione digitale di un manoscritto permette inoltre di mostrare contemporaneamente (e confrontare) più livelli di edizione del testo e, grazie ai moderni strumenti per l'acquisizione delle immagini, le scansioni ad alta risoluzione dei singoli fogli del manoscritto.

I dati contenuti nei testi trascritti e nelle immagini digitali possono essere processati, manipolati e interrogati dal computer. Attraverso l'integrazione di un motore di ricerca testuale si possono compiere ricerche complesse sul testo. È inoltre possibile offrire strumenti avanzati per l'elaborazione delle immagini (offrire la possibilità di compiere ingrandimenti per osservarne i dettagli, utilizzare righelli per calcolarne le dimensioni reali o filtri per l'elaborazione grafica) e strumenti utili per l'analisi dei manoscritti in senso fisico: la composizione delle pagine, lo studio dei caratteri e delle iniziali miniate, della grafia, degli errori e di tutti quei dettagli che rendono ogni artefatto unico. La produzione di edizioni digitali si rivela molto utile nei casi in cui l'interesse verte non sul testo inteso come entità astratta, ma sui singoli esemplari di un testo. La cosiddetta *new philology*, ad esempio, pone l'interesse sulle singole realizzazioni dei testi, sul singolo manoscritto, ne analizza le differenze rispetto agli altri esemplari, le particolarità del testo che trasmette: ciò che è diverso non è necessariamente sbagliato, può dipendere dalla ricezione del testo da parte dello scriba che lo ha trascritto o da una sua volontà di modernizzarlo oppure di renderlo comprensibile a destinatari meno colti.[1]

[1] Carlquist J., "Medieval Manuscripts, Hypertext and Reading. Visions of Digital Editions", in *Literary and Linguistic Computing*, 19, p. 112.

In quanto risultato di un lungo e difficile lavoro manuale, i manoscritti che sono stati conservati nel corso dei secoli e che sono giunti a noi costituiscono un tesoro prezioso e la consultazione diretta di tali artefatti mette a rischio la loro conservazione: creare un'edizione digitale è una buona soluzione per preservare i testi originali, oltre che per renderli più accessibili agli studiosi e a tutti gli interessati che, grazie alle capacità di diffusione del web, non sono più soggetti a limitazioni fisiche o temporali per la consultazione del testo.

1.2 Il progetto VBD - Vercelli Book Digitale

Il progetto *Vercelli Book Digitale* (VBD) nasce nel 2003 su iniziativa di Roberto Rosselli Del Turco[2] con l'obiettivo di produrre una versione digitale del *Codex Vercellensis*. Il codice, conservato a Vercelli presso la Biblioteca Capitolare[3], è oggetto di studio da parte di ricercatori di tutto il mondo e una versione digitale di tale codice può offrire agli studiosi una valida alternativa alla sua consultazione diretta.

1.2.1 Il codice

Il *Codex Vercellensis*, noto anche come *Vercelli Book*, è un manoscritto redatto nell'Inghilterra meridionale verso la fine del X secolo.
Il manoscritto è costituito da 136 fogli di sottile pergamena molto ben conservati; contiene una miscellanea di opere a carattere religioso in versi e in prosa[4] ed è uno dei quattro manoscritti[5] risalenti alla fine del X secolo che contengono circa il 90% di tutta la produzione poetica anglosassone.
I testi contenuti in questo codice sono essenziali per la comprensione della cultura e della lingua dell'Inghilterra anglosassone di quel periodo: le omelie in esso contenute sono di grande importanza per lo studio della prosa anglosassone e, in particolare,

[2] Ricercatore di Filologia Germanica presso l'Università di Torino.
[3] Biblioteca e archivio Capitolare di Vercelli: http://www.tesorodelduomovc.it/archivio/index.html.
[4] 23 omelie in prosa e 6 componimenti poetici.
[5] Gli altri sono l'*Exeter Book*, il *Cotton Vitellius A XV* e lo *Junius 11*.

della storia religiosa; alcuni dei componimenti in esso contenuti spiccano per qualità artistica e, inoltre, ben undici delle ventitré omelie e una parte dei componimenti poetici rappresentano l'unica copia esistente di tali testi.

1.2.2 Il progetto

Il progetto *Vercelli Book digitale* nasce sulla spinta del successo e dei continui sviluppi che ha riscontrato l'applicazione di tecnologie informatiche alle discipline umanistiche; sfruttando le potenzialità di un'edizione digitale di testi medievali, si propone di fornire allo studioso uno strumento completo e facilmente utilizzabile per compiere analisi e ricerche sul *Vercelli Book*.

Un'edizione digitale è composta, come minimo, dalla trascrizione del testo e da un software di visualizzazione, cui possono aggiungersi immagini e strumenti avanzati di vario tipo. Molti sono gli aspetti da curare per creare una valida edizione digitale e, per raggiungere tutti gli obiettivi che sono stati stabiliti per il progetto, esso è stato suddiviso in più moduli col fine di creare una struttura tale da permettere di eseguire in parallelo moduli diversi.
Al momento dell'inizio della mia collaborazione era stata effettuata una trascrizione completa del manoscritto (utilizzando una marcatura approfondita del testo che permette di generare due livelli di edizione dei testi e di compiere ricerche testuali di tipo complesso) ed era stata eseguita una digitalizzazione del manoscritto; era in corso la raccolta e l'organizzazione della bibliografia sul *Vercelli Book* e lo sviluppo del software per la visualizzazione dell'edizione; mentre erano state rimandate ad una fase successiva del progetto, invece, l'implementazione di un motore di ricerca testuale e la scelta, da compiere caso per caso, di includere materiale aggiuntivo nel VBD e di realizzare edizioni critiche dei testi più interessanti.

Vista la scarsa quantità di software attualmente esistenti per la navigazione dei manoscritti, lo sviluppo di questo tipo di programma per il *Vercelli Book* ha dato il via a un progetto separato: *EVT - Edition Visualization Technology*.

1.3 Il progetto EVT - Edition Visualization Technology

Il software EVT - *Edition Visualization Technology* - è un software di visualizzazione e consultazione per l'edizione digitale del *Vercelli Book*, realizzato nell'ambito del progetto *Vercelli Book Digitale* per rendere il manoscritto e i suoi contenuti facilmente accessibili a tutti gli studiosi e gli appassionati.
Per raggiungere questo obiettivo è essenziale che lo strumento sia completo, flessibile e facilmente utilizzabile anche da utenti con competenze informatiche di medio o basso livello.

Obiettivo di EVT è quello di offrire testi scientifici di alta qualità sul Web, sfruttando le potenzialità di diffusione di Internet e tutti i vantaggi che il mezzo digitale offre (come il supporto di strumenti avanzati di consultazione e analisi testuale o strumenti per l'elaborazione delle immagini).

1.3.1 EVT Builder

Nel corso degli ultimi cinque anni, il software per la visualizzazione del manoscritto ha subito diverse trasformazioni e al momento dell'inizio della mia collaborazione era appena stato riprogettato, creando EVT Builder.

EVT Builder nasce in seguito ad un'analisi delle precedenti versioni, per migliorare la gestione dei dati del manoscritto, ma conserva alcune delle scelte che erano state designate in fase di progettazione delle versioni precedenti, come quella di utilizzare formati e linguaggi non proprietari (sia per tutelare la durabilità del prodotto, sia per le possibilità di ridistribuzione del software stesso).

EVT Builder si basa sulla trasformazione di un file XML per mezzo di una serie di fogli di stile XSLT. Il prodotto delle trasformazioni è un'applicazione web concepita in modo tale da poter girare sia su un server web sia in locale. Il file principale di questa applicazione è la pagina *index.html*; questa pagina carica dinamicamente il testo delle pagine del manoscritto e le relative immagini e include tutti i link

necessari alla gestione dell'applicazione. Una panoramica sui linguaggi utilizzati e sul funzionamento di EVT builder sarà fornita nel capitolo 2.

EVT builder pone le basi per creare uno strumento che permette di pubblicare l'edizione digitale di un testo e soddisfa una parte delle funzionalità richieste al visualizzatore del *Vercelli Book*: con questo software viene data all'utente finale la possibilità di accedere, tramite la pagina principale, a tutti i contenuti (testo e immagini), di consultare il manoscritto in modalità testo-testo o in modalità testo-immagine e viene fornito un primo, utile, strumento per le immagini, lo zoom.
Inoltre, la trasformazione è impostata per dividere il testo del manoscritto in pagine HTML singole (ciascuna corrispondente a un lato, recto o verso, dei fogli che compongono il manoscritto) e offrire la possibilità di creare una o più edizioni del testo.
L'interfaccia grafica è provvisoria[6], ma è stata realizzata tendendo conto di alcuni dei punti che saranno la base per la futura interfaccia: la struttura è stata progettata in modo da mantenere un aspetto minimale e da massimizzare la visualizzazione dei contenuti. È stata inoltre realizzata in modo tale da avere un comportamento tendenzialmente adattivo: l'area occupata dai riquadri di testo e immagini si adatta sempre al massimo spazio disponibile e sono state assegnate ai vari riquadri delle dimensioni minime limite al fine di non rendere mai inutilizzabile l'interfaccia. Il gestore dell'interfaccia è stato progettato per tenere conto delle scelte dell'utente, consentendogli di navigare le pagine a disposizione mantenendo le impostazioni selezionate in precedenza.

1.3.2 La mia collaborazione

Durante la mia collaborazione mi sono impegnata per migliorare il sistema e sviluppare nuovi strumenti, il tutto con l'obbiettivo di raggiungere una versione del visualizzatore sufficientemente completa per essere mostrata a dei possibili utenti e ricevere un primo feedback[7] sul lavoro svolto da me e dagli altri sviluppatori.

[6] Un'altra studentessa si sta occupando dello sviluppo dell'interfaccia grafica.
[7] Le opinioni degli utenti sono un elemento fondamentale per migliorare il prodotto.

Si noti che questa versione preliminare si limita alla presentazione in formato digitale del poemetto *Il Sogno della Croce*[8].

Seguendo le linee guida stabilite dai filologi che si sono occupati della codifica del *Vercelli Book*, ho curato le due edizioni di testo (edizione facsimile ed edizione diplomatica) permesse dall'attuale codifica; ho gestito le trasformazioni necessarie alla creazione di tali edizioni e ho sviluppando il layout richiesto per la visualizzazione delle edizioni[9].
Ho poi lavorato alla realizzazione dello strumento di *collegamento testo-immagine*, strumento che permette di collegare ogni riga della scansione di un foglio con le rispettive righe nell'edizione del testo[10].
Infine ho realizzato lo strumento *lente di ingrandimento*, così da andare a completare il set di strumenti base per l'analisi dell'immagine (zoom e lente di ingrandimento)[11].

Per il corretto svolgimento di questi compiti è stato necessario approfondire il funzionamento di EVT builder e approfondire la conoscenza delle tecnologie utilizzate per il suo sviluppo (e di conseguenza necessarie per la sua implementazione). Durante lo studio di EVT builder si è inoltre rivelata essenziale la modifica di parte del sistema di trasformazione del file XML[12].

[8] *Vercelli Book*, ff. 104v-106r.
[9] Si veda il capitolo 3, paragrafo 3.1 per maggiori dettagli.
[10] Si veda il capitolo 3, paragrafo 3.2 per maggiori dettagli.
[11] Si veda il capitolo 3, paragrafo 3.3 per maggiori dettagli.
[12] Si veda il capitolo 2 per maggiori dettagli.

EVT BUILDER NEL DETTAGLIO

2.1 Tecnologie utilizzate per lo sviluppo

Di seguito è fornita una breve descrizione delle tecnologie utilizzate per lo sviluppo di EVT Builder.
La conoscenza di queste tecnologie è essenziale per il lettore che vuole comprendere il sistema mentre il loro approfondimento è indispensabile per chi, come me, deve implementare determinate funzionalità all'interno dello stesso.

2.1.1 XML

XML nasce per semplificare l'elevata complessità del suo predecessore, SGML, e ha l'obiettivo di permettere la descrizione, l'archiviazione e il trasporto dei dati. È un linguaggio standard[13] del W3C, il consorzio per la definizione degli standard del web.

Come suggerisce l'acronimo - *eXtensible Markup Language* - è un linguaggio estensibile di marcatura, un insieme di regole sintattiche standard per la descrizione della struttura di documenti e di dati.

Un documento XML è un file di testo in cui ciascun componente logico è rappresentato da un elemento. I dati sono rappresentati in modo gerarchico: è obbligatorio indicare un elemento principale, chiamato elemento radice, che contiene tutti gli altri elementi; ogni elemento può possedere degli attributi che ne descrivono le proprietà e può contenere altri elementi oppure del testo oppure entrambi. Possiamo rappresentare graficamente la struttura di un documento XML tramite un albero (*documenti tree*) i cui nodi sono gli elementi, gli attributi e il testo del documento.

[13] Le specifiche ufficiali sono consultabili all'indirizzo: http://www.w3.org/TR/2008/REC-xml-20081126/.

Un documento XML si definisce *ben formato* se rispetta le regole sintattiche base di XML (cioè se contiene almeno un elemento, se possiede un elemento radice che contiene tutti gli altri elementi e se tutti gli elementi sono correttamente nidificati).
Nonostante la sintassi semplice, XML riesce a rappresentare strutture dati molto complesse.

XML non impone alcun vincolo riguardante la natura degli elementi; questo lo rende molto flessibile e utilizzabile in una grande quantità di situazioni ma, per rispecchiare una determinata struttura logica, diventa necessario creare una grammatica che imponga dei vincoli, stabilendo quali elementi usare e come usarli. La grammatica viene solitamente definita tramite una DTD (Document Type Definition) o tramite il linguaggio XML Schema.
Un documento XML viene detto *valido* per uno specifico schema se rispetta la grammatica di tale schema.

Perché un documento XML associato a uno schema di codifica possa essere processato è necessario che esso sia valido e ben formato. I software che si occupano di effettuare questi controlli sono detti parser.
XML è stato concepito per trasportare i dati, non per visualizzarli direttamente: per ottenere in output la formattazione desiderata è necessario applicare a un documento XML dei fogli di stile.

2.1.2 XSL

XSL eXtensible Stylesheet Language è stato sviluppato per rispondere alla necessità di avere fogli di stile sviluppati appositamente per documenti di tipo XML.
Il fatto che XML non usi elementi predefiniti implica che il significato degli elementi utilizzati non sia universale e che un browser non sappia come visualizzarli.
XSL descrive il modo in cui i documenti XML devono essere visualizzati.

L'XSL incorpora tre linguaggi: XSLT, XPath e XSL-FO. Al fine di comprendere il presente lavoro è necessario conoscere XSLT e XPath.

2.1.2.1 XSLT

XSLT (*XSL Trasformations*) è un linguaggio per la trasformazione della struttura di documenti XML. Il risultato di tale trasformazione può essere sia un documento XML, sia un documento di altro tipo (ad esempio una pagina HTML).
La trasformazione viene eseguita da un parser XSLT il quale prende in input l'albero del documento XML e, applicando le regole contenute nel fogli di stile XSLT, produce come risultato un nuovo albero adatto al tipo di documento desiderato.
XSLT è un linguaggio dichiarativo: ogni regola descrive il risultato finale da ottenere quando in ingresso si incontra un determinato modello.

Un foglio di stile XSLT è composto principalmente da un insieme di regole chiamate *template* e scritte utilizzando la sintassi XML. I template descrivono come deve essere trattato l'elemento o il costrutto cui fanno riferimento. Grazie ai template e agli elementi che il linguaggio offre, è possibile compiere numerose operazioni come modificare gli elementi, ordinarli, aggiungere elementi e attributi al file finale o non riportare alcuni di essi presenti nel file originale o, ancora, effettuare scelte in base al risultato di test.

2.1.2.2 XPath

XPath (XML *Path*) serve per navigare e interrogare l'albero di un documento XML.
A differenza di XSLT, XPath è basato su espressioni che utilizzano una sintassi propria e non quella di XML. XPath fornisce una libreria di funzioni che possono essere utili all'interno delle sue espressioni; esistono, infatti, funzioni per svolgere svariate operazioni (manipolare stringhe o numeri, accedere a proprietà dei nodi o accedere a informazioni sulla posizione dei nodi).
Ogni espressione XPath viene valutata e produce un oggetto che può essere un set di nodi, un valore booleano, un numero o una stringa. Attraverso le espressioni è possibile spostarsi da un punto all'altro dell'albero del documento, selezionarne una porzione o effettuare dei test in vista di istruzioni condizionali.

XPath è di primaria importanza all'interno di un foglio di stile XSLT: è attraverso di esso che un template può selezionare i nodi cui far corrispondere una regola.

1.2.3 HTML

L'HTML (*HyperText Markup Language*) è il linguaggio di marcatura per la descrizione delle pagine web; attraverso una serie di tag predefiniti, indica gli elementi che costituiscono una pagina.
Anche il file HTML, come quello XML, ha una struttura ad albero annidato. È tuttavia necessario selezionare una delle DTD predefinite[14] per segnalare la versione di HTML di riferimento e, di conseguenza, le relative specifiche (che indicano quali elementi, attributi ed entità si possono utilizzare).

Uno dei vantaggi più rilevanti offerti da HTML è la possibilità di creare, tramite l'utilizzo di àncore che collegano una pagina HTML a un'altra, una struttura ipertestuale.

HTML rappresenta la struttura logica di una pagina, non la sua formattazione. Il set di elementi utilizzati dal linguaggio HTML è un set standard e ogni browser associa a questi elementi una certa formattazione di default, formattazione che l'utente può anche modificare attraverso le preferenze di tale browser. Per fare in modo che il layout di una pagina HTML sia personalizzabile e che sia visualizzato allo stesso modo da tutti i browser si fa ricorso all'utilizzo dei fogli di stile CSS.

1.2.4 CSS

I fogli di stile CSS - *Cascading Style Sheets* - definiscono come devono essere visualizzati gli elementi che costituiscono una pagina HTML.
Le regole CSS hanno una sintassi propria e sono formate da due componenti: un selettore per indicare l'elemento del quale vogliamo definire la formattazione e una

[14] Consultabili sul sito del W3C: http://www.w3.org/QA/2002/04/valid-dtd-list.html.

dichiarazione per indicare quale valore attribuire a ogni proprietà di quell'elemento. Un insieme di direttive stabilisce come utilizzare i selettori, quali sono le proprietà che ogni elemento può avere e il tipo di valori che esse possono assumere.

La separazione del contenuto di un testo dalla sua presentazione è importante: grazie ai CSS posso modificare completamente la formattazione di un documento lasciando inalterata la sua struttura oppure posso associare a un documento più formattazioni e scegliere quale utilizzare in base a determinate condizioni (ad esempio una formattazione per la stampa e una per la visualizzazione su schermo).

1.2.5 JavaScript e jQuery

JavaScript è un linguaggio di scripting orientato agli oggetti molto utilizzato per modificare dinamicamente un documento HTML e rendere interattivo un sito web.

Con linguaggio di scripting si intende un linguaggio di programmazione interpretato, un linguaggio, cioè, che non deve essere compilato ma che viene eseguito direttamente da un apposito interprete. JavaScript è inoltre un linguaggio lato client, viene cioè eseguito dall'interprete presente nel browser che visualizza il sito. Il fatto che sia orientato agli oggetti permette di dichiarare classi e definire costruttori per gli oggetti.

jQuery è un framework JavaScript, cioè una libreria di funzioni JavaScript.
Si propone come obiettivo quello di semplificare la programmazione delle pagine HTML offrendo una solida libreria di funzioni già pronte, da richiamare ogni qual volta si desideri eseguire determinati compiti, evitando così di scrivere lunghe porzioni di codice JavaScript.
Il codice sintetico di jQuery è appunto uno dei suoi punti di forza, insieme l'attenzione verso la compatibilità cross browser e al fatto di avere un'attiva comunità di sviluppatori.

2.1.6 TEI

La TEI (*Text Encoding Initiative*) è un consorzio internazionale che ha come obiettivo lo sviluppo, il mantenimento e la diffusione di uno standard per la rappresentazione dei testi in formato digitale.
La TEI definisce gli schemi di codifica XML per la marcatura del testo; questi schemi sono documentati dalle linee guida, le *Guidelines for Electronic Text Encoding and Interchange*[15].

Gli elementi forniti dalla TEI per la marcatura sono organizzati in forma modulare così da permettere un'elevata personalizzazione e, attraverso la combinazione di diversi moduli, la creazione di uno schema quanto più adatto possibile alle caratteristiche di un determinato testo.

Poiché la TEI cerca di descrivere oggetti che sono continuamente sotto studio e poiché gli approcci di ricerca sono in costante evoluzione, la realizzazione degli schemi TEI richiede continuo sviluppo e ricerca.
L'attuale versione schemi TEI, rilasciata il 1 novembre 2007, è la P5 (Proposal 5)[16] e descrive circa 500 elementi. Questa versione introduce numerose novità e miglioramenti, come l'aggiunta di nuovi moduli per specificare caratteristiche non presenti nelle versioni precedenti, l'aggiunta di elementi per aggiornare i moduli esistenti, la divisione degli elementi in classi per facilitarne la personalizzazione o la possibilità di esprimere lo schema di codifica non solo come DTD, ma anche nel linguaggio RELAX schema NG.

2.2 EVT Builder: funzionamento in breve

EVT builder è stato sviluppato all'interno del progetto EVT con lo scopo primario di creare un software per la visualizzazione e la consultazione dell'edizione digitale del Vercelli Book, ma anche con l'obiettivo, più ampio e ambizioso, di creare un

[15] Si veda: http://www.tei-c.org/Guidelines/.
[16] Si veda: http://www.tei-c.org/Guidelines/P5/.

software per la visualizzazione e la consultazione di tutte le edizioni digitali codificate secondo lo standard TEI P5.
Il sistema è stato quindi progettato tenendo di conto della tipologia di destinatario, includendo in questa categoria sia l'utente finale che utilizza l'applicazione web, sia il filologo che vuole pubblicare un'edizione digitale, uno studioso che probabilmente possiede limitate competenze informatiche.
Partendo da questo presupposto è stato creato un sistema semplice da utilizzare e che non richieda al destinatario la conoscenza delle tecnologie necessarie allo sviluppo di un'applicazione web.

All'utente che vuole creare un'edizione digitale è richiesto semplicemente di:

1. fornire le scansioni del manoscritto e inserirle nella cartella *scans* nel formato: *pagina.jpg* (es. 104v.jpg);
2. copiare il proprio documento XML, contenente la trascrizione del manoscritto, e tutti gli eventuali file necessari (ad esempio quelli contenenti lo schema di codifica) nella cartella *builder_pack*;
3. effettuare la trasformazione del documento XML utilizzando il foglio di stile *evt_builder.xsl*.

Perché l'applicazione finale risulti ben realizzata, lo sviluppatore deve invece:

1. progettare con cura le trasformazioni XSLT per generare correttamente tutte la pagine HTML necessarie;
2. creare file JavaScript (a cui le pagine HTML fanno riferimento) contenenti tutte le funzioni necessarie per la visualizzazione e la navigazione del manoscritto, per gestire gli strumenti che arricchiscono il visualizzatore e per gestire le interazioni dell'utente con l'applicazione;
3. fornire dei fogli di stile CSS per impostare un layout corrispondente ai requisiti del visualizzatore.

Per poter svolgere correttamente il mio compito ho dovuto conoscere, e talvolta modificare, tutti e tre questi aspetti. Si è rivelato di fondamentale importanza l'approfondimento del primo.

2.3 Le trasformazioni XSLT

2.3.1 I tipi di trasformazione

Il sistema di trasformazione è articolato in forma modulare.
Il file utilizzato per la trasformazione, *evt_builder.xsl*, include tutti gli altri moduli *evt_builder-*.xsl*.

La trasformazione ha essenzialmente due finalità:

- generare la pagina *index.html* che costituisce il punto di partenza per l'applicazione e per la navigazione del manoscritto;
- per ogni edizione prevista dalle impostazioni, generare le pagine HTML corrispondenti ai fogli del manoscritto.

La generazione delle pagine HTML è possibile grazie all'istruzione `<xsl:result-document>` introdotta nella versione 2.0 di XSLT. Questa funzione permette di ottenere come risultato della trasformazione più documenti di output, così da poter dividere un documento XML in file più piccoli.[17]

La comprensione del funzionamento della parte di trasformazione riguardante la creazione del file *index.html* non è rilevante ai fini di questo lavoro. Le modifiche che ho eseguito erano finalizzate esclusivamente all'aggiunta degli elementi HTML necessari alla gestione degli strumenti da me introdotti (ad esempio i pulsanti per attivare e disattivare gli strumenti) all'interno del template che genera la struttura HTML della pagina; tutti gli altri template sono stati lasciati inalterati.

Di essenziale importanza si è rivelato, invece, l'approfondimento del meccanismo di generazione delle pagine HTML relative alle varie tipologie di edizione. La complessità di questa trasformazione è dovuta alla difficoltà di suddivisione in pagine del documento XML prevista EVT Builder; la marcatura XML utilizzata dalla TEI per evitare la sovrapposizioni gerarchiche rende difficile questa operazione.

[17] Michael Kay, *XSLT 2.0 and XPath 2.0 Programmer's Reference*, p. 445.

2.3.2 Sovrapposizioni gerarchiche nella rappresentazione di strutture di dati

La codifica dichiarativa suggerita dalla TEI consente di rappresentare strutture testuali su più livelli.
È facile immaginare con quanta facilità il livello della struttura astratta di un testo e quello della struttura fisica della sua rappresentazione concreta si sovrappongano: basti pensare alla divisione in versi e a quella fisica in righe (in particolare nei manoscritti medievali) oppure a quella in capitoli e a quella in fogli; tutte le volte che non c'è corrispondenza fra i due elementi (e questo accade pressoché sempre), i due tipi di struttura si sovrappongono.[18]
XML impone, tuttavia, una rigida struttura gerarchica volta alla creazione di un unico albero, escludendo così la possibilità di rappresentare gerarchie sovrapposte.

Fra le varie soluzioni che la TEI propone al fine di rappresentare queste strutture sovrapposte per la codifica del Vercelli Book è stata impiegata quella che suggerisce l'utilizzo di elementi *milestone*.[19]
Gli elementi milestone sono elementi vuoti utilizzati come "delimitatori di confine": posti all'inizio di una sezione, ne indicano l'inizio; l'incontro del successivo elemento dello stesso tipo indica la fine di tale sezione e l'inizio di una nuova.

Nella codifica del Vercelli Book sono stati usati due tipi di elementi milestone:

- elemento <pb/>[20] (*page break*) per indicare l'inizio di una pagina tipografica;
- elemento <lb/> [21] (*line break*) per indicare l'inizio di una nuova linea tipografica.

[18] Altri esempi di gerarchie sovrapposte le soluzioni adottate per risolvere questo problema sono disponibili sul sito della TEI: http://www.tei-c.org/release/doc/tei-p5-doc/en/html//NH.html.
[19] Utilizzo di elementi milestone per evitare la sovrapposizione di diverse gerarchie: http://www.tei-c.org/release/doc/tei-p5-doc/en/html//NH.html#NHBM.

[20] Per ulteriori informazioni sull'elemento pb si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-pb.html.
[21] Per ulteriori informazioni sull'elemento lb si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-lb.html.

Di fatto questo metodo permette di aggirare il problema della rappresentazione di gerarchie sovrapposte, ma introduce un problema consistente: le sezioni indicate in questo modo non sono annidate in nessun elemento dell'albero e diventa significativamente complesso processarle utilizzando le tecnologie XML.

In EVT Builder questo crea problemi sia per la paginazione (che richiede la selezione del contenuto dei singoli fogli), sia per l'introduzione del collegamento testo-immagine (che richiede la selezione del contenuto delle singole righe).
I problemi, in sintesi, sono dovuti al fatto che:

- è necessaria la selezione del contenuto dei singoli fogli o delle singole righe, ma i fogli e le righe del manoscritto sono indicati utilizzando un elemento milestone;
- tutti gli altri elementi possono trovarsi in parte in un foglio/riga e in parte nel successivo, ed essere quindi spezzati nel momento in cui si effettua la selezione dei fogli/righe.

2.3.3 Paginazione in EVT Builder prima della mia collaborazione

Nella versione di EVT Builder precedente alla mia collaborazione la paginazione era effettuata utilizzando delle variabili che venivano passate da un template all'altro.
La paginazione era ottenuta tramite una serie di trasformazioni eseguite grazie moduli descritti qui di seguito:

1. builder_pack/modules/**evt_builder-main**.xsl: tramite l'istruzione `<xsl:for-each>`, per ogni `<pb>` viene creata una (o più[22]) pagina HTML e chiamato il template *data_structure*.
   L'istruzione *for-each* serve appunto per selezionare, tramite l'utilizzo di XPath, una sequenza di elementi (in questo caso tutti gli elementi `<pb>`) e ripetere lo stesso processo per ogni elemento della sequenza[23].

[22] Le pagine sono più di una quando dobbiamo creare più edizioni dello stesso foglio, ad esempio la versione facsimile di un foglio e la versione diplomatica dello stesso foglio.
[23] Michael Kay, *XSLT 2.0 and XPath 2.0 Programmer's Reference*, p. 322.

2. builder_pack/modules/html_build/**evt_builder-callhtml**.xsl: questo file contiene, fra gli altri, il template *data_structure*; il template crea la struttura HTML di base per la pagina che dovrà rappresentare il foglio del manoscritto. Questo template richiama a sua volta un altro template, *call_body*, necessario per selezionare il contenuto testuale da trasformare e inserire nella nuova pagina HTML.

3. builder_pack/modules/elements/**evt_builder-call_main**.xsl: il template *call_body* serve a richiamare i template necessari per la trasformazioni degli elementi TEI, passandogli di volta in volta i tre parametri essenziali per il loro corretto funzionamento: il parametro che indica l'inizio della pagina corrente (`pp_start`), quello che indica la fine della pagina corrente (`pp_end`) e quello che indica l'edizione di output desiderata (`output`)

4. Gli altri file contenuti nella cartella builder_pack/modules/**elements/** contengono tutti i template per la trasformazione dei vari elementi TEI.

Esempio di uno di questi template:

```
<xsl:template match="tei:damage">
   <xsl:param name="pp_start"/>
   <xsl:param name="pp_end"/>
   <xsl:param name="output"/>
   <xsl:if test="not(following::tei:pb[@n=$pp_start/@n]) and
not(preceding::tei:pb[@n=$pp_end/@n])">
      <xsl:if test="$output=$edition_array[1]">
         <span class=”damage”>
            <xsl:apply-templates>
               <xsl:with-param name="pp_start" select="$pp_start"/>
               <xsl:with-param name="pp_end" select="$pp_end"/>
            </xsl:apply-templates>
         </span>
      </xsl:if>
   </xsl:if>
</xsl:template>
```

Quando viene trovato un elemento `<damage>`[24] questo viene trasformato in un elemento `<span class='damage'>`[25] se e solo se esso si trova nella pagina che stiamo trasformando (quindi fra i due <pb> indicati da `pp_start` e `pp_end`) e se stiamo effettuando la trasformazione per l'edizione facsimile (cioè se la variabile `output` corrisponde a `edition_array[1]`)[26].
Vengono poi invocati, passandogli i soliti tre parametri, i template di trasformazione per gli elementi TEI, così che venga processato anche il contenuto dell'elemento.
Tramite questa serie di chiamate ricorsive viene processato tutto il contenuto testuale della pagina e viene creato il contenuto da inserire nella pagina HTML.

2.3.4 Modifiche al sistema di paginazione

Per fare a meno delle variabili `pp_star` e `pp_end` ho deciso di utilizzare l'istruzione `<xsl:for-each-group>`[27].
Quest'istruzione, introdotta da XSLT 2.0, seleziona un set di elementi, li divide in gruppi seguendo un determinato criterio e infine processa il contenuto di ogni gruppo.
Per la selezione dei gruppi ho utilizzato il criterio di selezione `group-starting-with`: questo criterio divide il blocco di elementi in gruppi creando un nuovo gruppo ogni volta che incontra l'elemento indicato come criterio di selezione.
Combinando queste due regole come segue:

```
<xsl:for-each-group       select="//tei:text/tei:body/tei:div/node()"
group-starting-with="//tei:pb">
```

ho potuto fare in modo che il contenuto testuale del documento XML fosse suddiviso in gruppi e che a ogni <pb> iniziasse un nuovo gruppo.
In questo modo ogni gruppo corrisponde a un foglio e, se all'interno dell'istruzione invoco un nuovo template, quest'ultimo agisce sull'intero blocco (quindi sull'intero

[24] Elemento utilizzato per indicare una parte di testo che contiene danni. Per ulteriori informazioni si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-damage.html.
[25] Per ulteriori informazioni sull'utilizzo dell'elemento *span* nelle edizioni si veda il capitolo 3, paragrafo 3.1.
[26] Per ulteriori informazioni sulle edizioni del testo e sulle trasformazioni si veda il capitolo 3, paragrafo 3.1.
[27] Michael Kay, *XSLT 2.0 and XPath 2.0 Programmer's Reference*, p. 329.

foglio) e non più solamente sull'elemento <pb> come accadeva utilizzando l'istruzione <xsl:for-each> (motivo per il quale dovevo poi utilizzare i parametri pp_start e pp_end per selezionare la porzione di testo corrispondete al foglio indicato dal <pb>).

Tuttavia, utilizzando l'istruzione <xsl:for-each-group>, il fatto che gli elementi TEI possano trovarsi in parte in una pagina e in parte nella successiva risulta essere un problema: se uno degli elementi contenuti nel gruppo selezionato utilizzando il criterio group-starting-with="//tei:pb" non è chiuso prima del <pb> il gruppo non termina al successivo <pb> (come dovrebbe) ma continua fino a quando l'elemento aperto non viene chiuso; così facendo viene inclusa nel gruppo una porzione di testo eccessiva.
Per evitare questo problema ho applicato l'istruzione <xsl:for-each-group> non direttamente al documento XML ma a una copia di esso che creo e nella quale tutti gli elementi che si trovano in parte prima di un <pb> (o di un <lb>) e in parte dopo vengono chiusi prima dell'elemento milestone e poi riaperti.

Tutto questo mi permette di semplificare i template per la trasformazione dei vari elementi TEI.

Dopo le mie modifiche, la paginazione avviene tramite la serie di trasformazioni eseguite grazie moduli descritti di seguito:

1. builder_pack/modules/html_build/evt_builder-copy_and_call_main.xsl: viene creata la copia con gli elementi chiusi prima di un <pb> (o di un <lb>) e nuovamente aperti dopo questo elemento; successivamente, per ogni gruppo iniziato da un <pb>, viene chiamato il template *page*.

2. builder_pack/modules/**evt_builder-main**.xsl: viene creata una (o più) pagina HTML e chiamato il template *data_structure.*

3. builder_pack/modules/html_build/**evt_builder-callhtml**.xsl: contiene, fra gli altri, il template *data_structure*; il template crea la struttura della pagina HTML che dovrà contenere il testo del foglio del manoscritto. All'interno di

data_structure vengono richiamati i template per la trasformazione degli elementi TEI impostando l'attributo `mode` sul valore corrispondente all'edizione da trasformare (ad esempio `mode="dipl"` per l'edizione diplomatica).

4. Gli altri file contenuti in builder_pack/modules/**elements** contengono i template per la trasformazione dei vari elementi TEI.

Esempio di template per la trasformazione di elementi TEI dopo le modifiche apportate:

```
<xsl:template match="tei:del" mode="dipl">
            <span class="del”>
                  <xsl:apply-templates mode="#current"/>
            </span>
      </xsl:template>
```

Come si può notare il template risulta molto più semplice:

- l'utilizzo delle variabili per indicare la pagina e l'edizione non è più necessario;
- l'utilizzo del `mode` al posto della variabile per indicare l'edizione rende il template più breve e più semplice da scrivere e dà la possibilità di separare le regole per la trasformazioni di un elemento TEI.

Quest'ultimo vantaggio è molto importante: spesso vengono creati più fogli di stile XSLT per queste trasformazioni (ad esempio: un file per la trasformazione relativa all'edizione facsimile e un file per la trasformazione dell'edizione diplomatica) oppure vengono dedicate parti diverse del documento alle varie trasformazioni (la prima parte del file per le regole di una trasformazione e la seconda parte per regole per un'altra trasformazione); utilizzando l'istruzione `<xsl:if>` per distinguere le regole relative a trasformazioni di uno stesso elemento per edizioni differenti non potevo dividere le regole su più template. Dividendo, ad esempio, un template con il `match` per un certo elemento TEI e all'interno due condizioni `<xsl:if>` (relative a due diverse trasformazioni) in due template separati contenenti ciascuno una delle

due condizioni, avrei creato due template con lo stesso `match`, ma due template con uguale `match` vanno in conflitto.
Utilizzando il mode per effettuare questa distinzione il problema non si crea: più template con lo stesso `match` e diverso `mode` non creano conflitti in quanto il `mode` non è un elemento interno al template (come `<xsl:if>`), ma un attributo del template e quindi viene valutato nella scelta del template da applicare.

IMPLEMENTAZIONE FUNZIONALITÀ

All'interno del progetto EVT ho dovuto curare la realizzazione dell'edizione facsimile e dell'edizione diplomatica del testo.
Ho poi realizzato lo strumento collegamento testo-immagine e ho curato il suo inserimento in EVT Builder per offrire all'utente la possibilità di scorrere il testo di un'edizione riga per riga evidenziando la relativa porzione di immagine, o viceversa.
Ho infine realizzato lo strumento lente di ingrandimento per offrire un ingrandimento di risoluzione maggiore rispetto a quello ottenibile con lo strumento zoom e per andare a completare il set di strumenti base per l'immagine.

3.1 Livelli di edizione del testo

L'edizione facsimile, come indica il nome, è quella che riproduce il testo originale nel modo più fedele possibile; le convenzioni utilizzate per la creazione di questa edizione sono standard: vengono riportati tutti i caratteri così come compaiono nel testo, viene rispettata la divisione in pagine, in righe e in parole, viene rispettata la punteggiatura originale, l'utilizzo delle lettere maiuscole e non vengono sciolte le abbreviazioni.
L'edizione diplomatica richiede una maggiore quantità di interventi editoriali; spesso, ad esempio, le abbreviazioni vengono estese e la parte aggiunta dal filologo viene messa graficamente in evidenza. Nella scelta delle convenzioni editoriali legate a quest'edizione alcune decisioni sono soggettive e legate alle preferenze del curatore (come ad esempio la scelta di mantenere o meno la divisione in righe e fogli, o se rispettare l'utilizzo delle maiuscole); nella creazione dell'edizione diplomatica del *Vercelli Book* è stato deciso di creare un'edizione vicina al testo originale (viene mantenuta la divisione in righe e, in parte, quella delle parole, viene mantenuto l'uso originale delle maiuscole e della punteggiatura).

Il mio compito è stato quello di realizzare i fogli di stile XSLT e CSS da inserire in EVT Builder per la generazione e la corretta visualizzazione delle due edizioni richieste de *Il sogno della Croce*.

Come è già stato illustrato nei cap. 2:

1. *Il sogno della Croce* è stato marcato utilizzando il linguaggio XML e gli schemi TEI P5;
2. durante la trasformazione XSLT effettuata dal sistema EVT Builder questo file viene diviso in porzioni più piccole corrispondenti ai fogli del manoscritto;
3. per ognuna di queste porzioni di testo vengono creati tanti file di output quanti sono quelli richiesti dalle impostazioni della trasformazione e, per la creazione dei contenuti di questi file, vengono selezionati i template corrispondenti all'edizione desiderata grazie al valore dell'attributo *mode*.

Ho impostato il software per ottenere i due livelli di edizione del testo richiesti e associato alla trasformazione per l'edizione facsimile il mode *facs* e alla trasformazione per l'edizione diplomatica il mode *dipl*. A questo punto ho dovuto creare i fogli XSLT contenenti i template per i due tipi di trasformazione. Seguendo le convenzioni editoriali stabilite per i due livelli di trasformazione richiesti, ho utilizzato i template per trasformare il contenuto degli elementi necessari e per non riportare gli elementi non necessari.

Tutti gli elementi TEI che vengono trasformati vengono inseriti in un elemento HTML `<span>` con classe *livello_di_edizione-nome_elemento_TEI* (esempio per l'elemento `<abbr>` nella trasformazione per l'edizione facsimile: *facs-abbr*).
L'elemento HTML `<span>` serve per delimitare una porzione di testo (o più elementi) senza influenzarne la visualizzazione e risulta essere il più adatto da utilizzare per la trasformazione di molti elementi TEI: permette di mantenere l'informazione di tipo semantico contenuta nella marcatura (riportando il nome dell'elemento TEI nella classe) e, all'occorrenza, di associare all'elemento con quella classe delle regole CSS per specificarne la visualizzazione e mettere in evidenza l'elemento.
Al termine della trasformazione si ottengono due pagine HTML per ogni foglio del manoscritto contenenti ciascuna il testo per il livello di edizione specificato e la marcatura necessaria per determinarne la visualizzazione.

Vediamo, ad esempio, come vengono marcate le abbreviazioni e come vengono poi trasformate per ottenere quanto richiesto dai due livelli di edizione.
Le abbreviazioni presenti nel manoscritto sono marcate utilizzando gli elementi TEI nel seguente modo:

```
<abbr><am>ũ</am></abbr><expan>u<ex>m</ex></expan>
```

<abbr>[28] contiene un'abbreviazione di qualsiasi tipo.
<am>[29] contiene i grafemi utilizzati nel manoscritto per indicare l'abbreviazione.
<expan>[30] contiene l'espansione dell'abbreviazione.
<ex> [31] contiene una sequenza di lettere aggiunte dal filologo per espandere un'abbreviazione.

Si noti che vengono usati direttamente caratteri UTF-8 grazie ai font Unicode che si basano sulle specifiche MUFI[32]. Nei fogli di stile CSS utilizzati nel visualizzatore viene associato alle pagine il font per la visualizzazione dei caratteri medievali Junicode[33].

Nell'edizione facsimile le abbreviazioni non vengono estese; la trasformazione seleziona la parte di codice relativa all'abbreviazione e non riporta la parte di codice relativa all'estensione ottenendo la seguente porzione di pagina HTML:

```
<span class="facs-abbr"><span class="facs-am">ũ</span></span></span>
```

Questo elemento non richiede una formattazione speciale quindi non vengono create regole CSS.

[28] Per maggiori informazioni sull'elemento abbr si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-abbr.html.
[29] Per maggiori informazioni sull'elemento am si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-am.html.
[30] Per maggiori informazioni sull'elemento expan si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-expan.html.
[31] Per maggiori informazioni sull'elemento ex si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-ex.html.
[32] Progetto che ha come obiettivo la definizione dei caratteri necessari per la digitalizzazione dei testi medioevali e l'inserimento di tali caratteri nello standard Unicode, per maggiori informazioni si veda: http://www.mufi.info/.
[33] Sito ufficiale di Junicode: http://junicode.sourceforge.net/.

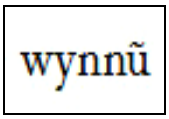

Fig. 1: Risultato finale di un abbreviazione nell'edizione facsimle.

Nell'edizione diplomatica le abbreviazioni vengono estese; la trasformazione seleziona la parte di codice relativa all'estensione mentre non riporta la parte di codice relativa all'abbreviazione, ottenendo la seguente porzione di pagina HTML:

```
<span class="dipl-expan">u<span class="dipl-ex">m</span></span>
```

La parte dell'estensione aggiunta dal filologo viene solitamente mostrata in corsivo quindi ho creato la regola CSS:

```
SPAN.dipl-ex {font-style: italic;}
```

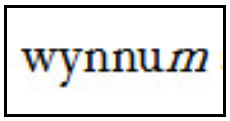

Fig. 2: Risultato finale di un abbreviazione nell'edizione diplomatica.

Seguendo le convenzioni editoriali stabilite per i due livelli di trasformazione richiesti, ho ripetuto queste operazioni (trasformazione degli elementi ed eventuale aggiunta di regole per la formattazione) per tutti gli elementi presenti nel documento XML così che al termine della trasformazione effettuata da EVT Builder si ottengono correttamente i due livelli di edizione richiesti. Attraverso il menù dell'interfaccia grafica è possibile selezionare il livello di edizione da visualizzare e, grazie alla modalità testo-testo è possibile confrontare i due livelli di edizione.

3.2 Collegamento testo-immagine

Il collegamento fra testo e immagine è lo strumento che evidenzia e mette in relazione le righe della scansione di un foglio del manoscritto con le corrispondenti righe nell'edizione del testo, e viceversa.

Mantenendo attivo il collegamento fra le due parti è possibile scorrere il contenuto di un'edizione riga per riga e osservarne, allo stesso momento, la sua forma originale grazie alla scansione o, viceversa, scorrere linea per linea un'immagine e osservarne la trascrizione in una determinata edizione.

3.2.1 La codifica TEI necessaria per collegare testo e immagine

Se all'interno del file XML sono marcati sia la scansione di un foglio, sia la sua trascrizione, possiamo collegare le due parti utilizzando gli ID degli elementi e alcuni attributi TEI creati appositamente per i collegamenti.
Gli schemi TEI, infatti, oltre ad offrire elementi per la marcatura delle trascrizioni di una fonte, offrono strumenti utili per inserire nei documenti XML dei riferimenti a immagini e ad aree di un'immagine.

Per descrivere la marcatura delle immagini è necessario utilizzare i seguenti elementi TEI:

- `<facsimile>`[34]: indica la porzione del file nella quale una fonte scritta viene rappresentata attraverso dei puntatori a immagini relative alla fonte.
4. `<surface>`[35]: da inserire all'interno di *facsimile*, definisce una superficie scritta (nel nostro caso il foglio del manoscritto);
5. `<graphic>`[36]: da inserire all'interno di *surface*, indica il percorso relativo all'immagine a cui viene fatto riferimento;
6. `<zone>`[37]: indica zone rettangolari di interesse (nel nostro caso le righe di un foglio); per delimitare le zone rettangolari è necessario inserire le coordinate dei quattro angoli del rettangolo fra gli attributi dell'elemento.

Per la creazione di questi elementi è utile sfruttare software che eseguono compiti come il calcolo delle coordinate dei rettangoli.

[34] Per maggiori informazioni sull'elemento facsimile si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-facsimile.html.
[35] Per maggiori informazioni sull'elemento surface si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-surface.html.
[36] Per maggiori informazioni sull'elemento graphic si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-graphic.html.
[37] Per maggiori informazioni sull'elemento zone si veda: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-zone.html.

Per la marcatura delle scansioni de *Il sogno della Croce* è stato utilizzato *IMT - Image Markup Tool*[38].
IMT è un software open source che, attraverso un'interfaccia grafica molto intuitiva e semplice, permette di caricare le immagini, individuarvi delle aree di forma rettangolare, aggiungervi delle annotazioni e salvare queste annotazioni in formato XML utilizzando la marcatura TEI.

Per poter effettuare il collegamento fra una riga della trascrizione (elemento milestone `<lb>`) e una riga dell'immagine (elemento `<zone>`), è necessario che a entrambe sia assegnato un ID e che i due elementi si richiamino nel seguente modo:

5. per ogni `<zone>` viene impostato l'attributo *@corresp*[39] sul valore dell'ID del relativo `<lb>`;
6. per ogni `<lb>` viene impostato l'attributo *@facs*[40] sul valore dell'ID del relativo `<zone>`.

Possiamo vedere come questa codifica sia stata utilizzata per "Il sogno della croce":

```
<zone rend="visible" rendition="Line" lrx="1072" lry="512" ulx="210"
uly="459" xml:id="VB_line_104v_07" corresp="#VB_lb_104v_07"/>
```

```
<lb facs="#VB_line_104v_07" n="7" xml:id="VB_lb_104v_07"/><hi
rend="init3.1">&hunc;</hi><hi rend="cap">W</hi>æt ic &slong;wefna
c&ydot;&sins;t secgan wylle</l>
<l n="2"><sic>hæt</sic><corr>hwæt</corr> me&ms;ge&ss;mætte<lb
xml:id="VB_lb_104v_08" facs="#VB_line_104v_08" n="8"/>
```

Un file codificato in questo modo offre tutto il necessario per poter creare un collegamento fra testo e immagine: è quindi possibile trasformare il documento XML in un file HTML che mantiene le informazioni dei riferimenti ai vari elementi, aggiungervi uno script per gestire il collegamento fra tali elementi e un foglio di stile CSS per determinarne la visualizzazione.

[38] Sito ufficiale di Image Markup Tool: http://tapor.uvic.ca/~mholmes/image_markup/index.php.
[39] Attributo che punta alla parte di immagine che corrisponde al contenuto dell'elemento.
[40] Attributo che punta alla parte di documento che corrisponde in qualche modo al contenuto dell'elemento.

3.2.2 Realizzazione dello strumento

3.2.2.1 La Web View di IMT

IMT prevede la realizzazione di un file di output chiamato *web view*.
L'idea di base della *web view* è quella di trasformare il file XML in una pagina HTML nel quale immagine e annotazioni siano collegate. La *web view* produce un output con immagine a dimensione fissa: una volta stabilite le dimensioni di quest'immagine crea tutti i file necessari per la generazione della pagina HTML e per la gestione del collegamento fra immagine e annotazioni.

In sintesi la generazione della *web view* funziona come segue:

- un foglio di stile XSLT trasforma il file XML e crea una pagina HTML in cui gli elementi *zone* diventano aree rettangolari che vengono poste sopra l'immagine e ogni riga della trascrizione viene delimitata inserendola in un elemento specifico;
- uno script JavaScript contiene funzioni per cambiare la classe degli elementi (e di conseguenza la loro formattazione) al passaggio oppure al click del mouse. Oltre a cambiare la classe di tale elemento, cambiano anche quella dell'elemento cui fanno riferimento, così che vengano evidenziati o selezionati entrambi.
- un foglio di stile CSS definisce la formattazione delle possibili classi degli elementi.

I file utilizzati per la creazione della web view sono un'utile base per la realizzazione del collegamento testo-immagine necessario per EVT Builder, ma hanno richiesto numerose modifiche e aggiunte.

3.2.2.2 La trasformazione

Per quanto riguarda il foglio di stile XSLT, dobbiamo dire che buona parte del file crea elementi che a noi non servono; questo perché la trasformazione utilizzata dalla *web view* serve per creare un'intera pagina HTML, mentre a EVT Builder serve

solamente la parte di trasformazione (da collegare poi con le altre trasformazioni effettuate dal sistema) volta alla creazione degli elementi (aree e righe di testo) da inserire nelle pagine da lui create.

La trasformazione per la creazione delle aree funziona nel seguente modo:

- per ogni `<zone>` presente nel file XML viene creato un elemento HTML `<div>` con classe *Area*;
- l'attributo ID dell'elemento `<zone>` viene passato al nuovo elemento;
- l'attributo contenente le coordinate dell'angolo in alto a sinistra del rettangolo `<zone>` viene utilizzato per indicare la posizione del nuovo elemento;
- grazie agli attributi contenenti le coordinate degli altri angoli del rettangolo `<zone>`, vengono calcolate altezza e larghezza del nuovo elemento;
- vengono infine associate all'elemento delle proprietà CSS e delle funzioni JavaScript da eseguire in base agli eventi (click o passaggio del mouse).

La parte di trasformazione legata alla creazione degli elementi inserisce ogni riga in un elemento con classe *AnnMenuItem* se e solo se viene trovato nel file un elemento `<zone>` che corrisponde a quella riga (cioè che abbia come ID il valore indicato nell'attributo *@facs* della riga).
Per come era scritta la regola in IMT, tuttavia, la trasformazione non riusciva a catturare il contenuto della riga: questo è dovuto al problema relativo all'utilizzo degli elementi milestone nella codifica del *Vercelli Book* per indicare le righe. Per riuscire a selezionare il contenuto della riga è stato necessario lo studio di EVT Builder e le modifiche già illustrate nel cap. 2. La soluzione per la selezione del contenuto delle righe è la stessa adoperata per la paginazione, cioè l'utilizzo dell'istruzione `<xsl:for-each-group>` e del criterio di selezione `group-starting-with="//tei:lb"`.

Il foglio di stile XSLT per la creazione del collegamento è stato inserito all'interno del sistema modulare della trasformazione effettuata da EVT Builder: dopo la creazione della pagina HTML viene controllato se è presente il `<surface>` relativo alla pagina e, se viene trovato, viene chiamato questo file il quale, dopo aver inserito il contenuto delle righe negli elementi *AnnMenuItem*, richiama i template per la

trasformazione degli elementi TEI da applicare alla riga; se non viene trovato il `<surface>`, vengono chiamati direttamente i template per la trasformazione degli elementi TEI da applicare all'intera pagina.

3.2.2.3 Gestione del collegamento

Per quanto riguarda lo script JavaScript possiamo dire che, anche in questo caso, la *web view* crea molte funzioni a noi non necessarie, come quelle per posizionare l'immagine o per calcolare la dimensione del riquadro contenente il testo in base alla dimensione dell'immagine. Ho selezionato le funzioni utili (cioè quella per attivare il collegamento e quelle riguardanti gli eventi legati al passaggio o al click del mouse sugli elementi) e ho aggiunto altre funzioni necessarie per il corretto funzionamento dello strumento all'interno di EVT Builder.

Il collegamento è uno strumento utile quando si vogliono confrontare testo e immagine riga per riga, ma in altri momenti può risultare di intralcio alla navigazione; quindi ho aggiunto una funzione per disattivarlo e una funzione per fare lo *switch* fra attivazione e disattivazione dello strumento (entrambe le funzioni vengono invocate da elementi presenti nella pagina *index.html*).

È stato poi necessario armonizzare il funzionamento del collegamento con quello del plugin relativo allo zoom. Caratteristica principale dello strumento zoom è quella di cambiare la dimensione dell'immagine tutte le volte che viene cambiato il fattore di ingrandimento, ma, come abbiamo già detto, il collegamento creato da IMT funziona solo su immagini statiche e, al variare delle dimensioni dell'immagine, la dimensione dei rettangoli resta invariata. Per evitare questo problema ho modificato le funzioni che gestiscono il posizionamento dei riquadri sull'immagine facendo in modo che posizione e dimensioni dei riquadri sia calcolata in rapporto all'immagine. Ho poi creato una funzione per aggiornare dimensione e posizione dei rettangoli quando viene cambiato il fattore di zoom e ho creato una funzione per cambiare la posizione dei riquadri quando utilizzo il drag dello zoom (e quindi sposto l'immagine all'interno del riquadro immagine).

Infine il collegamento deve integrarsi con il funzionamento dell'interfaccia (va ricordato che EVT Builder è formato da un'unica pagina HTML che carica di volta in volta testo e immagine richiesti); quando si seleziona una nuova pagina da caricare, se lo strumento collegamento è attivo, esso deve essere disattivato e attivato nuovamente con i dati della nuova pagina, così come deve essere disattivato se si passa alla modalità testo-testo. Ho perciò creato delle funzioni per effettuare questi controlli.

Terminate le modifiche allo script JavaScript, ho modificato il foglio di stile CSS per scegliere la formattazione più adatta da assegnare agli elementi.
Le regole contenute nel foglio di stile CSS definiscono la formattazione delle righe di testo e dei riquadri posti sopra le righe dell'immagine quando:

- essi hanno la loro classe: *HighlightedAnnMenuItem* o *Area*;
- quando essi hanno la classe che gli viene assegnata dallo script quando sono evidenziati dal passaggio del mouse: *HighlightedAnnMenuItem* o HighlightedArea;
- quando essi hanno la classe che gli viene assegnata dallo script quando sono selezionati dal click del mouse: *SelectedAnnMenuItem* o *SelectedArea*.

Terminate le modifiche e le aggiunte, ho ottenuto uno strumento collegamento testo immagine perfettamente funzionante all'interno di EVT Builder e insieme allo strumento zoom.

Fig. 3: Collegamento testo-immagine

3.3 La lente di ingrandimento

Nel visualizzatore è importante avere immagini di alta qualità per poter osservare e analizzare tutti i dettagli presenti nei fogli dei manoscritti.

La digitalizzazione del *Vercelli Book* è stata effettuata utilizzando una fotocamera digitale ad alta risoluzione; il risultato di questa operazione sono immagini di alta qualità: immagini a 600dpi, salvate in formato TIFF e grandi circa 13 MB l'una. Queste immagini sono ovviamente troppo grandi per essere utilizzate in un sito web. Per essere utilizzate all'interno del visualizzatore, le immagini sono state salvate a risoluzione minore e convertite in formato JPG; queste nuove immagini hanno tutte dimensione inferiore a 1GB e sono quindi adatte alla distribuzione sul web. Tuttavia, il formato JPG è un formato molto complesso e il suo utilizzo comporta una perdita di qualità delle immagini rispetto a quelle originali. Un ingrandimento eccessivo di queste immagini accentua la mancanza di risoluzione perciò lo strumento zoom di EVT Builder è stato impostato per ingrandire l'immagine fino a raggiungere il 140% delle sue dimensioni reali.

Lo strumento lente è stato inserito in EVT Builder per offrire all'utente un metodo di ingrandimento alternativo e per permettere di apprezzare maggiormente la qualità dei dettagli. Questo è possibile perché la lente di ingrandimento non utilizza l'immagine di base per effettuare l'ingrandimento, ma un'altra immagine di dimensioni e risoluzione maggiore.
Per gli ingrandimenti della lente, le immaginati TIFF sono state salvate in formato JPEG utilizzando però una compressione minore, così da mantenere una maggiore qualità senza comunque raggiungere dimensioni eccessive (tutte queste immagini hanno dimensione inferiore ai 2 GB). Queste immagini sono state inserite nella cartella *scans* nel formato: *pagina_big.jpg* (es. 104v_big.jpg).

Come base per la realizzazione della lente ho utilizzato un plugin jQuery.

3.3.1 Il plugin

Un plugin è un'estensione del framework (in questo caso di jQuery) creata per ampliarne le funzioni. Un buon plugin deve essere cross browser, avere una documentazione chiara e comprensibile, avere un codice ordinato e ben commentato e avere opzioni per permettere la sua personalizzazione da parte dell'utente.

Per creare lo strumento lente di ingrandimento di EVT builder era necessario utilizzare un plugin:

- per lenti di ingrandimento multi risoluzione (quindi che utilizza due immagini, e non una sola come quello dello zoom);
- con una lente rettangolare (di dimensioni impostabili da chi configura il plugin) e che possa essere spostata sull'immagine mediante il trascinamento da parte del cursore del mouse;
- con un riquadro di ingrandimento che segue gli spostamenti del riquadro della lente.

Dopo varie ricerche su internet ho deciso di utilizzare il plugin *jqZoom Evolution*[41], plugin realizzato da un ingegnere italiano e distribuito secondo la licenza BSD[42].
Ho scelto questo plugin perché rispetta tutti i criteri di un buon plugin e possiede alcune delle caratteristiche di cui abbiamo bisogno: utilizza due immagini di dimensioni diverse e stesse proporzioni per creare una lente di ingrandimento multi risoluzione e offre numerose opzioni per la personalizzazione, fra cui quella per spostare la lente attraverso il trascinamento del cursore del mouse sull'immagine.

Il plugin così come è stato creato, tuttavia, costituisce solo la base per quello che andrà inserito in EVT Builder. È necessario apportare varie modifiche al plugin prima che esso soddisfi tutti i requisiti richiesti: le dimensioni della lente sono personalizzabili solo indirettamente (perché dipendono da quelle del riquadro dell'ingrandimento e dal rapporto fra le due immagini) e il riquadro dell'ingrandimento ha una posizione statica.

Prima di comprendere le modifiche apportate al plugin è necessario conoscere gli elementi che esso crea all'interno della pagina HTML in cui è utilizzato:

5. `<div>` con classe *zoomPad*: è il contenitore dell'immagine piccola della quale dobbiamo creare l'ingrandimento;

[41] Sito ufficiale del plugin: http://www.mind-projects.it/projects/jqzoom/.
[42] Attuale versione della licenza: http://opensource.org/licenses/bsd-license.php.

6. `<div>` con classe *zoomPup*: è il rettangolo che si sposta all'interno di *zoomPad* e rappresenta l'area da ingrandire;
7. `<div>` con classe *zoomPreload* è il riquadro di attesa che viene mostrato mentre viene caricata l'immagine più grande;
8. `<div>` con classe *zoomWrapperImage*: è il contenitore dell'immagine più grande;
9. `<div>` con classe *zoomWindow*: è la porzione di immagine grande che viene utilizzata per l'ingrandimento

Fig. 4: elementi creati dal plugin per la lente di ingrandimento

3.3.2 Le modifiche del plugin

Per rendere il plugin il più "flessibile" possibile ho apportato alcune modifiche.

Il plugin è stato originariamente realizzato per mostrare le due immagini alla loro dimensione reale (e non a dimensioni impostabili dall'utente). Una volta impostate le dimensioni del riquadro di ingrandimento, il plugin calcola il rapporto fra l'immagine grande e quella piccola e, in base a questo rapporto, calcola le dimensioni che deve avere il riquadro della lente rispetto al riquadro dell'ingrandimento.
La prima modifica è stata apportata per fare in modo che le dimensioni personalizzabili dall'utente fossero quelle della lente e che, di conseguenza, quelle calcolate dal plugin fossero quelle del riquadro dell'ingrandimento (l'esatto contrario del funzionamento originale); in questo modo l'utente può scegliere, ad esempio, l'altezza della lente in base all'altezza media di una riga del manoscritto.
Ho inoltre preso in considerazione la possibilità che l'utente non voglia utilizzare per l'ingrandimento l'immagine grande al 100% delle sue dimensioni, quindi ho aggiunto un'opzione (*zoomRatio*) per indicare il rapporto fra le dimensioni reali dell'immagine grande e quelle da utilizzare.

Al termine delle modifiche le dimensioni dei vari elementi sono calcolate nel seguente modo:

4. Il filologo (o chi configura EVT Builder) può personalizzare i valori di default utilizzati per altezza lente, larghezza lente e *zoomRatio*;
5. le dimensioni dell'immagine grande vengono impostate come *dimensioni reali immagine * zoomRatio*;
6. le dimensioni del riquadro dell'ingrandimento sono uguali a quelle della *lente * rapporto immagini*, dove *rapporto immagini* indica il rapporto fra le dimensioni dell'immagine piccola e le dimensioni impostate per l'immagine grande.

L'altra importante modifica che ho eseguito è stata fatta per fare in modo che il riquadro di ingrandimento (*zoomWindow*) seguisse quello della lente (*zoomPup*).
Si noti che i rettangoli vengono posizionati inserendo le coordinate del loro angolo in alto a sinistra e che per fare sì che *zoomWindow* sia posizionato correttamente sopra *zoomPup* è necessario che il centro dei due rettangoli coincida.

Nel momento in cui il plugin viene attivato, devo utilizzare le coordinate dell'angolo di *zoomPup* e le sue dimensioni per risalire alle coordinate del centro e, a partire da questo punto e dalle dimensioni di *zoomWindow*, calcolare le coordinate in cui posizionare il suo angolo. A ogni spostamento di *zoomPup*, devo poi prendere le coordinate del mouse (che corrispondono a quelle del centro) e calcolare nuovamente quelle per posizionare nuovamente *zoomWindow*.

3.3.3 Inserimento in EVT Builder

L'utilizzo del plugin richiede l'inclusione nel file *index.html* di uno script JavaScript contenente il codice da eseguire e di un foglio di stile CSS per la personalizzazione della formattazione. Inoltre, nel codice HTML della pagina, l'immagine da ingrandire deve essere contenuta in un elemento àncora e questo deve fare riferimento al percorso dell'immagine più grande.

Ho creato un script JavaScript per gestire l'inserimento di questi elementi tutte le volte che viene caricata una nuova immagine. Ad esempio, se in modalità testo immagine, viene selezionato il foglio 104v, al caricamento dell'immagine il codice JavaScript crea la seguente porzione di codice:

```
<a href="scans/104v_big.jpg" title="image_ 104v_big"
class="magnifier">
      <img src=" scans/104v.jpg " title=" image_ 104v ">
</a>
```

Si noti che la classe *magnifier* viene utilizzata come selettore dal plugin, serve cioè per indicare l'elemento su cui esso deve agire.
Il codice JavaScript controlla anche le dimensioni dell'immagine: altezza e larghezza sono impostate per fare in modo che la larghezza massima dell'immagine non superi i 445px, cioè la larghezza minima che può raggiungere, al momento, il riquadro che contiene l'immagine. In questo modo, su desktop di dimensioni limitate, un'immagine rettangolare orientata verticalmente (come quella dei fogli del manoscritto) riempie tutto lo spazio disponibile in larghezza (i 445px), mentre in

lunghezza supera lo spazio disponibile e viene inserita una barra di scorrimento per permettere di spostarsi su e giù.
Questo è risultato essere il compromesso migliore per garantire una buona visualizzazione su tutti i tipi di schermo e una buona consultazione: un'immagine a dimensioni maggiori genera su schermi piccoli una barra di scorrimento sia in verticale sia in orizzontale e risulta scomoda da esplorare; un'immagine che si adatta sia in altezza sia in larghezza agli schermi più piccoli, invece, risulta essere troppo piccola perché l'utente possa esaminarla.

Sempre all'interno di questo file ho impostato le opzioni del plugin.
Ho poi aggiunto una funzione da associare a un elemento della pagina HTML per fare lo *switch* fra lo strumento zoom e lo strumento lente (i due strumenti funzionano in modo esclusivo: o è attivo uno o è attivo l'altro) e ho creato una funzione da richiamare in vari punti dello script JavaScript che gestisce l'interfaccia per fare in modo che quando viene cambiato il foglio del manoscritto venga mantenuto attivo lo strumento che era stato selezionato in precedenza.

Terminate le modifiche al plugin e aggiunti gli script necessari a EVT Builder ho ottenuto un valido strumento per offrire un metodo di ingrandimento alternativo e più preciso rispetto a quello offerto dallo strumento zoom.

Conclusioni

Durante la collaborazione per lo svolgimento di questo progetto ho avuto modo di mettere in pratica le conoscenze acquisite durante il mio percorso di studi universitari e di sfruttarle per lavorare ad un progetto volto a favorire la diffusione e la fruizione di contenuti culturali.
Questo progetto è un esempio di applicazione delle tecnologie informatiche alle scienze umane e mi ha permesso di scoprire quanto, a volte, questi due mondi siano ancora lontani e quanto sia importante una figura quale quella dell'informatico umanistico, figura creata appunto per padroneggiare professionalmente gli strumenti informatici pertinenti al trattamento di contenuti culturali.

Per offrire un contributo effettivo e significativo ho dovuto conoscere il contesto in cui si è sviluppato il progetto, comprenderne le esigenze e l'evoluzione delle soluzioni adottate per il software. Ho dovuto analizzare un sistema realizzato da altri, ho dovuto comprendere i motivi delle scelte che erano state fatte e ho cercato di migliorare e arricchire il sistema mantenendo i principi su cui esso si basava. Secondo questi principi ho sfruttato le mie conoscenze tecniche per realizzare un prodotto completo e attento alle esigenze di un utente che, probabilmente, non possiede le mie stesse competenze tecniche.

Ho lavorato per la prima volta all'interno di un *team* di sviluppo e ho imparato a confrontarmi e collaborare con gli altri per raggiungere un obiettivo comune. Ho compreso quanto sia importante per la buona riuscita di un progetto di questo tipo creare un *team* con competenze eterogenee: nel mio caso ho avuto a disposizione il supporto offerto dal mio relatore, figura guida del progetto e con approfondite conoscenze nel campo della filologia, il supporto informatico offerto dal mio correlatore, che ha potuto confermarmi scelte che ho preso nel corso del progetto, e la possibilità di un continuo confronto con gli studenti del mio corso di studi che collaborano al progetto.

Sono entrata in contatto con un mondo, quello della filologia digitale, per me nuovo, ma ho anche approfondito le tecnologie a me più familiari, in particolare, ho

arricchito le mie conoscenze relative alla programmazione JavaScript e all'utilizzo delle trasformazioni XSLT.

Le difficoltà incontrate non sono state poche, oltre alla necessità di approfondire le mie conoscenze e di comprendere un lavoro già in corso di sviluppo, ho dovuto adattare il mio lavoro al software che era stato sviluppato; questo mi ha permesso, tuttavia, di arricchire il lavoro da cui sono partita: l'evidenziazione del collegamento testo-immagine contestuale con lo zoom delle righe del manoscritto, ad esempio, è un chiaro miglioramento rispetto al collegamento originale sviluppato dall'autore di IMT.

Visto il carattere sperimentale del progetto, e visti i pochi progetti con cui confrontarsi (ci sono alcuni progetti simili, ma di questi qualcuno sembra non essere più sviluppato[43], altri sono ancora in una fase iniziale di sviluppo[44]), è necessario ricordare che tutto il lavoro è aperto a successivi interventi di modifica e miglioramento.

[43] Come il progetto TEIViewer di Stephanie Schlitz, per maggiorni informazioni si veda: http://teiviewer.org/about/.
[44] Come il progetto TEI Boilerplate, per maggiorni informazioni si veda: http://dcl.slis.indiana.edu/teibp/.

Bibliografia

Bampi, Massimiliano. “Nuove prospettive d’impiego delle tecnologie informatiche in ambito filologico” in *Linguistica e filologia*, 16, 2003, pp. 147-157.

Buzzoni, Marina. “Le edizioni elettroniche dei testi medievali fra tradizione e innovazione: applicazioni teoriche ed empiriche all'ambito germanico” in *Annali di ca' foscari*, vol. XLIV, 2005, pp. 41-58.

Carlquist, Jonas. “Medieval Manuscripts, Hypertext and Reading. Visions of Digital Editions”.in *Literary and Linguistic Computing*, 19, 2004, pp. 105-118.

Driscoll, Matthew James. “The words on the page: Thoughts on philology, old and new” in *Creating the medieval saga: Versions, variability, and editorial interpretations of Old Norse saga literature*. Odense, Syddansk Universitetsforlag, 2010, pp. 85-102.

Ferrarini, Edoardo. “La trascrizione dei testimoni manoscritti: metodi di filologia computazionale” in *Digital Philology and Medieval Texts*. Pisa, Pacini editore, 2007, pp. 103-120.

Haugen , Odd Einar. “Parallel Views: Multi-level Encoding of Medieval Nordic Primary Sources” in *Literary and Linguistic Computing*, 19, 2004, pp. 73-91.

Kay, Michael. *XSLT 2.0 and XPath 2.0 4th Edition Programmer’s Reference*. Wiley Publishing, Indianapolis, 2008.

Pierazzo, Elena. “A rationale of digital documentary editions” in *Literary and Linguistic Computing*, 26, 2011, pp. 463-477.

Rosselli Del Turco, Roberto. “La digitalizzazione di testi letterari di area germanica: problemi e proposte” in *Digital Philology and Medieval Texts*. Pisa, Pacini editore, 2007, pp. 187-213.

Sitografia

Tutti i siti sono stati visualizzati l'ultima volta nel mese di Gennaio 2013.

- Biblioteca e Archivio Capitolare di Vercelli: http://www.tesorodelduomovc.it/archivio/index.html.
- HTML, CSS, XSL, JavaScript references: http://www.w3schools.com.
- Image Markup Tool: http://tapor.uvic.ca/~mholmes/image_markup/index.php.
- jQuery plugin iViewer: https://github.com/can3p/iviewer/wiki.
- jQuery plugin jqzoom Evolution: http://www.mind-projects.it/projects/jqzoom/.
- Libreria JavaScript “jQuery”: http://jquery.com.
- Licenza BSD: http://opensource.org/licenses/bsd-license.php.
- Milestone XSLT Wiki: http://wiki.tei-c.org/index.php/MilestoneXSLT.
- Progetto Junicode: http://junicode.sourceforge.net/.
- Progetto Vercelli Book digitale: http://islp.di.unipi.it/bifrost/vbd.
- Progetto Boilerplate: http://dcl.slis.indiana.edu/teibp/.
- Progetto TEIViewer: http://teiviewer.org/about/.
- Standard W3C, World Wide Web Consortium: http://www.w3.org.
- TEI P5 Guidelines: http://www.tei-c.org/Guidelines/P5.

- TEI, Text Encoding Iniziative: http://www.tei-c.org/index.xml.