UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

TESI DI LAUREA TRIENNALE

EVT Builder: un sistema per la pubblicazione di edizioni digitali

Relatore

Prof. Roberto Rosselli Del Turco

Candidato

Raffaele Masotti

Correlatore

Prof. Vittore Casarosa

Anno Accademico 2011/2012

Indice generale

1 – Introduzione

Il presente lavoro nasce con l'idea di sviluppare un sistema per semplificare la pubblicazione di edizioni digitali, inserendosi nel contesto più ampio del progetto *Vercelli Book Digitale.* Quest'ultimo è nato con il proposito di produrre una versione elettronica del *Codex Vercellensis*, al fine di rendere accessibile il manoscritto a un ampio pubblico.

Il rapporto di collaborazione con questo progetto inizia durante l'attività di tirocinio, in cui sono stati effettuati degli interventi su un software preesistente per la gestione e la visualizzazione di edizioni digitali. Gli interventi effettuati hanno riguardato principalmente l'introduzione di alcune caratteristiche mancanti e una riorganizzazione dei file del progetto, al fine di un successivo rilascio pubblico del software. Durante questa fase, lavorando per integrare nuove funzionalità, sono emerse alcune problematiche che hanno reso necessaria una profonda analisi dell'impostazione generale del sistema.

In questa relazione viene tracciato un quadro del percorso affrontato, partendo da una valutazione dei formati e delle tecnologie impiegate nel progetto, fino ad arrivare alla descrizione formale delle nuove scelte adottate.

Nel secondo capitolo viene introdotto il manoscritto che è stato al centro del lavoro svolto, illustrandone le caratteristiche principali e motivando la scelta di un'edizione digitale.
Successivamente, nel terzo capitolo, viene disegnata una panoramica dell'attività progettuale svolta, partendo da un breve *excursus* sui lavori precedenti, fino a una rassegna dei formati utilizzati. Grande attenzione è stata data anche all'analisi concettuale dei dati da trattare, affrontando i problemi legati alla loro visualizzazione.
Nel quarto capitolo viene descritto nel dettaglio lo sviluppo, le tecnologie usate, l'impostazione del progetto e la struttura dei diversi moduli del sistema, per arrivare in seguito alla descrizione dell'applicazione web prodotta nello stadio finale.
Infine, nel quinto capitolo, vengono riassunti i risultati raggiunti e le riflessioni personali sul progetto.

2 – Il Codex Vercellensis

2.1 – Il manoscritto

Il *Codex Vercellensis*, comunemente noto come *Vercelli Book*, è un codice pergamenaceo del tardo X secolo conservato presso l'Archivio e Biblioteca Capitolare[1] di Vercelli. Il suo arrivo a Vercelli si attesta sulla fine dell'XI secolo – inizio del XII, ed è probabilmente connesso all'intreccio di strade e percorsi intrapresi dai pellegrini in cammino verso Roma. La sua storia è conosciuta solo a grandi linee, i canonici italiani non erano in grado di comprendere la lingua in cui era stato redatto, quindi per moltissimi anni fu custodito senza che l'importanza del suo contenuto fosse conosciuta. Soltanto nel 1822, grazie allo studioso Friedrich Blume, fu riconosciuto come un manoscritto redatto in inglese antico, oggi considerato uno dei quattro codici più importanti del suo periodo per quanto riguarda la trasmissione dei testi poetici in lingua anglosassone. Gli altri tre sono: l'*Exeter Book*[2], il ms *Junius*[3] e il ms *Cotton Vitellius A. XV*[4].

Scritto da una sola mano in minuscola quadrata anglosassone[5], contiene 23 omelie in prosa e 6 componimenti poetici essenziali per lo studio della poesia anglosassone. Una collezione messa insieme sulla base di una grande varietà di testi omiletici o salvifici di autori e periodi diversi, selezionati per produrre una miscellanea che fosse ideale per nutrire il lato spirituale del lettore. Lo scriba che si occupò di redigere il manoscritto trascrisse i 136 fogli originari operando in modo perlopiù meccanico, riproducendo spesso anche parole prive di senso.

Delle 23 omelie che custodisce, 11 ci sono pervenute tramite il *Vercelli Book* come testimone unico. I 6 componimenti poetici non hanno un titolo, ma per convenzione vengono comunemente indicati come segue:

1 "Biblioteca e archivio Capitolare di Vercelli": http://www.tesorodelduomovc.it
2 Conservato presso la biblioteca della cattedrale di Exeter.
3 Conservato presso la Bodleian Library di Oxford.
4 Conservato presso il British Museum di Londra.
5 Cfr. Luiselli Fadda, *Tradizioni manoscritte e critica del testo nel Medioevo germanico*, Edizioni Laterza, 1999, pp. 49

- *Andreas* (ff. 29v-52v): narra la missione dell'apostolo Andrea nella terra dei Mermedoni antropofagi;
- *I Fati degli Apostoli* (ff. 52v-54r): martirologio dei 12 apostoli;
- *Discorso dell'anima al corpo* (ff. 101v-103v): un frammento di una poesia sul corpo e l'anima;
- *Bi manna lease* (ff. 104r-104v): un frammento poetico sulla falsità degli uomini;
- *Il Sogno della Croce* (ff. 104v-106r): una visione in cui al poeta appare la croce che racconta in prima persona la crocifissione di Gesù;
- *Elena* (ff. 121r-133v): un poema che narra il ritrovamento della croce da parte dell'imperatrice Elena, madre di Costantino.

2.2 – L'edizione digitale

Negli ultimi anni l'incontro fra filologia e informatica ha reso possibile lo sviluppo di numerosi strumenti utili per la diffusione e l'analisi dei documenti. Uno di questi strumenti è l'edizione digitale, ovvero un'edizione critica o diplomatica di un testo fruibile in formato digitale. Generalmente un'edizione digitale è composta da tre componenti fondamentali: testo, immagini e software di visualizzazione, a cui possono aggiungersi strumenti avanzati di indagine linguistica.

Nel caso specifico del *Vercelli Book*, la produzione di un visualizzatore rappresenta l'ultimo passo di un lungo processo di digitalizzazione del manoscritto e di trascrizione dei testi. L'idea alla base del progetto è quella di rendere il manoscritto facilmente accessibile, potendo usufruire sia delle potenzialità di diffusione di Internet sia del supporto di strumenti avanzati per la sua consultazione.

Nel terzo capitolo verrà proposta una panoramica sulle tecnologie impiegate nel progetto, dei problemi legati alla visualizzazione di questo tipo di dati e delle scelte progettuali adottate.

3 – Il progetto

3.1 – Il punto di partenza

Prima di entrare nel vivo della descrizione dell'architettura del sistema, e di quelle che sono state le principali scelte metodologiche e pratiche, è utile chiarire le fasi cardine e lo stato del progetto al momento dell'inizio della collaborazione che ha portato alla realizzazione di questo lavoro.

Nel corso degli ultimi cinque anni, il software per la consultazione dell'edizione digitale del *Vercelli Book* ha subito diverse evoluzioni. Un primo prototipo fu sviluppato utilizzando il linguaggio di programmazione Java, ma in seguito il software venne riprogettato in JavaScript, prendendo il nome di "*EVT, Edition Visualization Technology*". Questo cambiamento coincise con la volontà di passare da un'impostazione concepita principalmente per una fruizione offline ad un approccio web oriented, in linea con le possibilità tecnologiche più recenti e con una capacità di diffusione maggiore. Nel corso del tempo, comunque, il progetto ha mantenuto inalterati obiettivi e principi iniziali, brevemente riassunti di seguito:

- permettere la visualizzazione dei fogli del manoscritto, fornendo una rosa di strumenti per un'indagine più accurata degli stessi;
- permettere il confronto fra le scansioni digitali delle pagine e la corrispondente trascrizione;
- permettere il confronto fra edizioni diverse del testo, al fine di garantire uno studio più approfondito delle sezioni prese in esame.

Al fine di conseguire tali obiettivi vennero tracciate una serie di linee guida che tutt'ora costituiscono le fondamenta del progetto. Fra queste, è opportuno citare la scelta di utilizzare formati e linguaggi non proprietari, sia per la durabilità del prodotto sia per la possibilità di ridistribuzione del software stesso.

Prima dell'attuale ristrutturazione del sistema (ampiamente trattata nel capitolo 4), il caricamento dei dati e la costruzione dell'interfaccia utente erano interamente affidati alla libreria JavaScript jQuery[6].

3.2 – Tecnologie alla base del progetto

Delle tecnologie prese in esame verrà data una descrizione marginale, più che altro volta a illustrare i motivi che hanno portato a determinate scelte per il progetto, oppure a inquadrare meglio alcuni aspetti del lavoro.

3.2.1 – TEI, Text Encoding Initiative

La scelta di uno schema di codifica è una pratica comune a chiunque si appresti a effettuare la trascrizione e marcatura di un testo (letterario o non letterario), o anche semplicemente a chi voglia rappresentare un testo con un alto livello di espressività.

La diffusione delle tecnologie informatiche ha determinato lo sviluppo di svariati schemi di codifica, relativi a diversi settori disciplinari. Nell'ambito delle scienze umane, uno dei progetti più rilevanti in questo campo è la TEI (*Text Encoding Initiative*), uno standard per la creazione e l'interscambio di documenti elettronici nato nella seconda metà degli anni ottanta. Obiettivo principale della TEI è quello di sviluppare e mantenere un insieme di schemi di codifica, accompagnati dalla relativa documentazione e da una serie di strumenti utili (come fogli di stile per vari tipi di conversione), che possano essere usati per la rappresentazione dell'informazione testuale e la gestione di dati umanistici in formato elettronico. È fondamentale sottolineare come l'importanza di uno standard libero si rifletta nell'indipendenza da una determinata struttura hardware e applicazione software.

Le prime specifiche di tale standard furono pubblicate nel 1991 con il titolo *Guidelines for Electronic Text and Interchange, TEI P1*[7]. Fin dall'inizio venne

6 Sito web della libreria JavaScript jQuery: http://jquery.com
7 Le specifiche dello standard TEI: http://www.tei-c.org/Vault/Vault-GL.html

adottata una codifica di tipo dichiarativo[8] utilizzando lo *Standard Generalized Markup Language*[9] (SGML) per il markup dei documenti. Nel 2000 fu costituito il TEI Consortium, un'organizzazione internazionale senza scopo di lucro fondata al fine di sostenere e sviluppare ulteriormente lo schema di codifica TEI. Il primo risultato del TEI Consortium fu la pubblicazione nel 2002 di una nuova versione degli schemi di codifica (nota come P4), basata sull'*eXtensible Markup Language*[10] (XML). La versione corrente (TEI P5) è stata pubblicata alla fine del 2007. Quest'ultima versione introduce il supporto agli schemi RelaxNG[11] e prevede svariati miglioramenti, fra cui una maggior modularità e una personalizzazione semplificata.

Dunque, allo stato attuale, si potrebbe definire la TEI come una collezione di schemi di codifica XML (e di relativa documentazione) che permettono di effettuare la codifica di documenti di vario tipo. Sono circa 530 gli elementi di markup definiti dalla TEI, organizzati in forma modulare in modo da permettere, e anzi incoraggiare, la personalizzazione al fine di ottenere uno schema di codifica il più aderente possibile alle caratteristiche del testo da codificare.

L'adozione di un determinato schema di codifica comporta sempre pro e contro. Per quanto riguarda il caso specifico del *Vercelli Book Digitale*, la scelta di utilizzare lo standard TEI è risultata la decisione più opportuna, pur tenendo conto del fatto che non si tratta dell'unico standard in questo campo. Inoltre, va considerato che di base non prevede un sistema automatico per la pubblicazione del materiale.

3.2.2 – XML

XML: *eXtensible Markup Language*, ovvero linguaggio estensibile di marcatura. Più che un linguaggio, XML è in effetti un formalismo (una sintassi) che permette di utilizzare degli elementi per effettuare la marcatura del testo, al fine di fornire informazioni riguardo alla struttura dei dati. È stato concepito per per superare i limiti di *HyperText Markup Language* (HTML) e per semplificare l'alta complessità

8 I linguaggi di codifica dichiarativi identificano la funzione strutturale di interi blocchi testuali.
9 Documento del W3C che introduce SGML: http://www.w3.org/MarkUp/SGML
10 Documento del W3C che introduce XML: http://www.w3.org/XML
11 Sito ufficiale del linguaggio schema RelaxNG: http://relaxng.org

del suo diretto genitore, SGML.

Nella sintassi XML i dati vengono rappresentati tramite una struttura gerarchica a forma di albero (chiamata *albero XML*), avente un unica radice con un numero illimitato di nodi. Questi nodi possono appartenere a categorie differenti, dividendosi principalmente in:

- Nodi elemento: tipicamente composti da due *tag*, uno di apertura e uno di chiusura (es. <libro></libro>), oppure da un singolo *tag* nel caso di elementi vuoti (es. <pagina/>). Possono, a loro volta, contenere un numero arbitrario di figli e costituiscono il componente principale per la marcatura del testo.
- Nodi attributo: identificatori dichiarati internamente ai nodi elemento, aventi funzione di metadati.
- Nodi di testo: corrispondenti a un frammento dell'informazione archiviata nel documento XML. Questi tipi di nodi non hanno figli, quindi costituiscono sempre le parti terminali dell'albero.

Di base XML non prevede un vocabolario predefinito, il che lo rende flessibile e adattabile a una grande quantità di situazioni. Pur concepito inizialmente come standard web (per la trasmissione di dati via Internet), la sua grande duttilità lo rende impiegabile in vari contesti: la definizione di strutture di documenti, lo scambio di informazioni fra diverse piattaforme, la definizione di formati di dati, fino alla rappresentazione di oggetti tridimensionali.

Va specificato che XML non è un linguaggio di programmazione, ma un insieme di regole per produrre dei file, in formato testo, in grado di rappresentare strutture di dati. Una diretta conseguenza dei file in formato testo è la possibilità di leggerne il contenuto pur non disponendo di un programma specifico. Inoltre, a fronte di una sintassi tendenzialmente semplice, i dati strutturati nei documenti possono rappresentare entità molto complesse.

Un documento XML si compone principalmente di due parti:

- un prologo: contenente dichiarazioni di intestazione che possono specificare la versione e il tipo di codifica di caratteri;
- un corpo: obbligatorio, che deve necessariamente contenere almeno un elemento.

La struttura XML è fortemente gerarchica, ciascun elemento rappresenta un componente logico del documento. Un file XML si definisce "ben formato" se obbedisce alle regole della sintassi nativa XML, ovvero se contiene almeno un elemento, se possiede un elemento radice che contiene tutti gli altri elementi e se tutti i tag sono correttamente nidificati. Si definisce invece "valido" se, oltre ad essere ben formato, è conforme alle regole contenute in una DTD o in un XML Schema. Queste regole definiscono quali elementi possono essere utilizzati nel documento, quali attributi può accettare ogni elemento e impongono dei vincoli sulle relazioni tra gli elementi stessi.

Per rendere fruibile un file XML in un browser è necessario passare per un processo di trasformazione. Fra i vari metodi per effettuare questa operazione, l'utilizzo di *eXtensible Stylesheet Language Transformations*[12] (XSLT) è sicuramente il metodo più diffuso.

3.2.3 – XSLT

XSLT: *eXstensible Stylesheet Language Transformations* è un linguaggio di trasformazione di un file XML in un altro documento. È possibile distinguere due tipi di trasformazione principali: da un file XML a un altro XML, oppure da un file XML a un altro formato.

È utile sottolineare che un file XSLT è esso stesso un documento XML ben formato.

12 Documento del W3C che introduce XSLT 1.0: http://www.w3.org/TR/xslt

La trasformazione è ottenuta applicando le regole di trasformazione contenute nel foglio di stile ad un albero XML sorgente per generarne uno derivato.

```
<xsl:template match="tei:p">
      <p>
            <xsl:apply-templates/>
      </p>
</xsl:template>
```

Le regole sono essenzialmente dei template in cui viene dichiarato un *pattern* (es. match="tei:p") che specifica le condizioni per effettuare il controllo di corrispondenza con un nodo presente nel file d'origine. L'elaborazione degli elementi dell'albero è relativa alla presenza o meno di una regola che ne espliciti il trattamento. Dunque, le generalità dell'albero di destinazione saranno stabilite a partire da una regola definita per il nodo radice e sulla base della struttura dichiarata internamente al template.

L'elemento radice in un documento di questo tipo è *xsl:stylesheet*, i cui elementi figli sono chiamati elementi *top-level*; i principali sono:

- *xsl:include* e *xsl:import*;
- *xsl:output*;
- *xsl:template*.

I primi due servono per combinare fogli di stile e creare strutture modulari. La differenza principale fra i due elementi sta nella priorità che viene attribuita ai template contenuti nei file richiamati. Mentre con *xsl:include* viene effettuata un'inclusione diretta dei template richiamati (dunque con una parità dal punto di vista semantico), con *xsl:import* il foglio di stile importato avrà una precedenza maggiore sulle istruzioni contenute nel documento che effettua il richiamo. Per chiarire meglio tali affermazioni, supponiamo di avere uno stylesheet A che importa uno stylesheet B; nel caso in cui entrambi i documenti avessero un template che riguarda uno stesso ambito di applicazione, quello contenuto in B avrebbe

precedenza operazionale su quello contenuto in A.

L'elemento *xsl:output* è necessario per stabilire il formato dell'albero di destinazione, sulla base di quelli supportati: XML, HTML, Text. I suoi attributi specificano: il metodo da utilizzare (relativo, appunto, al formato), la versione, l'indentazione (attiva o meno) e la codifica di caratteri da utilizzare.

Infine, *xsl:template* definisce, tramite regole, un modello riutilizzabile per il trattamento degli elementi. La sua struttura interna serve a modellare l'albero di destinazione.

Esistono due versione stabili del linguaggio: la versione 1.0 e la versione 2.0. Al fine di comprendere le scelte del presente lavoro, qui di seguito vengono elencati alcuni vantaggi del passaggio alla seconda versione:

- maggior robustezza;
- supporto alle espressioni regolari;
- possibilità di importare uno schema XML esterno;
- miglioramento delle strutture di raggruppamento;
- gestione multi-output.

Di XSLT esiste già una versione 3.0[13] che prevede una modalità di trasformazione streaming, in cui né il documento di origine né quello di destinazione vengono tenuti in memoria durante l'avanzamento del processo. Nel complesso, novità sicuramente interessanti, ma ancora troppo acerbe da poter essere applicate a un progetto stabile.

Per certi versi, esistono già delle tecniche per aggirare alcune limitazioni delle precedenti versioni. Ad esempio, utilizzando un mix di JavaScript e di opportuni processori[14] è possibile applicare trasformazioni XSLT in modalità client-side. Tuttavia, la trattazione di questi metodi va oltre le esigenze del lavoro corrente.

13 Documento del W3C che introduce XSLT 3.0: http://www.w3.org/TR/xslt-30
14 Saxon Client Edition 1.0: http://www.saxonica.com

3.2.4 – JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti. La fase di maggior diffusione di questo linguaggio, in ambito web, inizia parallelamente ad un progressivo abbandono delle applet Java, limitate dalle restrizioni di sicurezza. Comunque, a fronte di un nome simile, Java e JavaScript hanno poco in comune, a parte una sintassi vagamente analoga.

Oltre alla semplicità di utilizzo, sono diverse le caratteristiche che hanno contribuito alla sua diffusione:

- è orientato agli oggetti, quindi permette di dichiarare classi e istanziare oggetti definendo un costruttore;
- è un linguaggio dinamico, dunque permette di definire le proprietà di un oggetto e le sue funzioni durante l'esecuzione;
- permette di utilizzare variabili globali, il che semplifica alcune operazioni, ma può produrre strutture poco portabili;
- ha una gestione flessibile dei tipi delle variabili, ad esempio è possibile convertire agevolmente un numero in una stringa.

In relazione al progetto, sono tre gli aspetti che rendono l'utilizzo di JavaScript indispensabile: la modifica dinamica del documento, il caricamento di file esterni e la creazione di un'interfaccia interattiva (dalle interazioni utente fino alla creazione di strumenti avanzati).
Per soddisfare queste esigenze è possibile percorrere due strade: la creazione ex novo di tutte le funzioni necessarie o l'utilizzo di un framework che provveda a fornire una solida base su cui lavorare.

L'utilizzo di un framework comporta un numero rilevante di vantaggi, che vanno dall'assunto comunemente riassumibile in "inutile reinventare la ruota", fino alla possibilità di fare affidamento su una documentazione già esistente e su una

numerosa e vitale comunità di sviluppatori.
Fra le tante librerie disponibili per JavaScript, jQuery sembra essere la scelta più convincente. I principali motivi che la rendono appetibile sono i seguenti:

- è distribuita come software libero con licenza GNU GPL[15];
- è cross-browser, dunque si occupa di risolvere in partenza molti problemi di compatibilità nel passaggio da una piattaforma all'altra;
- sono disponibili migliaia di plug-in, utili alle esigenze più disparate;
- ha un codice sintetico, cioè permette di effettuare, con poche righe di codice, operazioni generalmente più complesse;
- ha un'ottima documentazione ed è supportata da una comunità molto attiva.

Successivamente saranno descritte nel dettaglio le soluzioni adottate e come si sia tentato di rendere il prodotto finale meno dipendente dalla libreria utilizzata.

3.3 – I dati da trattare

Una volta scelto lo schema di codifica più congruo alle proprie esigenze, è necessario analizzare il modo in cui i dati sono strutturati e valutare che tipo di problemi possono presentarsi nel passaggio alla loro visualizzazione.

3.3.1 – Dal sistema di codifica alla visualizzazione

Terminata la fase di codifica di un documento, si sarà in possesso di un file strutturato, suddiviso in diverse partizioni formali: capitoli, paragrafi, versi, ecc.
Nel caso del formato TEI, un documento sarà suddiviso principalmente in due parti:

- un'intestazione (elemento <teiHeader>), contenente i metadati relativi al

15 Specifica della licenza GNU General Public License: http://www.gnu.org/licenses/gpl-3.0.txt

documento elettronico;

- uno o più testi.

```
<TEI>
      <teiHeader> [intestazione] </teiHeader>
      <text>
            <body> [testo] </body>
      </text>
<TEI>
```

L'elemento *body* (interno a *text*), a sua volta, conterrà ulteriori elementi utili a definire dettagliatamente la struttura del testo.

Questo tipo di file, per quanto ricco di informazioni, non è pronto per essere visualizzato in un browser o altre piattaforme. Per passare dal sistema di codifica alla sua visualizzazione sarà necessario associare il documento XML a un foglio di stile CSS[16] o utilizzare le trasformazioni XSLT. Generalmente la scelta del metodo viene effettuata sulla base delle specifiche esigenze, in quanto CSS è un linguaggio puramente descrittivo[17], mentre il linguaggio XSLT permette di definire regole (anche complesse) che consentono di manipolare radicalmente la struttura del documento di origine.

Un file XSL per la conversione da XML a HTML può presentarsi, di base, in questa forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      version="2.0">
      <xsl:output indent="yes" method="html" encoding="UTF-8"
            doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
            doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"/>
      <xsl:template match="/">
            <xsl:result-document
                  method="html"
                  href="{$filePrefix}/index.html"
                  indent="yes">
```

16 Documento del W3C che indroduce CSS: http://www.w3.org/TR/CSS

17 CSS non permette di effettuare modifiche strutturali degli elementi, ma si limita a caratterizzarli tipograficamente.

```
                        {Struttura HTML}
            </xsl:result-document>
      </xsl:template>
</xsl:stylesheet>
```

Bisognerà poi tenere conto delle esigenze specifiche di visualizzazione, ad esempio per ottenere una navigazione suddivisa per pagine, piuttosto che per capitoli o paragrafi.

3.3.2 – Le gerarchie sovrapposte e la paginazione

La codifica dichiarativa consente di rappresentare strutture testuali a più livelli, permettendo di codificare la struttura editoriale, la struttura grammaticale, la struttura fisica di un documento ed altro ancora. Questo però si scontra spesso con alcuni limiti di rappresentazione, intrinsecamente inclusi nello stampo gerarchico.
Si pensi, ad esempio, nel processo di trascrizione di un testo, alla necessità di rappresentare sia la delimitazione dei capitoli sia delle pagine. In casi del genere, non è raro che si presenti la situazione seguente:

```
      <pagina n=”4”>
            <capitolo n=”1”>
            {testo}
      </pagina>
      <pagina n=”5”>
            {testo}
            </capitolo>
      </pagina>
```

Com’è possibile vedere, la necessità di marcare sia le pagine che i capitoli può portare ad una sovrapposizione degli elementi. È importante notare che XML impone una gerarchia di contenimento che genera un singolo albero, dunque non è possibile utilizzare gerarchie sovrapposte senza produrre documenti non “ben formati”.

La TEI propone diverse tecniche per affrontare questo problema, da utilizzare a

seconda dei casi specifici in cui ci si trova. Considerata la natura estesa del tema, in questa trattazione verrà affrontata solo la tecnica che riguarda direttamente il progetto, ovvero l'utilizzo di elementi di tipo *milestone*.
Mantenendo l'esempio precedente come guida alla comprensione del problema, il primo passo per inquadrare la natura della soluzione proposta è quello di identificare logicamente una gerarchia cui attribuire priorità sintattica, dunque primaria, ed una secondaria. Identificando le caratteristiche fisiche come secondarie rispetto alla struttura astratta del testo, riconosceremo gli elementi pagina come subordinati alla gerarchia primaria dei capitoli. Chiarito questo, è possibile esplicitare che la soluzione proposta da TEI consiste nell'utilizzo di elementi vuoti per marcare la posizione degli elementi secondari all'interno del testo. Questi elementi vuoti – dunque privi di contenuto – dividono il testo in segmenti, senza invalidare la struttura del documento XML.
L'esempio precedente si presenterebbe quindi in questo formato:

```
<pagina n="4" />
<capitolo n="1">
      {testo}
      <pagina n="5" />
      {testo}
</capitolo>
```

Va specificato che l'elemento TEI utilizzato per identificare il limite di pagina è il tag <pb/> (*page break*), in cui l'attributo @n identifica il numero di pagina corrispondente.

Sul fronte puramente rappresentativo, questo metodo consente di aggirare il problema delle gerarchie multiple, pur mantenendo il documento ben formato. Tuttavia, questa soluzione introduce un'insidia particolarmente spinosa: diventa complesso ricostruire la struttura originaria partendo da una conformazione di questo tipo, in quanto gli elementi in questione non vengono rappresentati come nodi dell'albero, né includono una relazione di inizio/fine. Inoltre, è possibile che tali elementi delimitatori siano presenti a diversi livelli della gerarchia, andando a complicare ulteriormente il problema.

Gli inconvenienti di questo tipo di codifica sono sufficientemente consistenti da scoraggiare la ricostruzione di determinate strutture (in fase di pubblicazione) e da giustificare la soluzione sperimentale adottata in questo lavoro[18].

3.4 – Scelte progettuali

Per la proposta di un nuovo sistema per la visualizzazione dell'edizione del Vercelli Book si è dovuto tener conto di diversi fattori, fra cui le esigenze a breve/lungo termine in rapporto alle risorse disponibili.

3.4.1 – Le possibili strade

Per un'opportuna gestione dei dati sono state valutate tre possibili strade. Di seguito verrà fornita una rapida panoramica dei pro e dei contro delle varie opzioni.

3.4.1.1 – Utilizzo di un database XML nativo

Una delle alternative più interessanti per la gestione di dati in XML è l'utilizzo di un software chiamato eXist-db[19], ovvero un database XML nativo. A differenza di un database relazionale, che gestisce dati archiviati in tabelle, un database XML nativo usa gli stessi documenti come unità fondamentale di archiviazione, sfruttando quindi la struttura dei file per estrapolare relazioni e dati.

Un approccio di questo tipo comporta un certo numero di vantaggi, fra cui la possibilità di effettuare vere e proprie query sui dati a disposizione. Inoltre, eXist-db gode di un crescente interesse da parte della comunità TEI.
Tuttavia, l'utilizzo di un sistema del genere comporta anche una serie di problematicità. Sulla base di eXist, e sfruttando opportunamente XQuery[20], è possibile costruire un'intera applicazione web. Questo però comporta un certo grado di dipendenza dalla libreria stessa, la cui continuità di sviluppo non è garantita. In

18 Cfr. capitolo 4, paragrafo 4.3.2.2
19 Sito ufficiale di eXist-db: http://exist-db.org/exist/index.xml
20 Documento W3C che introduce XQuery: http://www.w3.org/TR/xquery

aggiunta, bisogna considerare che questa impostazione comporta la creazione di una struttura lato server, dunque il prodotto finale risulterebbe difficilmente utilizzabile in modalità offline.

Per i motivi sopracitati, questa ipotetica strada è stata rimandata ad approfondimenti futuri.

3.4.1.2 – Visualizzazione diretta file XML

Un'altra interessante proposta è quella avanzata dall'Indiana University con il progetto TEI Boilerplate[21]. L'idea si basa sulla possibilità di visualizzare i file XML in formato TEI direttamente nel browser, senza effettuare trasformazioni server-side. Si tratta quindi di un sistema leggero per la pubblicazione online di documenti TEI, che non necessita di particolari competenze per essere utilizzato e non dipende da una struttura server.
Purtroppo, anche questa opzione si scontra con un numero rilevante di limitazioni. Prima fra tutte, il file di una trascrizione potrebbe avere dimensioni consistenti, quindi risultare inadatto alla diffusione via web in mancanza di un'opportuna architettura che si occupi di gestire un caricamento progressivo.
In secondo luogo, al momento il progetto TEI Boilerplate si basa su XSLT 1.0, effettuando minime trasformazioni in client-side per inserire il contenuto testuale in un guscio HTML. Questa è una scelta potenzialmente elegante, ma pone sia limiti tecnologici (XSLT 2.0 risulta ormai più robusto e flessibile), sia limiti di fruizione offline (alcuni browser non permettono di effettuare trasformazioni XSLT quando i file risiedono nel file system locale).

Anche questa strada risulta quindi inadatta alla correnti necessità del progetto.

3.4.1.3 – Trasformazione XSLT statica

Una delle soluzioni più classiche, per la pubblicazione di documenti XML in formato TEI, si basa sulla trasformazione statica XSLT. Percorrendo questa si mira a generare

21 Sito ufficiale di TEI Boilderplate: http://dcl.slis.indiana.edu/teibp

delle pagine HTML partendo dai documenti della codifica XML, sarà quindi necessario lavorare con i fogli di stile in modo tale da ottenere in output una struttura quanto più vicina alle proprie esigenze.

Utilizzando questa impostazione è possibile produrre l'intera struttura di un sito web, eventualmente potenziabile con l'utilizzo di JavaScript e personalizzabile graficamente con opportuni CSS. Alcuni progetti[22] forniscono un'ottima base alla conversione di documenti in formato TEI standard, gestendo la trasformazione di un buon numero di elementi. Ciò nonostante, per i problemi precedentemente esposti legati alla paginazione, generalmente si fa ricorso ad una trasformazione monolitica[23] dell'intero contenuto del documento di partenza; dunque risultando inadatta a specifiche esigenze, come quella di associazione di una singola pagina alla relativa scansione digitale. Questo comunque non è una limitazione del linguaggio, ma un semplice freno dovuto alla complessità di elaborazione di determinate strutture dati. Il progetto *Berliner Intellektuelle 1800-1830*[24], pur generando in output strutture ipertestuali semplici, propone interessanti soluzioni per il controllo dei page break, in parte riprese e approfondite in questo lavoro.

Ovviamente, sono presenti degli svantaggi specifici anche per questo approccio. Principalmente sintetizzabili nella necessità di utilizzare un editor XML o un altro software necessario che permetta di effettuare la trasformazione XSLT (ad esempio Oxygen[25]), e nella perdita delle informazioni che non vengono adeguatamente trattate in fase di trasformazione.

Nonostante i punti negativi, allo stato attuale questa strada si dimostra la più vantaggiosa per le esigenze del progetto. Nel capitolo 4 verranno esposte nel dettaglio le soluzioni adottate.

22 Il progetto tei2html di Jawalsh può essere considerato un buon esempio di trasformazione dal formato TEI a HTML: https://github.com/jawalsh/tei2html

23 Per trasformazione monolitica si intende la creazione di un nuovo documento, comprendente l'intero contenuto del documento di origine, senza alcuna forma di suddivisione.

24 Progetto *Berliner Intellektuelle 1800-1830*: https://sites.google.com/site/annebaillot/digitale-edition

25 Oxygen, XML Editor: http://www.oxygenxml.com

3.4.2 – Dipendenze e durabilità

I prodotti software sono sempre in corso di revisione. I progetti molto grandi, se ben strutturati, vengono costantemente aggiornati per includere nuove funzionalità ed adattarsi a diverse piattaforme. Principalmente, gli interventi su un prodotto possono riguardare la correzione di bug o il miglioramento dell'interfaccia per aumentare il grado di accessibilità del software stesso. In entrambi i casi i costi di intervento possono risultare esosi, soprattutto se il sistema fa uso di librerie andate in disuso o inadatte al mercato contemporaneo.

L'utilizzo di framework, e degli annessi plugin, può essere considerata una buona pratica, in quanto permette una programmazione di livello più alto e può idealmente risolvere – in partenza – alcuni problemi di compatibilità. Il loro utilizzo dovrebbe essere subordinato a reali esigenze sul fronte pratico, considerando che il loro impiego produce inevitabilmente una legame di dipendenza. Se un'applicazione web è interamente basata su un determinato framework, la sua longevità sarà

condizionata dalla durata temporale della libreria stessa.

La situazione è leggermente differente nel caso della scelta di una linguaggio. Nel corso del tempo è probabile che le preferenze generali si orientino verso altri standard, ma è meno frequente un totale abbandono in un arco di tempo ristretto.

3.4.3 – I destinatari del progetto

Nella riprogettazione del visualizzatore è stata data particolare attenzione alla valutazione dei destinatari del progetto, che risultano separati in due figure:

- il filologo digitale, che intende usare il software per pubblicare del materiale;
- l'utente finale, ovvero lo studioso che accede ai contenuti via web o supporto removibile.

Lo studio di questi due profili è essenziale per indirizzare le scelte progettuali e per fornire uno strumento conforme agli obiettivi.

Riguardo al primo profilo, è risultato accettabile richiedere competenze relative all'utilizzo di un editor XML, di conoscenza della codifica TEI e – opzionalmente – di XSLT per personalizzare specificatamente la visualizzazione degli elementi trasformati. Sarebbe invece meno opportuno rendere necessarie competenze in ambito web, in quanto generalmente non fanno parte del background di un profilo umanistico.

Mentre per l'utente finale, oltre alle buone pratiche di usabilità e accessibilità, si è tenuto conto sia della modalità di consumo del materiale pubblicato (online/offline), sia di una gestione ottimale dell'interfaccia, costruita in modo tale da massimizzare la visualizzazione del contenuto e da permettere una navigazione intuitiva del manoscritto.

3.4.4 – Fruizione online/offline

La scelta dei canali di distribuzione si riflette inevitabilmente sulla rosa di tecnologie impiegabili per lo sviluppo del progetto.
Se l'obiettivo, come nel caso corrente, è quello di distribuire l'applicazione non solo via web, ma anche offline (tramite supporti removibili come: CD/DVD, USB, SD, ecc.), diventa necessario accantonare soluzioni server-side a favore di un'impostazione completamente client-side. Da un certo punto di vista, questo limita alcune possibilità di implementazione (come lo sviluppo di strumenti di authoring), ma è compatibile con gli obiettivi principali del progetto.

Nel capitolo dedicato allo sviluppo si vedrà quali accorgimenti sono stati presi per garantire la diffusione sul maggior numero di piattaforme e per aggirare le restrizioni di alcuni navigarori relative all'accesso ai file locali.

4 – Lo sviluppo

4.1 – Fasi di sviluppo

In questa sezione verranno elencati i vari passaggi che hanno portato dall'analisi progettuale allo sviluppo del prodotto finale (*EVT Builder*).
È possibile schematizzare le diverse fasi nel seguente modo:

1. **Studio delle soluzioni precedenti**: fase iniziata durante l'attività di tirocinio, necessaria a valutare le due soluzioni web precedentemente proposte, al fine di non vanificare il lavoro già svolto.
2. **Studio di altri progetti dello stesso settore**: la conoscenza degli sviluppi raggiunti nello stesso ambito è un passo fondamentale per la giusta calibrazione degli obiettivi e la gestione delle risorse.
3. **Analisi requisiti del progetto**: descritta nel capitolo precedente, è opportuna per stabilire gli obiettivi del progetto in modo da orientare al meglio la fase di sviluppo.
4. **Scelta della tecnologia e dei linguaggi**: stabilita sulla base dell'analisi svolta (illustrata nel terzo capitolo).
5. **Progettazione del sistema, modalità di trasformazione e interfaccia**: necessaria per la definizione della struttura software che realizza le specifiche e che sarà implementata per mezzo di una trasformazione XSLT.
6. **Implementazione trasformazioni XSLT**: fase di sviluppo dei moduli di trasformazione XSLT.
7. **Implementazione applicazione web**: include sia la gestione dell'output generato dalla trasformazione, sia lo sviluppo delle interazioni utente.
8. **Produzione documentazione**: attività indispensabile per consentire future espansioni del sistema.

9. **Testing**: fase di verifica dei risultati ottenuti.

4.2 – Tecnologia e linguaggi usati

Sulla base delle motivazioni esposte nel capitolo 3, sono stati individuati i linguaggi più opportuni al trattamento dei formati utilizzati nel progetto. Viene qui fornita una panoramica sintetica sulla tecnologia adottata e sul suo ruolo.

Com'è già stato chiarito in precedenza, le trascrizioni del *Vercelli Book* sono codificate secondo lo **schema TEI**, dunque come file **XML**. L'idea alla base del lavoro è quella di operare su un file XML unico, contenente l'intera trascrizione del manoscritto. A partire da questo file, mediante l'applicazione di una serie di fogli di stile **XSLT**, vengono generati automaticamente una serie di file **HTML**, gerarchicamente divisi in sottocartelle. I file prodotti includono riferimenti a file esterni, in particolare: funzioni **JavaScript** e fogli di stile **CSS**. Le funzioni JavaScript, realizzate sulla base del framework **jQuery**, si occupano di gestire le interazioni dell'utente e di fornire opportuni strumenti per la visualizzazione e navigazione del manoscritto, mentre i fogli di stile CSS delineano il layout e permettono di associare alle pagine il font **Junicode**[26] per la visualizzazione dei caratteri medievali.
Si è tentanto di limitare il più possibile l'impiego di plugin, mentre per l'utilizzo di funzioni non altrimenti implementabili, ne è stato adoperato uno per lo zoom delle immagini (jQuery **iViewer**[27]) ed uno per la gestione degli hashtag nell'URL (jQuery **BackButton**[28]).
Nella configurazione corrente, è previsto che le immagini delle scansioni siano in formato **JPG** e che vengano posizionate all'interno della cartella *scans*.

Sul piano pratico, chi utilizza il sistema è dispensato dal rapporto diretto con gran parte di questi standard, ma deve solo occuparsi di effettuare la trasformazione XSLT con un editor XML o con altri software appropriati.

26 Junicode è un font Unicode sviluppato per i medievalisti.
27 Pagina ufficiale del plug-in jQuery iViewer: https://github.com/can3p/iviewer
28 Pagina ufficiale del plug-in jQuery BackButton: http://benalman.com/projects/jquery-bbq-plugin

4.3 – La trasformazione

EVT Builder si basa principalmente sulla trasformazione di un file XML (in formato TEI P5[29]) per mezzo dell'applicazione di una serie di fogli di stile XSLT. Attraverso questo processo viene generata l'intera struttura dell'applicazione web, gestita poi con JavaScript. Il sistema di trasformazione è articolato in forma modulare e prevede l'utilizzo di svariate funzioni. Nei seguenti paragrafi verrà illustrata l'architettura del sistema che rende possibile tale operazione.

4.3.1 – La struttura modulare

Sono diversi i motivi per preferire una struttura modulare ad un unico file. In primo luogo, una struttura simile semplifica la manutenibilità, potendo far affidamento su una suddivisione che intuitivamente rimanda alle parti di interesse. In secondo luogo, con questa impostazione è possibile raggruppare logicamente codice con diversa destinazione; ad esempio distinguendo i moduli relativi alla trasformazione degli elementi TEI da quelli necessari a generare l'HTML.

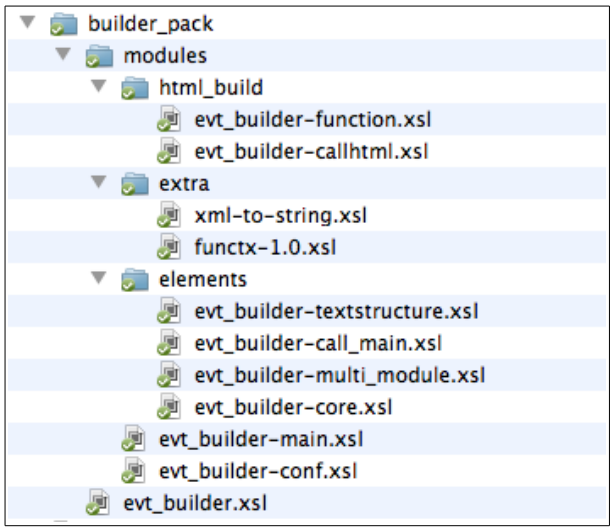

Fig 1: Gerarchia cartelle

L'inseme dei file XSL è articolato in una cartella principale chiamata *builder_pack* (v. fig. 1), contenente il file *evt_builder.xsl*[30], responsabile dell'inclusione di tutti i moduli, e la cartella *modules*. A sua volta, quest'ultima contiene tre cartelle e due

29 Linee guida TEI P5: http://www.tei-c.org/Guidelines/P5
30 Per il codice completo si veda in appendice: *evt_builder.xsl*

file, separati come segue:
Le cartelle:

- *html_build*, relativa alle trasformazioni che generano la struttura HTML;
- *extra*, contenente funzioni accessorie[31] incluse nel progetto per velocizzare operazioni standard (ad esempio l'elaborazione delle stringhe o la sostituzione dei caratteri);
- *elements*, dedicata ai moduli di trasformazione TEI.

I file:

- *evt_builder-main*[32], che si occupa di effettuare le chiamate principali per la generazione dell'applicazione web;
- *evt_builder-conf*[33], contenente una collezione di parametri e variabili configurabili per la personalizzazione dell'output.

Il *template* contenuto nel file *evt_builder-main* ha priorità massima rispetto a quelli contenuti in tutti gli altri moduli:

```
<xsl:template match="/">
```

Questa regola definisce la trasformazione per l'elemento *root* del documento di origine (il carattere *slash*, definito nell'esempio come valore dell'attributo *match*, indica appunto la radice del documento), andando a stabilire un modello di corrispondenza con l'intero albero. All'interno di questo template si ritrovano le chiamate verso i moduli HTML[34], ovvero sia la chiamata per la generazione della pagina principale sia le dichiarazioni riguardanti l'output delle singole pagine, divise per edizione.

31 Le funzioni contenute nella cartella *extra* non sono state sviluppate nel corso di questo lavoro.
32 Per il codice completo si veda in appendice: *evt_builder-main.xsl*
33 Per il codice completo si veda in appendice: *evt_builder-conf.xsl*
34 Cfr. capitolo 4, paragrafo 4.3.5

4.3.2 – I moduli

4.3.2.1 – Modulo HTML

Il modulo HTML è sostanzialmente un insieme di template che servono in parte a definire la struttura dei file HTML e in parte a definire frammenti di codice riutilizzabili.

I due template principali di questo modulo sono *index_build* e *data_structure*. A fine esplicativo verrà illustrato il secondo di questi (ovvero il più sintetico), in modo da chiarire la gestione dei rimandi e gli scopi del modulo in generale.

Il template *data_structure*, interno al file *evt_builder-callhtml.xsl* si presenta come segue:

```
<xsl:template name="data_structure">
      <xsl:param name="output"/>

      <html lang="en-US">
            <xsl:call-template name="html_head">
                  <xsl:with-param
                        name="html_path"
                        select="$dataPrefix"/>
                  <xsl:with-param
                        name="html_tc"
                        select="'datastructure'"/>
            </xsl:call-template>
            <body>
                  <section id="central_wrapper">

                        <div id="text_frame">

                              <div id="text">
                                    <xsl:call-template
                                          name="call_body">
                                          <xsl:with-param
                                                name="output"
                                                select="$output" />
                                    </xsl:call-template>
                              </div>
                        </div>
                  </section>
                  <footer>
                        <!--<p>2012@rafmas</p>-->
                  </footer>
            </body>
      </html>
</xsl:template>
```

Come prima cosa, notiamo che inizia con la dichiarazione del parametro “output”. Questo valore viene passato dalla chiamata contenuta nel file *evt_builder-main* e serve a distinguere la generazione delle pagine sulla base delle diverse edizioni del manoscritto (es. diplomatica, critica).
Il codice che segue rappresenta la struttura HTML delle singole pagine da generare. Internamente a questa sezione viene richiamato il template che definisce l’*head* della pagina, esplicitando – tramite i parametri – che la chiamata proviene dal template corrente e non da *index_build*. Mentre, a seguire troviamo il corpo della pagina con un ulteriore richiamo: *call_body*[35]. Quest’ultimo segna il ponte di collegamento fra la struttura HTML e la gestione della struttura TEI, infatti punta al file *evt_builder-call_main*, contenuto nella cartella *elements*; come verrà approfondito nel prossimo paragrafo, lo scopo di questo template è quello di popolare l’elemento *text* con il contenuto testuale di ogni pagina del manoscritto (una per ogni edizione).

Pur tralasciando un’illustrazione iterativa del template *index_build*, è utile focalizzare su uno dei template che internamente richiama, in quanto elemento fondamentale per la comprensione degli aspetti legati alla navigazione del manoscritto. Si tratta del template *pp_select_build*, responsabile della produzione dei form contenenti l’elenco delle pagine:

```
<xsl:template name="pp_select_build">
      <xsl:param name="html_select_main"/>
      <select class="main_pp_select">
             <xsl:if
                  test="$html_select_main='html_select_main'">
                  <xsl:attribute name="id">
                        <xsl:value-of
                              select="'control_pp_select'" />
                  </xsl:attribute>
             </xsl:if>
            <xsl:for-each select="//tei:pb">
                  <option>
                        <xsl:attribute name="value">
                              <xsl:value-of select="@n" />
                         </xsl:attribute>
                        <xsl:value-of select="@n"/>

                  </option>
            </xsl:for-each>
      </select>
</xsl:template>
```

35 Cfr. capitolo 4, paragrafo 4.3.2.2

Come spesso accade in una struttura simile, anche questo template è concepito per essere riutilizzato in punti diversi, quindi il parametro iniziale serve a distinguere la produzione del selettore principale da quelli secondari.
Nell'intreccio di XSLT e HTML è possibile distinguere la struttura di un tag select, popolato da una serie di opzioni i cui valori sono stabiliti grazie ad un'operazione ciclica effettuata sugli elementi <pb />, da cui viene estratto l'attributo @n.
L'elemento che processa questa selezione è <xsl:for-each>, il cui attributo "select" permette di specificare quali elementi devono essere selezionati. Volendo "tradurre" il codice in linguaggio naturale, il significato sarebbe: "per ogni elemento pagina, crea un'ulteriore opzione con valore e contenuto testuale corrispondenti al numero di pagina corrente". Il form generato verrà poi gestito da opportune funzioni JavaScript per spostarsi da una pagina all'altra del manoscritto.

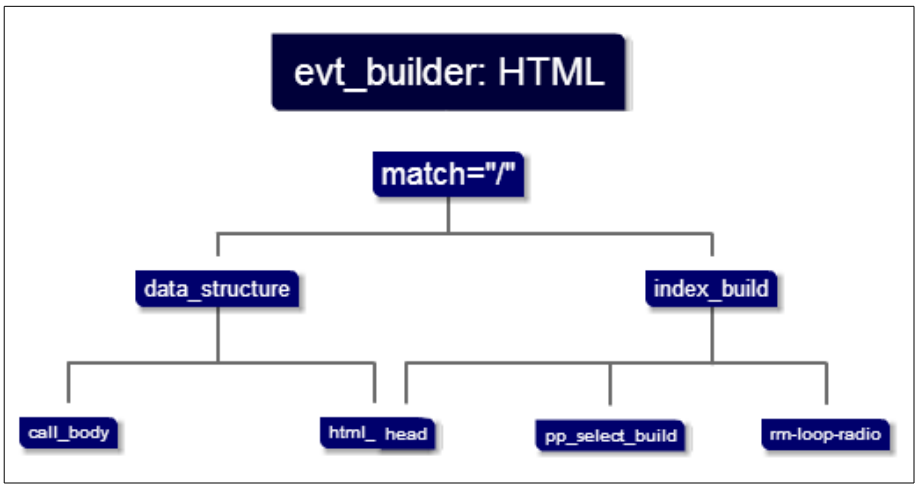

Fig 2: Template HTML

La generazione dell'HTML è dunque messa in atto da una ramificata serie di chiamate (v. fig. 2), ma è importante sottolineare che il codice HTML puro è presente solo all'interno del modulo relativo, mentre i rimandi negli altri file fungono da ponte all'interno della struttura.

4.3.2.2 – Modulo TEI

Per quanto concerne la generazione dell'HTML delle singole pagine del manoscritto, suddivise e organizzate sulla base delle diverse edizioni (diplomatica, critica, ecc.), è necessario precisare che le trasformazioni degli elementi TEI contenute in questo

lavoro hanno carattere puramente esemplificativo, in quanto una trattazione completa di tali aspetti esula dagli obiettivi prefissati. Nonostante questo, il modus operandi adottato per affrontare il problema della paginazione[36] può essere motivo di interesse, senza però dimenticare l'impronta tendenzialmente sperimentale del prodotto realizzato.

È opportuno iniziare l'analisi di questo modulo proprio dal collegamento con il modulo HTML. Come si è potuto vedere, nel template *data_structure* viene richiamato il template *call_body*, contenuto nel file *evt_builder-call_main* e presentato di seguito:

```
<xsl:template name="call_body">
      <xsl:param name="output"/>
      <xsl:apply-templates select="//tei:body">
            <xsl:with-param
                  name="pp_start" select="."/>
            <xsl:with-param
                  name="pp_end" select="following::tei:pb[1]"/>
            <xsl:with-param
                  name="output" select="$output"/>
      </xsl:apply-templates>
</xsl:template>
```

Questa regola darà ricorsivamente seguito a tutte le altre istruzioni del modulo TEI, in quanto dichiara di effettuare il parsing del contenuto dell'elemento body. Oltre al parametro di output, utile – come già detto – a identificare il tipo di edizione che si sta processando, è importante notare la presenza di altri due valori: pp_start e pp_end, che verranno utilizzati in gran parte delle dichiarazioni per identificare l'intervallo della pagina corrente.
Da questa parte in poi, i template per il match degli altri elementi contenuti nel body potranno usufruire di questi parametri per determinare la posizione durante la scansione del testo. A questo punto, per verificare l'intervallo fra un elemento pagina e l'altro, risulta fondamentale l'utilizzo della seguente formula[37]:

```
<xsl:if
      test="  not(following::tei:pb[@n=$pp_start/@n])
                  and
              not(preceding::tei:pb[@n=$pp_end/@n])">
```

36 Cfr. capitolo 3, paragrafo 3.3.2
37 Proposta da Solenne Coutagne, studente presso la parigina École des chartes.

Traducibile in linguaggio naturale come: “eseguire le seguenti istruzioni solo se non ci si trova prima o dopo un’interruzione di pagina”, dunque condizione verificata solo nel momento in cui il processore XSLT si trova nell’intervallo di una singola pagina.

Ovviamente quest’impostazione rende necessari alcuni prerequisiti, ad esempio non potrebbe essere applicata in mancanza degli elementi <pb/>. Pensando ad un visualizzatore standard, probabilmente l’ideale sarebbe rendere la paginazione opzionale, magari prevedendo che la suddivisione sia configurabile sulla base di un qualsiasi elemento. Tuttavia, se pur limitata, al momento risulta congrua con le necessità del progetto.

Per fissare infine la struttura dei template di questo modulo, si prenda come esempio quello relativo alla gestione dell’elemento TEI *corr*[38]:

```
<xsl:template match="//tei:corr">
      <xsl:param name="output"/>
      <xsl:param name="pp_start"/>
      <xsl:param name="pp_end"/>
      <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[2]">
                  <xsl:apply-templates>
                        <xsl:with-param
                              name="pp_start" select="$pp_start"/>
                        <xsl:with-param
                              name="pp_end" select="$pp_end"/>
                  </xsl:apply-templates>
            </xsl:if>
      </xsl:if>
</xsl:template>
```

Questa impostazione è generalmente valida per il trattamento di gran parte degli elementi. Inoltre, nel caso dell’esempio si vede come sia possibile restringere l’applicazione del template solo a un tipo di edizione[39].

Dei file contenuti in questo modulo (cartella *elements*) non viene data una

38 Elemento TEI contenente la forma corretta di un passaggio apparentemente erroneo.
39 Esemplificativamente si fa riferimento alla variabile $edition_array[2], cioè l’edizione “Critical” di questa demo.

descrizione dettagliata, in quanto le differenze sul piano pratico sono tendenzialmente irrilevanti: in questo caso, suddividere in più parti ha l'obiettivo di permettere un accesso rapido alle trasformazioni, raggruppando gli elementi in maniera analoga alla suddivisione dei moduli TEI (es. *evt_builder-core* per gli elementi del modulo TEI Core).

4.3.3 – La personalizzazione

Il sistema prevede la possibilità di personalizzare l'applicazione web mediante la configurazione di parametri e variabili, raggruppati nel file *evt_builder-conf*.

Le personalizzazioni più interessanti sono tre: una relativa alla personalizzazione dei percorsi, una alla presenza del frame immagine ed una riguardante le edizioni.

I parametri per la personalizzazione dei percorsi permettono di definire percorsi assoluti per il posizionamento dei file. Ad esempio, si potrebbe voler indicare che i file delle scansioni si trovano in un server specifico, differente da quello contenente l'applicazione web; per questa variazione sarà sufficiente anteporre l'indirizzo desiderato all'interno del parametro *filePrefix*.

Non è scontato che un utente intenzionato a pubblicare una trascrizione sia, da subito, in possesso delle relative scansioni digitali. In questi casi è utile il parametro *image_frame*, che è in realtà un semplice valore booleano[40]. Impostandolo su "false()", una serie di controlli si occuperanno di predisporre lo scheletro HTML in modo tale da poter visualizzare solo il testo della trascrizione.

Probabilmente la personalizzazione più interessante è quella relativa alla produzione di edizioni differenti. La variabile si presenta come segue:

```
<xsl:variable name="edition_array" as="element()*">
      <Item>Diplomatic</Item>
      <Item>Critical</Item>
</xsl:variable>
```

Il nome *edition_array* è del tutto arbitrario: non si tratta propriamente di un array,

40 I valori booleani prevedono due soli valori: *true* ('vero') e *false* ('falso').

quanto piuttosto di una 'variabile di elementi'. La funzione di questa variabile è duplice: prevede sia a definire i nomi delle edizioni sia a escludere del tutto una determinata produzione. È possibile impedire la generazione di pagine di una determinata edizione semplicemente rimuovendo il riferimento al tipo di edizione che si vuole escludere, ad esempio cancellando "Critical" da: <Item>Critical</Item>. Mentre, per aggiungere una nuova edizione bisognerà inserire un nuovo *item* e – necessariamente – la dichiarazione per i file di output nel file *evt_builder-main*. La generazione dei form destinati alla gestione delle edizioni nell'interfaccia utente viene attuata da un template richiamato ricorsivamente, che sarà oggetto del prossimo paragrafo.

4.3.4 – Le chiamate ricorsive

XSLT è un linguaggio molto potente, ma allo stesso tempo tendenzialmente rigido. Ad esempio, è possibile effettuare dei cicli con l'elemento <xsl:for-each>, ma non gestirli come se si trattasse di un linguaggio di programmazione convenzionale (ovvero definendo contatori o imponendo interruzioni non previste). Facendo un uso dichiaratamente creativo dei classici template, è possibile realizzare delle strutture non convenzionali in grado di soddisfare le esigenze meno comuni.

Di seguito verrà presentato il template *rm-loops_radio*, a cui è affidata la generazione dei form delle edizioni. Data la lunghezza, per comodità verrà spiegato in tre parti.

```
<xsl:template name="rm-loops_radio">
      <xsl:param name="rm_for" />
      <xsl:param name="rm_object" />
      <xsl:param name="rm_counter">1</xsl:param>
      <xsl:param name="rm_counter_test" />
      <xsl:variable name="radio_value"
            select="$rm_object[$rm_counter+0]" />
      <xsl:variable name="rm_counter_test_inc">
            <xsl:choose>
                  <xsl:when
                        test="$radio_value!=''">0</xsl:when>
                  <xsl:otherwise>1</xsl:otherwise>
            </xsl:choose>
      </xsl:variable>
(...)
```

In questa prima parte vengono definiti i parametri principali del template, che corrispondono idealmente ai parametri di un ciclo di un linguaggio di programmazione[41], indicando qual è la variabile da scandire, qual è la condizione di fine del ciclo e identificata la posizione del primo *item* pieno.

```
(...)
<xsl:if test="$rm_counter != $rm_for">
      <xsl:call-template name="rm-loops_radio">
            <xsl:with-param name="rm_for" select="$rm_for" />
            <xsl:with-param
                  name="rm_counter_test"
                  select="$rm_counter_test +
$rm_counter_test_inc" />
            <xsl:with-param name="rm_object" select="$rm_object" />
            <xsl:with-param
                  name="rm_counter"
                  select="$rm_counter + 1" />
      </xsl:call-template>
</xsl:if>
(...)
```

Finché il contatore non raggiunge il parametro stabilito per il termine del ciclo, il template continua a essere richiamato ricorsivamente, incrementando i contatori. Ad ogni chiamata, la funzione ripartirà dalla dichiarazione dei parametri, continuando a effettuare ulteriori chiamate fino al verificarsi della condizione.

```
(...)
<xsl:if test="$radio_value!=''">
      <xsl:choose>
            <xsl:when test="$rm_counter = $rm_counter_test">
                  <input
                        type="radio" name="edition_r"
                        class="regular-radio" checked="checked"
                        value="{$radio_value}" />
                  <xsl:value-of select="$radio_value" /> |
            </xsl:when>
            <xsl:otherwise>
                  <input
                        type="radio" name="edition_r"
                        class="regular-radio"
                         value="{$radio_value}" />
                  <xsl:value-of select="$radio_value" />
            </xsl:otherwise>
      </xsl:choose>
</xsl:if>
</xsl:template>
```

41 Nei linguaggi di programmazione, il ciclo determina l'esecuzione di una porzione di codice ripetuta *per un certo numero di volte.*

Infine la parte realmente operativa, in cui viene prodotta la sintassi HTML per ogni elemento contenuto nella variabile passata. In questo caso, l'elemento generato è un *radio*[42] che permetterà di muoversi da un'edizione all'altra.

Generalmente, le funzioni ricorsive comportano un peso significativo in fase di elaborazione, ma in questo caso il numero di ricorsioni è sufficientemente basso da permetterne un utilizzo tranquillo. Inoltre, questa impostazione permette di semplificare l'approccio dell'utente, che non dovrà curarsi di intervenire nei vari moduli per eventuali personalizzazioni.

4.3.5 – Output delle pagine

Per concludere la trattazione sul processo di trasformazione, è necessario tornare a parlare di uno dei file principali: *evt_builder-main*. A questo punto dovrebbero essere stati forniti tutti tasselli per poter comprendere come si articoli la generazione delle pagine, a partire dalle istruzioni contenute in questo file.

Al fine di produrre in output un numero variabile di documenti, partendo da un unico documento sorgente, è indispensabile un'istruzione introdotta in XSLT 2.0: *xsl:result-document*. Questo comando prevede diversi parametri[43] che consentono di personalizzare i documenti di destinazione, ma per questo lavoro è sufficiente dichiararne tre:

- *method="html"*, per selezionare il formato di destinazione;
- *href*, per stabilire l'indirizzo finale del file;
- *indent="yes"*, per aumentare la leggibilità del codice prodotto.

42 Pagina di riferimento del W3C per l'elemento HTML *radio*: http://www.w3.org/wiki/HTML/Elements/input/radio

43 Specifica W3C per i parametri dell'elemento xsl:result-document: http://www.w3.org/TR/xslt20/#element-result-document

Dunque, ricordando che ci si trova all'interno del template che seleziona il nodo radice (match="/"), viene proposta come esempio la dichiarazione necessaria ad avviare la trasformazione delle pagine di edizione "Critical":

```
<xsl:if test="$edition_array[2]!=''">
      <xsl:variable
            name="edition_current"
            select="lower-case($edition_array[2])" />
      <xsl:for-each select="//tei:pb">
            <xsl:result-document
                  method="html"
                  href="{$filePrefix}/data/
{$edition_current}/page_{@n}_{$edition_current}.html"
                  indent="yes">
                  <xsl:call-template name="data_structure">
                        <xsl:with-param
                              name="output"
                               select="$edition_array[2]"/>
                  </xsl:call-template>
            </xsl:result-document>
      </xsl:for-each>
</xsl:if>
```

L'istruzione viene eseguita solo nel caso in cui l'elemento dell'edizione selezionata esista. Se la verifica del controllo ha esito positivo, per ogni elemento <pb/> richiederà la generazione di un documento, posizionandolo infine nella cartella indicata. Il contenuto del documento sarà dipendente dall'insieme di regole approfondite nei paragrafi precedenti richiamate dal template *data_structure*.

4.4 – L'applicazione web

Il prodotto delle trasformazioni analizzate è a tutti gli effetti un'applicazione web, concepita in modo tale da poter girare sia su un server web che in locale. Il file principale di questa applicazione è la pagina *index.html*, la quale include i link a tutta una serie di ulteriori file e carica dinamicamente il testo delle pagine del manoscritto. I vari file sono organizzati gerarchicamente in diverse cartelle (v. fig. 3).

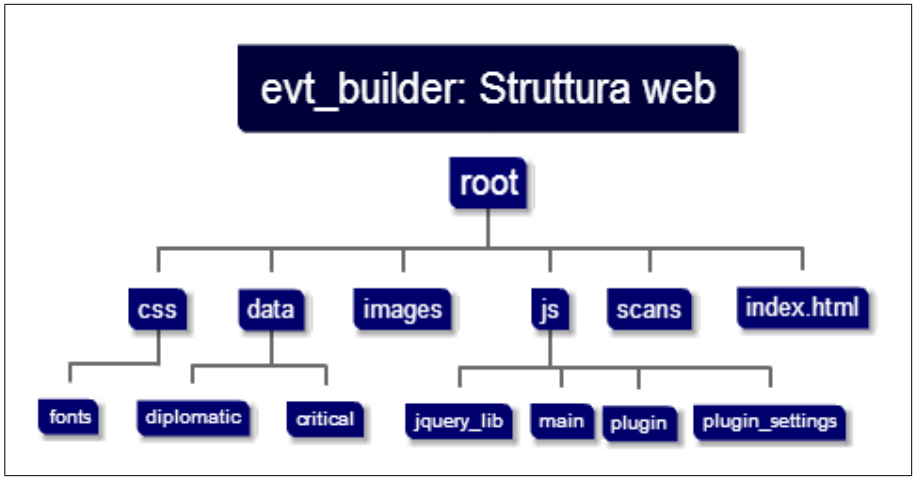

Fig 3: La struttura delle cartelle dell'applicazione web

4.4.1 – Gestione interna

Come precedentemente accennato, la gestione dell'applicazione web è affidata a una serie di funzioni JavaScript basate su jQuery. Al primo caricamento della pagina principale viene inizializzata la struttura di base, mentre successivamente vengono caricati dinamicamente i testi sulla base dell'input dell'utente.

4.4.1.1 – Il caricamento delle pagine

Il caricamento dei testi, e delle relative immagini, viene gestito tramite la lettura dell'hashtag[44] contenuto nell'indirizzo. Prendiamo come esempio questo ipotetico link:

*www.sitomanoscritto.it/index**#104v***.

Il sistema è realizzato in modo tale da permettere il collegamento diretto alle pagine di interesse, quindi digitando il precedente link si verrebbe reindirizzati direttamente alla pagina "104v". Questo avviene grazie ad un apposito controllo[45] che, al caricamento della pagina, verifica la presenza di un eventuale hashtag e richiama la funzione *gotopage()*. L'azione non determina solo il caricamento di testo e immagine, ma si riflette anche nel cambiamento di tutti i selettori e dell'etichetta che segna il numero di pagina corrente.

44 Gli hashtag sono delle parole precedute dal simbolo # e poste al termine degli indirizzi URL
45 Per il codice completo si veda in appendice: *interface_control.js*, funzione *hashchange*

È bene ricordare che, una volta terminata la trasformazione XSLT, tutti i percorsi sono automaticamente memorizzati nella struttura dell'applicazione web, quindi chi utilizza il sistema per pubblicare la propria trascrizione non dovrà configurare manualmente ogni singolo indirizzo.

Parallelamente al caricamento delle pagine, viene gestito anche il caricamento di una determinata edizione, sulla base di quelle prodotte dalla trasformazione. Una volta selezionata una determinata edizione, verrà tenuta traccia di questa scelta, in modo tale da visualizzare i testi delle pagine successive nell'edizione corrispondente.

4.4.1.2 – Le diverse modalità

È possibile utilizzare il visualizzatore in due distinte modalità: testo-immagine e testo-testo. Di default viene visualizzata la modalità testo-immagine, ma è sufficiente utilizzare i link presenti nella barra superiore per intervenire sulla visualizzazione.

La modalità testo-testo è progettata in modo da poter consultare parallelamente lo stesso testo in edizioni differenti. Anche in questo caso la navigazione terrà conto delle scelte effettuate dall'utente, dunque se venisse selezionata la modalità *Diplomatic* nel frame destro e *Critical* in quello sinistro, l'avanzamento delle pagine resterebbe costante rispetto a questa disposizione.

4.4.2 – Interfaccia

4.4.2.1 - Struttura

L'interfaccia grafica è stata progettata in modo da mantenere un aspetto minimale e da massimizzare la visualizzazione dei contenuti (v. fig. 4).

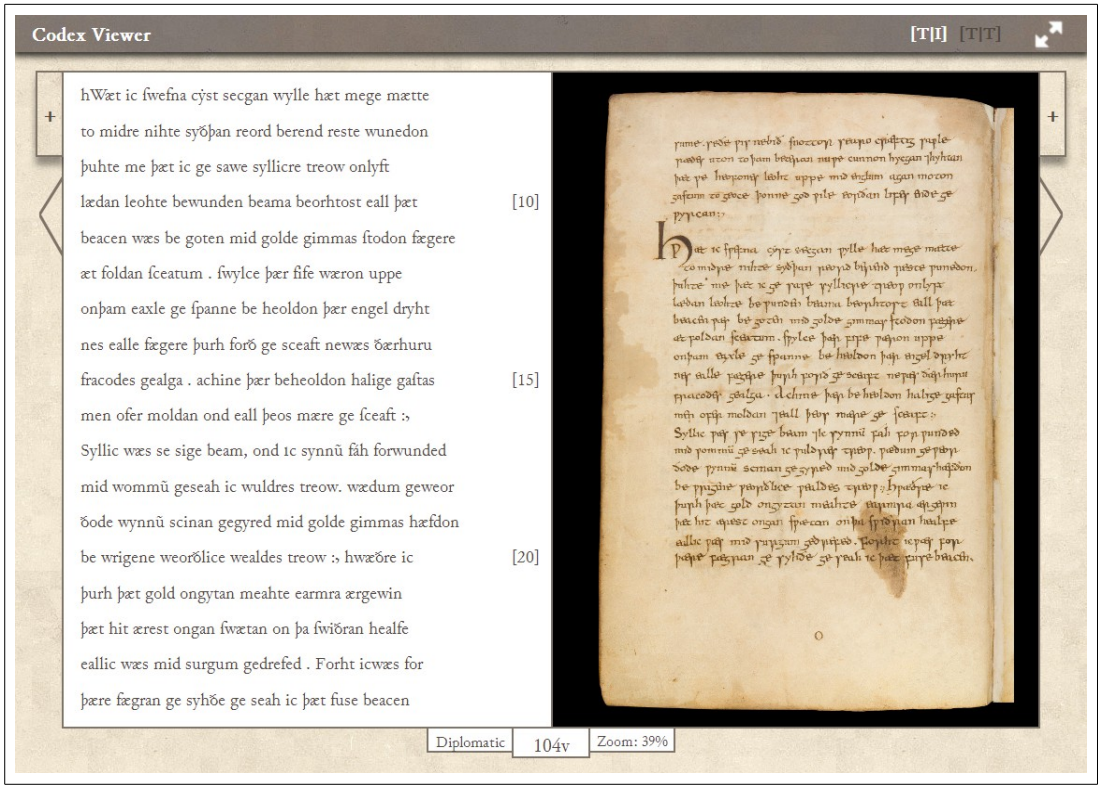

Fig 4: Interfaccia utente

Le parti sempre visibili si limitano a: una sottile barra superiore, due riquadri dei contenuti, i tasti di interazione e a delle etichette informative nella parte inferiore. I menu di default sono nascosti, ma facilmente raggiungibili con un singolo click.

L'interfaccia non si basa su un layout preimpostato di una libreria, ma è stata costruita ex novo per meglio adattarsi alle esigenze del progetto. La struttura è realizzata in modo tale da prevedere un comportamento tendenzialmente adattivo. In primo luogo non è possibile renderla inutilizzabile diminuendo le dimensioni della finestra, grazie ad una serie di regole che impongono delle dimensioni limite; in aggiunta, l'area occupata dai riquadri si adatta sempre al massimo spazio disponibile.

4.4.2.2 – Navigazione interna

È possibile navigare il manoscritto in due modalità distinte: mediante interazioni con il mouse o con tastiera.

Operando con il mouse si ha modo di spostarsi fra le pagine tramite diverse azioni (v. fig. 5):

- utilizzando le frecce, posizionate lungo il bordo dei riquadri;
- utilizzando il selettore rapido per raggiungere rapidamente la pagina di interesse;
- utilizzando la navigazione per miniature.

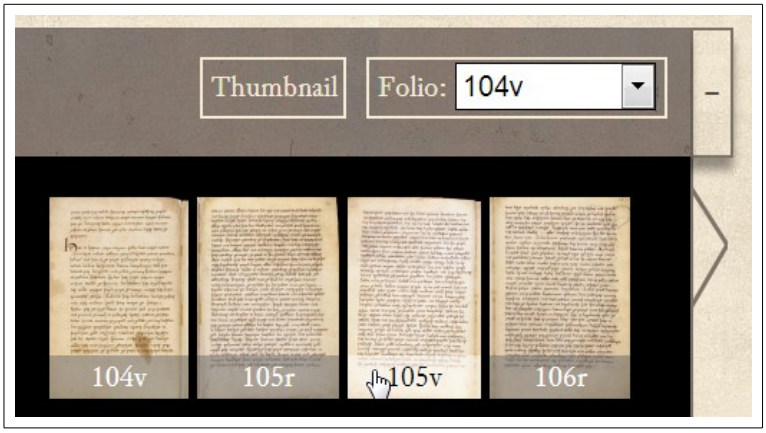

Fig 5: Interfaccia utente

Il passaggio con il mouse sugli elementi cliccabili produce un feedback visivo per l'utente.
Inoltre, nel momento in cui il mouse entra nel riquadro dell'immagine, sulla destra viene visualizzato un piccolo rettangolo contenente gli strumenti che permettono di interagire con lo zoom (v. fig. 6).

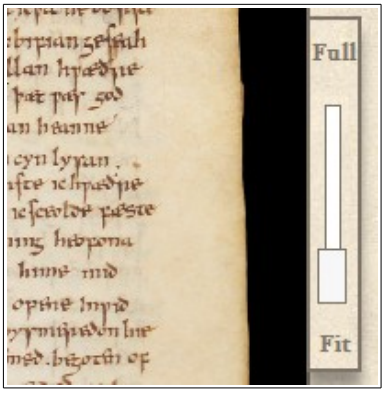

Fig 6: Toolbox zoom

La navigazione del manoscritto è controllabile anche da tastiera. Il gestore di eventi[46]

46 Per il codice completo si veda in appendice: *keydown.js*

permette di spostarsi avanti e indietro nelle pagine premendo i tasti direzionali, oppure di chiudere rapidamente i menu premendo il tasto *esc*.

4.4.3 – Criteri di usabilità e accessibilità

Nell'implementazione dell'applicazione web si è tentato di adottare tutti gli accorgimenti possibili per consentire all'utente un'esperienza di utilizzo lineare. Per la disposizione degli elementi e lo sviluppo delle interazioni, si è tenuto conto della fruizione del prodotto mediante piattaforme differenti e dispositivi multi-risoluzione.

4.4.3.1 – Compatibilità cross browser

I destinatari del prodotto potrebbero avere a disposizione mezzi molto differenti, sia per composizione hardware sia per architettura software, per cui l'applicazione è stata testata su diverse piattaforme e in diverse condizioni.
L'interfaccia è stata realizzata in modo da essere visualizzata correttamente con i browser associati ai sistemi operativi più diffusi (Windows, MacOSX, Linux). I test sono stati effettuati con:

- Chrome v.20;
- Firefox v.16;
- Safari v.5;
- Opera v.10;
- Internet explorer v.8.

4.4.3.2 – Limitazione offline

Nel corso dell'implementazione dei caricamenti dinamici delle pagine, è emersa una limitazione nell'accesso ai file locali da parte di alcuni browser (es. Chrome). Questa restrizione, applicata per motivi di sicurezza, potrebbe essere motivo di frustrazione da parte degli utenti che volessero utilizzare l'applicazione web tramite un supporto fisico offline, dunque si è cercato un modo di aggirarla.
La soluzione è stata fornita dall'utilizzo di iFrame[47] per gestire il caricamento dei testi. Inserendo nel riquadro testuale un iFrame, e poi aggiornando dinamicamente il suo indirizzo per mezzo di JavaScript, il browser non considera il l'operazione come un accesso vero e proprio ai file, dunque permette di variare il contenuto della parte testuale pur non effettuando un reale passaggio ad una pagina differente. Questa impostazione consente di evitare il caricamento dell'intera interfaccia ad ogni cambio pagina e permette una fruizione più snella del contenuto.

4.5 – Il sistema di documentazione

Poter usufruire di una documentazione è certamente un buon punto di partenza per l'implementazione di un software. Allo stesso tempo, mantenere commenti e documentazione contemporaneamente aggiornati richiede una certa quantità di impegno da parte di chi sviluppa il codice. Per questo si è scelto di integrare nel progetto uno strumento di documentazione automatica.

Lo strumento in questione si chiama XSLTdoc[48]. Esso definisce delle convenzioni per commentare il codice direttamente nel codice sorgente e fornisce un foglio di trasformazione XSLT per generare la documentazione in formato HTML.
I commenti vengono scritti in formato XML, definendo un namespace differente che permetta di distinguerli dal resto del codice.

L'ipertesto prodotto, nonostante l'aspetto spartano, è altamente navigabile, completo di indici e codice sorgente incorporato. In questo modo lo sforzo si riduce a tener

47 Specifica W3C dell'elemento iFrame: http://www.w3.org/TR/html4/present/frames.html#edef-IFRAME
48 Strumento di documentazione per XSLT: http://www.pnp-software.com/XSLTdoc/index.html

aggiornati i commenti, potendo generare man mano una documentazione aggiornata.

4.6 – L'utilizzo

Il risultato finale di questo progetto è una cartella contenente un insieme di file in diversi formati, organizzati in sottocartelle. Per sintetizzare il lavoro di sviluppo svolto, si può fornire una sintesi delle azioni che un ipotetico utilizzatore deve svolgere per passare dal proprio file XML all'applicazione web completa. I passaggi da effettuare sono principalmente tre:

1. Inserire nella cartella *scans* le immagini corrispondenti alle pagine, nel formato: pagina.jpg (es. 104v.jpg);
2. Copiare il proprio file XML della trascrizione nella cartella *builder_pack*, più gli eventuali file necessari (es. schema di codifica);
3. Effettuare la trasformazione del file XML con il foglio di stile *evt_builder.xsl*.

Come precedentemente descritto, è anche possibile configurare una serie di parametri per personalizzare il risultato finale.
Al termine del processo, nella cartella principale verrà creata sia la *index.html* sia una cartella contenente le diverse pagine (divise per edizione).

L'applicazione web generata è predisposta sia al caricamento su un server web sia all'utilizzo in locale (quindi a essere distribuita tramite CD o altri supporti removibili offline). L'utente finale, accedendo alla pagina principale, potrà consultare il manoscritto in diverse modalità e con l'ausilio di strumenti appositamente concepiti (ad esempio lo zoom delle scansioni). Le modalità a disposizione (testo-immagine e testo-testo) consentono di visualizzare sia la trascrizione delle singole pagine, affiancata alla relativa scansione del manoscritto, sia di confrontare più edizioni dello stesso testo contemporaneamente. Inoltre, il gestore dell'interfaccia tiene conto delle scelte dell'utente, consentendogli di navigare le pagine a disposizione mantenendo la modalità precedentemente selezionata.

5 – Conclusioni

Il lavoro fin qui descritto è imperniato sulla costruzione di un ponte fra rappresentazione e visualizzazione dei dati, tracciando le basi di un sistema che possa prestarsi facilmente a personalizzazioni ed espansioni future.

Per prefigurare il punto di arrivo sono stati necessari numerosi confronti, l'approfondimento di diverse tecnologie e una profonda analisi degli scenari di applicazione del sistema stesso. In questo processo si è tenuto conto non solo della componente tecnologica del sistema, ma anche del profilo dei destinatari di questo lavoro. I risultati emersi vanno a costituire le basi di uno strumento che permetta di pubblicare l'edizione digitale di un testo, senza la necessità di approfondire direttamente le tecnologie che costituiscono l'applicazione finale.

Il risultato raggiunto apre sicuramente nuove problematiche da affrontare, ma nello stesso tempo costituisce un punto di partenza incoraggiante verso la creazione di un sistema sempre più slegato dal contesto specifico per cui è stato realizzato. La scelta di utilizzare standard aperti e librerie *open source* va di pari passo alla volontà di distribuire liberamente il codice prodotto, nella speranza di contribuire allo sviluppo e alla trasmissione della conoscenza.

Nel campo dell'informatica umanistica si percepiscono segnali deboli di un futuro in lento avvicinamento. Lentezza dovuta in parte alla scarsità di risorse disponibili e in parte a una difficile conciliazione di settori apparentemente distanti. L'incontro fra scienze umane e scienze esatte spesso risulta difficoltoso e la figura dell'informatico umanistico può essere l'anello mancante fra competenze fortemente specializzate in entrambi i settori, affermandosi come mediatore in grado di stabilire un dialogo fra le due parti. L'evoluzione culturale dovrebbe tenere conto degli sviluppi tecnologici e ricercare la giusta chiava per adattarsi a un nuovo contesto, quale quello di una nuova società interconnessa, che permette di annullare tempi e spazi per la fruizione dei contenuti.

Bibliografia

- Fadda, Luiselli. *Tradizioni manoscritte e critica del testo nel Medioevo germanico*. Roma-Bari, Editori Laterza, 1999.
- Pierazzo, Elena. *La codifica dei testi*. Roma, Carocci Editore, 2005.
- Stussi, Alfredo. *Fondamenti di critica testuale*. Bologna, Il Mulino, 2006.
- Fawcett Joe, Zakas Nicholas C., McPeak Jeremy. *Ajax: guida per lo sviluppatore*. Hoepli, 2006.
- Niederst, Jennifer. *Web Design in a nutshell: a desktop quick reference, 3rd Edition*. Sebastopol, O'Reilly, 2006.
- Gardner John Robert, Rendon Zarella. *XSLT and XPATH: A Guide to XML Transformations*. Prentice Hall PTR, 2001.

Sitografia

Tutti i siti sono stati visualizzati l'ultima volta nel mese di Novembre 2012.

- Progetto “Vercelli Book digitale”: http://islp.di.unipi.it/bifrost/vbd
- Biblioteca e Archivio Capitolare di Vercelli: http://www.tesorodelduomovc.it
- TEI, Text Encoding Iniziative: http://www.tei-c.org/index.xml
- TEI: P5 Guidelines: http://www.tei-c.org/Guidelines/P5
- Digital Humanities Oxford Summer School: http://tei.oucs.ox.ac.uk/Talks/2010-07-oxford
- W3C, World Wide Web Consortium: http://www.w3.org
- CSS references: http://www.w3schools.com/cssref/default.asp
- Libreria JavaScript “jQuery”: http://jquery.com
- jQuery user interface: http://jqueryui.com
- jQuery plug-in iViewer: https://github.com/can3p/iviewer
- jQuery plug-in Back Button: http://benalman.com/projects/jquery-bbq-plugin
- XSLTdoc – A code documentation Tool for XSLT: http://www.pnp-software.com/XSLTdoc/index.html
- Oxygen XML Editor: http://www.oxygenxml.com
- Progetto tei2html Jawalsh: https://github.com/jawalsh/tei2html
- Progetto Digitale Edition "Das intellektuelle Berlin um 1800": https://sites.google.com/site/annebaillot/digitale-edition

Appendice

File “evt_builder.xsl”

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xpath-default-namespace="http://www.tei-c.org/ns/1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:eg="http://www.tei-c.org/ns/Examples"
    xmlns:xd="http://www.pnp-software.com/XSLTdoc"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:tei="http://www.tei-c.org/ns/1.0"
    xmlns="http://www.w3.org/1999/xhtml"
    exclude-result-prefixes="#all">
      <xd:doc type="stylesheet">
            <xd:author>RafMas</xd:author>
            <xd:short>
                  IT: Serie di include
            </xd:short>
              <xd:detail>
                  IT: Questo è il file principale per effettuare la
trasformazione, serve a includere tutti gli altri moduli evt_builder-*.xsl

              </xd:detail>
      </xd:doc>
      <!-- Basic -->
      <xsl:include href="modules/evt_builder-conf.xsl"/>
      <xsl:include href="modules/evt_builder-main.xsl"/>
      <xsl:include href="modules/html_build/evt_builder-function.xsl"/>
      <xsl:include href="modules/html_build/evt_builder-callhtml.xsl"/>
      <!-- Elements -->
      <xsl:include href="modules/elements/evt_builder-core.xsl"/>
      <xsl:include href="modules/elements/evt_builder-textstructure.xsl"/>
      <xsl:include href="modules/elements/evt_builder-multi_module.xsl"/>
      <xsl:include href="modules/elements/evt_builder-call_main.xsl"/>
      <!-- Extra -->
      <xsl:include href="modules/extra/functx-1.0.xsl"/>
</xsl:stylesheet>
```

File “evt_builder-main.xsl”

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xpath-default-namespace="http://www.tei-c.org/ns/1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:eg="http://www.tei-c.org/ns/Examples"
    xmlns:xd="http://www.pnp-software.com/XSLTdoc"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:tei="http://www.tei-c.org/ns/1.0"
    xmlns="http://www.w3.org/1999/xhtml"
    exclude-result-prefixes="#all">
      <xd:doc type="stylesheet">
            <xd:short>
                  IT: Le chiamate principali del sistema
            </xd:short>
      </xd:doc>
      <!-- All -->
```

```
    <xd:doc>
        <xd:short>IT: Chiamata principale del sistema</xd:short>
        <xd:detail>
                    IT: La prima parte produce in output i file HTML delle
diverse edizioni, mentre la secoda richiama il template generale per il
build della index.
             </xd:detail>
    </xd:doc>
      <xsl:output indent="yes" method="html" encoding="UTF-8"
             doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
             doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"/>

      <xsl:template match="/" priority="1">
             <xsl:if test="$edition_array[1]!=''">
                    <xsl:variable name="edition_current" select="lower-
case($edition_array[1])" />
                    <xsl:for-each select="//tei:pb">
                           <xsl:result-document method="html"
href="{$filePrefix}/data/
{$edition_current}/page_{@n}_{$edition_current}.html" indent="yes">
                                  <xsl:call-template name="data_structure">
                                         <xsl:with-param name="output"
select="$edition_array[1]"/>
                                  </xsl:call-template>
                           </xsl:result-document>
                    </xsl:for-each>
             </xsl:if>
             <xsl:if test="$edition_array[2]!=''">
                    <xsl:variable name="edition_current" select="lower-
case($edition_array[2])" />
                    <xsl:for-each select="//tei:pb">
                           <xsl:result-document method="html"
href="{$filePrefix}/data/
{$edition_current}/page_{@n}_{$edition_current}.html" indent="yes">
                                  <xsl:call-template name="data_structure">
                                         <xsl:with-param name="output"
select="$edition_array[2]"/>
                                  </xsl:call-template>
                           </xsl:result-document>
                    </xsl:for-each>
             </xsl:if>
             <xsl:result-document method="html"
href="{$filePrefix}/index.html" indent="yes">
                    <xsl:call-template name="index_build" />
             </xsl:result-document>
    </xsl:template>
</xsl:stylesheet>
```

File “evt_builder-conf.xsl”

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xpath-default-namespace="http://www.tei-c.org/ns/1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:eg="http://www.tei-c.org/ns/Examples"
    xmlns:xd="http://www.pnp-software.com/XSLTdoc"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:tei="http://www.tei-c.org/ns/1.0"
    xmlns="http://www.w3.org/1999/xhtml"
```

```
    exclude-result-prefixes="#all">
      <xd:doc type="stylesheet">
            <xd:short>
                  IT: Questo file è una collezione di parametri e
variabili configurabili, usati negli altri moduli.
            </xd:short>
      </xd:doc>
      <!-- GLOBAL -->
      <xsl:param name="mainPrefix" select="'.'"/> <!-- index -->
      <xsl:param name="filePrefix" select="'..'"/> <!-- image -->
      <xsl:param name="dataPrefix" select="'../..'"/> <!-- page -->
      <!-- Index title -->
      <xsl:param name="index_title" select="'Codex Viewer'"/>
      <!-- Hide/Show scans -->
      <xsl:param name="image_frame" select="true()"/>
      <!-- Edition -->
      <xsl:variable name="edition_array" as="element()*">
        <Item>Diplomatic</Item> <!-- IT: Per l'elaborazione nei moduli:
$edition_array[1] -->
        <Item>Critical</Item> <!-- IT: Per l'elaborazione nei moduli:
$edition_array[2] -->
    </xsl:variable>
</xsl:stylesheet>
```

File "inferface_control.js"

```
/**
 * Interface Control jQuery
 * Version 0.1 (201210)
 **/

$( function() {
      var keycount=0;
      var fulltogg=false;
      var pp_temp_val=$(".main_pp_select").val();
      $('#edval span').text($("input[name=edition_r]:checked").val());

      /* Funzioni */
            function setallselect(pp_num){
                  $(".main_pp_select").each(function(){
                        $(this).val( pp_num );
                  });
            }
            function gotopage(pp_val, state){
                  var edition=$
("input[name=edition_r]:checked").val().toLowerCase();

                  $('#text_elem').remove();
                  $('<iframe id="text_elem" class="scroll-
ios">').appendTo('#text_cont');
                  $("#text_elem")
                        .attr("src",
"data/"+edition+"/page_"+pp_val+"_"+edition+".html")/**/
                        .hide()
                        .fadeIn(400);
                  if ($("#text_cont-add").length > 0){
                        var edition_add=$("input[name=edition_r-
add]:checked").val().toLowerCase();
                        $('#text_elem-add').remove();
                        $('<iframe id="text_elem-add" class="scroll-
```

```
ios">').appendTo('#text_cont-add');
                              $("#text_elem-add")
                                    .attr("src",
"data/"+edition_add+"/page_"+pp_val+"_"+edition_add+".html")/**/
                                    .hide()
                                    .fadeIn(400);
                              $('#zvalopz')
                                    .text($("input[name=edition_r-
add]:checked").val())
                                    .hide()
                                    .fadeIn(200);
                        } else {
                              $('#zval>span')
                                    .hide()
                                    .fadeIn(200);
                        }
                        $('#central_page_number span').text(pp_val).hide()
                              .fadeIn(200);
                        $('#edval span')
                              .hide()
                              .fadeIn(200);
                        $("#iviewerImage").attr("src", "images/null.jpg"); //
Loading...
                        $('#folio_page_number').val(pp_val).change(); // IT:
Questo attiva l'evento nel file js/plugin/jquery.iviewer
                        preload([
                              'images/'+$('.main_pp_select
option:selected').prev('option').val()+'.jpg',
                              'images/'+$('.main_pp_select
option:selected').next('option').val()+'.jpg'
                        ]);
                        if($("#image_elem").css('display')=="none"){
                              $("#image_elem").show();
                              $("#image_tool").show();
                              $("#thumb_cont").hide();
                        }
                  }
                  function gotoedition(pp_val, pp_el, frame_id, parent_id){
                        $('#'+frame_id).remove();
                        $('<iframe
id="'+frame_id+'">').appendTo('#'+parent_id);
                        $('#'+frame_id)
                              .attr("src",
"data/"+pp_el+"/page_"+pp_val+"_"+pp_el+".html");
                        var pp_el_upp = pp_el;
                        pp_el_upp = pp_el_upp.toLowerCase().replace(/\b[a-z]/g,
function(letter) {
                              return letter.toUpperCase();
                        });
                        if (frame_id.indexOf("-add")>-1) {
                              $('#zvalopz').text(pp_el_upp);
                        } else{
                              $('#edval span').text(pp_el_upp);
                        }
                  }
                  function preload(arrayOfImages) {
                        $(arrayOfImages).each(function(){
                              $('<img/>')[0].src = this;
                        });
                  }
      /* / Funzioni */

      /* Gestione eventi */
            $('.main_pp_select').change(function(){
                  window.location.hash = $(this).val();
```

```
            });
            $('input[name=edition_r]').change(function(){
                  gotoedition(location.hash.replace( /^#/, '' ),$
(this).val().toLowerCase(),"text_elem", "text_cont");
            });
            $("input[name=edition_r-add]").live("change", function(){
                  gotoedition(location.hash.replace( /^#/, '' ),$
(this).val().toLowerCase(),"text_elem-add", "text_cont-add");
            });
            $("#image_cont").mouseenter(function(){
                  if($("#image_elem").css('display')!="none"){
                        $("#image_tool").show("slow");
                        keycount=1;
                  }
            });
            $("#image_cont").mouseleave(function(){
                  $("#image_tool").hide("slow");
                  keycount=0;
            });

      /* / Gestione eventi */


      /* Gestione click */
            $("#home_title").click(function(){
                  window.location="index.html";
            });

            $("#main_left_arrow").click(function(){
                  if($('.main_pp_select
option:selected').prev('option').val()){
                        window.location.hash = $('.main_pp_select
option:selected').prev('option').val();
                  }
            });
            $("#main_right_arrow").click(function(){
                        if($('.main_pp_select
option:selected').next('option').val()){
                              window.location.hash = $('.main_pp_select
option:selected').next('option').val();
                  }
            });

            $("#main_left_menu").click(function(){
                  if($("#main_left_menu-openlink").css('display')!
="none"){
                        $("#main_left_frame header").show('slide',
{direction: 'left'}, 1000);
                        $("#main_left_frame-single
header").show('slide', {direction: 'left'}, 1000);
                        $("#main_left_menu-openlink").toggle();
                        $("#main_left_menu-closelink").toggle();
                        keycount=1;
                  } else{
                        $("#main_left_frame header").hide('slide',
{direction: 'left'}, 1000);
                        $("#main_left_frame-single
header").hide('slide', {direction: 'left'}, 1000);
                        $("#main_left_menu-openlink").toggle();
                        $("#main_left_menu-closelink").toggle();
                        keycount=0;
                  }
            });

            $("#main_right_menu").click(function(){
```

```
                    if($("#main_right_menu-openlink").css('display')!
="none"){
                            $("#main_right_frame header").show('slide',
{direction: 'right'}, 1000);
                            $("#main_right_menu-openlink").toggle();
                            $("#main_right_menu-closelink").toggle();
                            keycount=0;
                    } else{
                            $("#main_right_frame header").hide('slide',
{direction: 'right'}, 1000);
                            $("#main_right_menu-openlink").toggle();
                            $("#main_right_menu-closelink").toggle();
                            keycount=1;
                    }
             });

             $("#thumb_link").click(function(){
                    if($("#image_elem").css('display')=="none"){
                            $("#image_elem").show();
                            $("#image_tool").show();
                            $("#thumb_cont").hide();
                    } else{
                            $("#image_elem").hide();
                            $("#image_tool").hide();
                            $("#thumb_cont").show();
                    }
             });

             $(".thumb_single").click(function(){
                    window.location.hash = $
(this).attr('id').replace( '_small', '' );
                    $("#thumb_link").trigger('click');
                    if($("#main_right_frame header").css('display')!
="none"){
                            $("#main_right_menu").trigger('click');
                    }

             });

             // MODE -
             $("#txtimg_link").click(function(){
                    if($(this).attr("class")!="current_mode"){
                            $("#txtimg_link").addClass("current_mode");
                            $("#imgimg_link").removeClass("current_mode");
                            $("#txttxt_link").removeClass("current_mode");

                            $("#text_cont-add").remove();
                            $("#radio_edition-add").remove();

                            $("#text_menu").show();
                            $("#text_cont").show();
                            $("#image_menu").show();
                            $("#image_cont").show();

                            $('#zvalint').show();
                            $('#zvalopz').text("");

                    }
             });
             $("#txttxt_link").click(function(){
                    if($(this).attr("class")!="current_mode"){
                            $("#txttxt_link").addClass("current_mode");
                            $("#txtimg_link").removeClass("current_mode");
                            $("#imgimg_link").removeClass("current_mode");
```

```
                        $("#image_menu").hide();
                        $("#image_cont").hide();

                        $('#text_cont')
                                .clone()
                                .attr("id", "text_cont-add")
                                .insertAfter("#right_header")
                        ;
                        $('#text_cont-add>#text_elem')
                                .attr("id", "text_elem-add")
                        ;

                        $('#zvalint').hide();
                        $('#zvalopz').text($
("input[name=edition_r]:checked").val());

                        $('#radio_edition')
                                .clone()
                                .attr("id", "radio_edition-add")
                                .appendTo("#right_header")
                        ;

                        $('#radio_edition-
add>input[name="edition_r"]').each(function(index) {
                                this.name = this.name + "-add";
                        });
                }
        });
        // /MODE -
    /* / Gestione click */

    /* HASH CHANGE - ba.bbq plugin */
        $(window).hashchange( function(){
                var hash = location.hash;

                if(hash){
                        gotopage(hash.replace( /^#/, '' ), "none");
                        setallselect(hash.replace( /^#/, '' ));
                }else{
                        window.location.hash=$(".main_pp_select
option:first").val();
                }
        })
        $(window).hashchange();
    /* / HASH CHANGE - ba.bbq plugin */
});
```

File “keydown.js”

```
/**
 * KeyDown jQuery
 * Version 0.1 (201210)
 **/
$(document).keydown(function(e){
    if (e.keyCode == 37) {
        if($('.main_pp_select option:selected').prev('option').val())
{
            window.location.hash = $('.main_pp_select
option:selected').prev('option').val();
        }
```

```
            return false;
      }
      if (e.keyCode == 39) {
            if($('.main_pp_select option:selected').next('option').val())
{
                  window.location.hash = $('.main_pp_select
option:selected').next('option').val();
            }
            return false;
      }
      if (e.keyCode == 27) {
            if($("#main_left_menu-openlink").css('display')=="none"){
                  $("#main_left_frame header").hide('slide', {direction:
'left'}, 1000);
                  $("#main_left_menu-openlink").toggle();
                  $("#main_left_menu-closelink").toggle();
            }
            if($("#main_right_menu-openlink").css('display')=="none"){
                  $("#main_right_frame header").hide('slide', {direction:
'right'}, 1000);
                  $("#main_right_menu-openlink").toggle();
                  $("#main_right_menu-closelink").toggle();
            }
            keycount=0;
            return false;
      }
      if (e.keyCode == 189){
            if(keycount==1){
                  if($("#main_left_menu-
openlink").css('display')=="none"){
                        $("#main_left_frame header").hide('slide',
{direction: 'left'}, 1000);
                        $("#main_left_menu-openlink").toggle();
                        $("#main_left_menu-closelink").toggle();
                  keycount=0;
                  }

            }
            if(keycount==0){
                  if($("#main_right_menu-
openlink").css('display')=="none"){
                        $("#main_right_frame header").hide('slide',
{direction: 'right'}, 1000);
                        $("#main_right_menu-openlink").toggle();
                        $("#main_right_menu-closelink").toggle();
                  keycount=1
                  }

            }
            return false;
      }
});
```