ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

PHILOEDITOR 3.0: UN WEB EDITOR PER LA RICERCA FILOLOGICA

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
Gioia Donati

Sessione II
A.A. 2015/2016

Indice

Introduzione

La presente dissertazione descrive il progetto PhiloEditor 3.0, versione espansa e riveduta della precedente (2.0). I due traguardi di questa versione sono l'aggiunta del supporto ad opere multiple e la gestione dell'output in formato TEI P5. Altri cambiamenti più o meno significativi hanno risguardato il refactoring della maggior parte del codice, in modo da garantire la compatibilità con le nuove strutture dati implementate ai fini della gestione multi opera. Infine sono stati risolti numerosi bug ed è stato aggiunto un sistema di accounting, migliorando al contempo la gestione dei permessi sui file.

Nel corso della storia una delle sfide che l'uomo periodicamente si è trovato ad affrontare con l'avanzare delle tecnologie è la trasmissione della conoscenza e della cultura. L'edizione elettronica ha portato ad una rivoluzione paragonabile a quella che ci fu con la nascita della stampa. I progetti di digitalizzazione e di creazione di archivi elettronici sono in continua crescita. Si pensi ad esempio al progetto Codex sinaiticus[1], nato con l'obiettivo di riunire in forma digitale il manoscritto risalente al IV secolo e contenente la Bibbia cristiana in greco, oppure al "Vespasiano da Bisticci, Lettere"[2] che è un progetto di ricerca del CRR-MM (Centro di Risorse per la Ricerca MultiMediale dell'Ateneo di Bologna) e prevede la realizzazione di un'edizione digitale ipertestuale delle lettere risalenti al XV secolo inviate e ricevute dal copista Vespasiano da Bisticci.

Il tipo di supporto modifica quindi profondamente il metodo di scrivere e diffondere la conoscenza. Nel caso specifico del supporto elettronico è possibile, in spazi molto ridotti e a costi contenuti, la consultazione e la conservazione di grandi quantità di dati. Permette inoltre di condividere informazioni e consentire una consultazione simultanea di un documento a un ampio numero di persone, indipendentemente dal luogo in cui si trovano. Il supporto elettronico assume quindi tutte quelle caratteristiche che lo rendono uno strumento ideale per una comunità di ricercatori che voglia lavorare su grandi quantità di dati e debba farlo in tempo reale e in condivisione planetaria[DI98].

[1] *Codex Sinaiticus Project*: http://codexsinaiticus.org/.
[2] Vespasiano da Bisticci, Lettere: http://vespasianodabisticciletters.unibo.it/.

In seguito alla diffusione di questo mezzo di comunicazione sono anche cambiate le metodologie della ricerca umanistica. Nello specifico, in questa dissertazione ci si concentra sull'analisi e sullo sviluppo di strumenti innovativi per la ricerca ecdotica. Oggi il filologo, tramite il supporto di appositi tool informatici, ha la possibilità di pubblicare su supporto digitale i risultati degli studi compiuti su un particolare testo, dando così vita a un'edizione digitale. Attualmente l'insieme degli strumenti che rendono possibile la pubblicazione di tali edizioni risultano presentare alcuni problemi, come ad esempio la mancanza di uno standard, che portano alla creazione di confusione sia da parte dell'editore che da parte del lettore. Inoltre non sono ancora presenti dei tool che si propongano come supporti adeguati per la critica delle varianti, disciplina molto importante nell'ambito specifico della filologia d'autore. L'applicazione delle nuove tecnologie a questo settore risulta apportare numerosi vantaggi al ricercatore. Nella filologia d'autore è infatti fondamentale poter lavorare su testi dinamici che mostrino le diverse varianti del testo, ovvero cambiamenti presenti da una versione all'altra. Il software PhiloEditor, sviluppato in seno all'Università di Bologna, si inserisce nell'ambito della creazione di edizioni digitali, più precisamente di *edizioni genetiche analitiche*. Il sistema consente la produzione di questo nuovo tipo di edizione perché permette lo studio di testi letterari su due livelli distinti, quello filologico e quello interpretativo. Dal punto di vista filologico PhiloEditor da la possibilità di visualizzare due edizioni e di vedere sincronicamente le varianti, mentre dal punto di vista interpretativo permette di intervenire sul testo marcando le varianti a seconda delle esigenze di studio e apportando al testo un valore aggiunto. Dopo il successo dell'esperimento su due versioni de I promessi sposi di Alessandro Manzoni, si è deciso di ampliare il progetto.

La nuova versione PhiloEditor 3.0 è in grado di supportare lo studio di più opere. Nell'applicazione sono presenti oltre che l'opera manzoniana anche alcuni capitoli di due versioni de "Le avventura di Pinocchio" di Carlo Collodi. Il progetto ha inserito anche la possibilità di esportare le edizioni in TEI P5, che è ormai un requisito fondamentale per questo tipo di applicazione. La nuova versione di PhiloEditor si presenta inoltre più stabile rispetto a quella precedente, e ha introdotto nuove funzionalità base di cui prima era sprovvista.

Nel corso di questa dissertazione verrà fatta una breve panoramica generale sulla *filologia digitale* e sulle forme in cui essa si è evoluta nel corso dell'ultimo decennio. Verrà poi descritto come PhiloEditor ha migliorato l'esperienza del filologo facilitando la critica di varianti e la creazione e visualizzazione di edizioni genetiche analitiche. Infine, vedremo alcuni interessanti sviluppi che il software può offrire, come ad esempio un'ulteriore espansione della raccolta di documenti, l'introduzione di scansioni dei documenti o il supporto ad altri tipi di dispositivi.

Capitolo 1

La filologia e le edizioni digitali

Tra le discipline umanistiche la filologia è forse quella che più di tutte può trarre benefici dall'incontro con l'informatica. Per questo motivo gli ambiti in cui essa può essere applicata sono spesso oggetto di dibattito tra i filologi, che assumo opinioni talvolta contrastanti sull'argomento. Tra gli umanisti c'è chi infatti diffida dell'idea di utilizzare l'informatica in quanto scienza nello studio filologico e la vede relegata a mero strumento per la creazione delle proprie edizioni [Tom01]. Nonostante ciò, nel corso dell'ultimo decennio è divenuta ormai chiara ed evidente a tutti la potenzialità dell'informatica applicata allo studio letterario, anche grazie alla diffusione sempre maggiore delle tecnologie legate al cosiddetto *Web Semantico*.

Un pioniere dell'utilizzo informatico nell'ambito filologico è stato il filologo italiano Domenico De Robertis che intorno agli anni '90 si cimentò nell'impresa di rappresentare le varianti di opere leopardiane sul web. Nonostante il suo sito oggi non sia più raggiungibile, a causa dell'implementazione fatta con tecnologie ormai entrate in disuso, la sua idea rimane incredibilmente innovativa e attuale [BDIV16]. L'esperimento intrapreso da De Robertis, inteso come la realizzazione di edizioni digitali accessibili e modificabili con un computer, è stato di grande ispirazione per lo sviluppo e l'innovazione delle tecniche dello studio filologico, e ha dato vita ad un ambito di ricerca autonomo spesso definito *filologia digitale.*

In questo capitolo verranno trattati nel dettaglio i benefici e i problemi riscontrati dal filologo durante il suo lavoro e di come l'informatica può venire in suo aiuto, e si esporranno esempi notevoli di attuali software esistenti per la creazione di edizioni critiche. Infine verrà introdotta la codifica di testi XML in formato TEI, divenuto ormai a tutti gli effetti la "lingua franca" della filologia digitale.

1.1 I limiti della filologia su carta

L'avvento del medium informatico nello studio filologico cambia radicalmente alcuni dei canoni di questa disciplina e fa emergere in maniera chiara i limiti storici della filologia "su carta". In [BDIV16] vengono descritti in maniera dettagliata gli effetti di questa interazione sia sullo studio di testi letterari in genere che sull'atto filologico vero e proprio. Nel testo vengono divisi quattro ambiti, all'interno dei quali l'ecosistema digitale offre prospettive di cambiamento anche radicale. Il primo ambito descritto è quello *temporale*: un testo digitale, infatti, non esiste più soltanto come un'entità singola ma come tutto l'insieme delle sue varianti diacroniche. Questo ha ovviamente un sensibile impatto anche sulla filologia: le edizioni più comuni hanno spesso un approccio diplomatico piuttosto che critico, concentrandosi spesso sulle varianti di un testo piuttosto che sulle diverse fasi della sua genesi. Il secondo ambito descritto è quello *spaziale*. Prima della digitalizzazione dei testi, lo studio letterario tendeva spesso ad essere *unitario*: il letterato studiava il testo come entità isolata, senza avere a disposizione informazioni sulla sua interpretazione topografica (altri testi dello stesso autore, edizioni critiche del testo, testi simili fatti da altri autori…). L'informatica invece permette di collegare semanticamente i vari testi, diminuendo sensibilmente i limiti spaziali presenti in passato. Questo si riflette anche sullo studio filologico: avendo a disposizione una grande varietà di testimoni semanticamente collegati tra di loro, il "Filologo digitale" tende spesso a preferire la pubblicazione parallela di numerosi manoscritti piuttosto che la ricostruzione di un un'unica edizione finale, ritenendo che quest'ultima soluzione comporti il rischio di perdere l'informazione storica trasmessa dai vari testimoni. Il terzo ambito descritto nell'articolo è l'ambito *formale*. In un'edizione digitale, infatti, la forma in cui il testo è riportato assume un'importanza maggiore e diventa in alcuni casi parte dell'ermeneutica del testo. Si pensi alle numerose edizioni critiche presenti sul web in cui viene mostrata la scannerizzazione di un documento storico e vengono riportate le correzioni in forma stampata[3]. Il nuovo approccio allo studio della forma del testo si riflette anche sulla filologia: spesso,

[3] Esempi notevoli di questa categoria includono il *Vercelli Book Digital* (http://vbd.humnet.unipi.it/) o Electronic Beowulf 4.0 (http://ebeowulf.uky.edu/ebeo4.0/start.html).

infatti, nell'edizione digitale la rappresentazione grafica del testo assume un'importanza tale da diventare essa stessa l'edizione critica, specialmente nel caso in cui l'edizione digitalizzata abbia una rilevante importanza storica e iconografica. Infine il quarto ambito, ma non ultimo per importanza, è quello Sociale. Uno dei cardini fondamentali della ricerca accademica è costituito dalla condivisione delle conoscenza raccolta dai vari centri di ricerca sparsi in tutto il mondo. L'informatica semplifica notevolmente la condivisione di queste informazioni, poiché grazie alle nuove tecnologie sociali le risorse risultano facilmente reperibili e accessibili, indipendentemente dalla collocazione geografica dell'interessato. La facile condivisione dell'informazione porta ad una importante novità: il lettore non occupa più un ruolo passivo ma diventa lui stesso un critico. In quest'ottica le edizioni, spesso, non sono più relative ad un singolo autore ma diventano edizioni *della comunità di ricerca,* così come le varie opinioni raccolte dai diversi lettori possono contribuire a formare interpretazioni collaborative.

1.2 La filologia d'autore e la critica delle varianti

La filologia è una disciplina vasta al cui interno si possono distinguere diversi filoni. Recentemente, tra di essi, è emersa come disciplina autonoma la filologia d'autore. Essa si occupa dei problemi e dei metodi relativi all'edizione di opere conservate da uno o più manoscritti autografati, oppure da stampe sorvegliate dall'autore [Ros03]. Le varianti presenti in tali opere testimoniano una mutazione della volontà dell'autore e il loro studio costituisce un punto critico per il filologo. Con la filologia d'autore e la critica delle varianti nasce dunque un nuovo modo di considerare le opere letterarie. Il testo viene visto come "un organismo in evoluzione, poiché viene considerato come espressione di una ricerca, il cui prodotto finale è soltanto il risultato delle progressive approssimazioni a un valore che è dipendente dal rapporto con i testi precedenti" [IR10]. Il filologo che si cimenta in questa disciplina si ritrova dunque ad intraprendere un laborioso lavoro di interpretazione delle varianti. Tale lavoro, senza un adeguato supporto informatico, risulta più complesso, poiché lo studioso deve lavorare separatamente sulle versioni del testo senza avere a disposizione un'edizione "dinamica" che rappresenti il confronto tra di esse. La seconda parte di questa

dissertazione sarà dedicata alla descrizione del progetto PhiloEditor, pensato proprio per facilitare la critica delle varanti nell'ambito della filologia d'autore, in modo da sollevare il filologo dal compito di svolgere lavori marginali e permettergli di concentrarsi invece sui compiti di interpretazione e analisi che richiedono le competenze di specialisti e non possono essere svolti in maniera automatica.

1.3 Edizioni digitali: lo stato dell'arte

Quello di edizione digitale è un concetto ambiguo soggetto a differenti interpretazioni. Ambiguo poiché esso può riferirsi sia alle edizioni digitali intese come la trasposizione di una studio filologico su un sistema informatico, che ad "un insieme di attività relative allo sviluppo di programmi che siano in grado di favorire l'interazione fra l'editore e l'elaboratore" [Boz06]. Un primo approccio per definire l'edizione digitale può essere classificarne le varie tipologie in base ad una serie di caratteristiche. In [DP15] viene effettuata una classificazione plausibile delle diverse tipologie di raccolte di materiali testuali, identificabili come "edizione elettronica" o "digitale". Nell'ambito di tale classificazione, una delle categorie definite è quella di edizioni digitali di singole opere o autori, che codificano una o più versioni di un'unica opera permettendo, eventualmente, la visualizzazione parallela delle varianti. Si tratta della tipologia più comune di edizione digitale, nonché di quella che interessa più da vicino il filologo, risultando in un certo senso particolarmente affine al suo operato.
Vengono di seguito illustrati alcuni esempi notevoli di questo tipo di edizione digitale: essi serviranno, oltre che come riferimento, ad evidenziare alcune falle e mancanze o al contrario alcuni pregi e qualità da cui è nata l'idea di PhiloEditor 3.0.

Electronic Beowulf 4.0

L'eletronic Beowulf[4] è un ottimo esempio rappresentativo dei web editor per la filologia anglosassone. Qui il testo del poema, che per l'appunto è l'anonima opera intitolata Beowulf, è accompagnato da immagini che mostrano la

[4] Ultima versione accessibile al link http://ebeowulf.uky.edu/ebeo4.0/CD/main.html

digitalizzazione del manoscritto. Il progetto, partito nel 1999, era originariamente rilasciato su DVD per PC e Mac ed è stato poi convertito in un'applicazione free-download compatibile con i principali browser dell'epoca (Electronic Beowulf 2.0). La versione 3.0 del software ha aggiunto la maggior parte delle feature grafiche presenti in quella attuale, ed è stata riscritta quasi interamente nel 2015 per eliminare l'applet Java e convertire tutto il codice del client in Javascript.

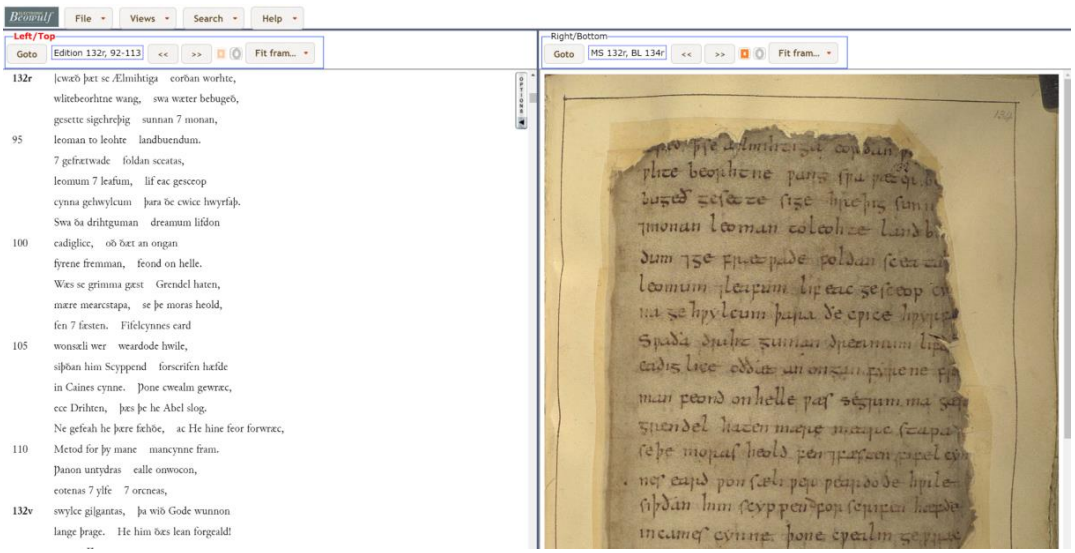

Figura 1.1 L'eletronic Beowulf: interfaccia principale

La figura 1.1 mostra l'interfaccia principale dell'applicazione. Essa si presenta divisa in due sezioni: a destra viene mostrato il testo del poema, a sinistra la relativa pagina digitalizzata. L'Eletronic Beowulf 4.0 è un'applicazione web gratuita ed è implementata per mezzo di tecnologie standard (HTML, CSS, JavaScript). L'applicazione non si limita ad offrire la sola consultazione del poema ma anche quella del suo apparato critico costituito da numerosi restauri e correzioni editoriali, dando la possibilità di effettuare studi della grammatica e della metrica. Mette inoltre a disposizione diverse modalità di visualizzazione del poema e permette una navigazione ipertestuale: è possibile cioè scorrere il manoscritto in entrambe le direzioni e raggiungere in maniera diretta una determinata pagina o riga. Tra le funzionalità più interessanti troviamo la possibilità di accedere a una tabella di statistiche metriche. L'Electronic Beowulf offre una vasta gamma di strumenti per lo studio del testo ed è indubbiamente uno strumento di rilievo per la ricerca filologica. Tuttavia, esso presenta ancora delle lacune, riscontrabili principalmente nel sistema di navigazione, che risulta essere

poco intuitivo e praticamente inaccessibile senza consultare adeguatamente la corposa guida che si trova online[5].

The Digital Vercelli Book

Il Vercelli Book[6] digitale nasce nel 2003 su iniziativa di Roberto Rosselli Del Turco, ricercatore di filologia germanica presso l'Università di Torino. Il progetto ha previsto la digitalizzazione del prezioso manoscritto conservato nella città di Vercelli e che rappresenta una delle prime testimonianze scritte in inglese antico.

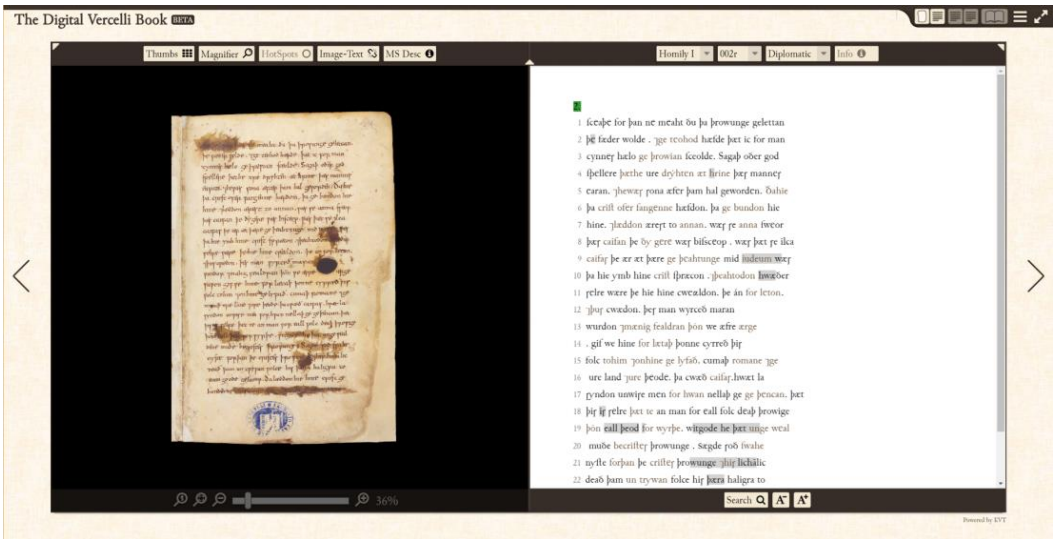

Figura 1.2 The Digital Vercelli Book: interfaccia principale

Lo scopo principale del progetto era rendere l'opera più facilmente accessibile e diminuirne il pericolo di usura. L'edizione è stata pubblicata sul World Wild Web grazie al software EVT (Edition Visualization Tool[7]) Questo software, inizialmente pensato unicamente per gestire la codifica del Vercelli, è ormai diventato un tool stand-alone ed è stato sviluppato con standard open source per garantirne il funzionamento ottimale anche su lunga durata. Tra le feature che presenta spiccano una integrazione completa con lo standard TEI P5 e una serie di strumenti che ottimizzano il rendering e l'interazione con il manoscritto digitalizzato. La flessibilità dell'implementazione e la possibile apertura futura al supporto di opere diverse è indubbiamente tra gli aspetti più interessanti del progetto: ad oggi già altre edizioni digitali fanno uso di EVT come ad esempio il

[5] http://ebeowulf.uky.edu/
[6] Ultima versione disponibile al link http://vbd.humnet.unipi.it/beta2/#doc=Homily-1&page=VB_fol_002r
[7] https://sourceforge.net/projects/evt-project/

progetto *Codice Pelavicino Digitale*[8]. Il software per ora supporta solo edizioni di livello diplomatico e diplomatico-interpretativo; il supporto alle edizioni di livello critico è attualmente in fase di studio e verrà aggiunto in futuro.

Nietzsche Source

Come ultimo esempio si è scelto di prendere in analisi un progetto di natura diversa da quelli visti finora. Nietzsche Source[9] è un archivio digitale dedicato alla pubblicazione di contenuti scientifici e traduzioni riguardati la vita e l'opera di Friedrich Nietzsche. Lo scopo dell'applicazione non è quindi quello di costruire un ambiente digitale munito di tool per la critica del testo, ma di creare "un luogo unico in cui far convergere tutto ciò di cui disponiamo su Nietsche" [DI98]. Il suo ideatore, il filosofo Paolo D'Iorio, ha pensato infatti che il buon funzionamento del progetto si fondasse su due presupposti. Il primo è la presenza di un comitato scientifico, che si occupasse della validazione dei documenti raccolti dalla piattaforma e che quindi garantisse prestigio al servizio offerto. Il secondo è la digitalizzazione dei manoscritti e documenti che servono come base per la ricerca scientifica. L'applicazione di D'Iorio ambisce in particolar modo a sfruttare il mezzo elettronico nell'ambito dello spazio. Si vuole cioè rendere disponibile al lettore la più completa e valida raccolta di informazioni sotto forma di digitalizzazioni, biografie, edizioni critiche esistenti a riguardo. L'intenzione dell'applicazione non è però quella di escludere i contributi che i lettori potrebbero apportare, che anzi rappresentano un tassello fondamentale del progetto. Tuttavia, per mantenere una qualità alta delle fonti, gli eventuali contributi devono prima passare al vaglio del comitato scientifico. Di questo progetto sono sicuramente notevoli l'ottima organizzazione dei contenuti e la ricchezza e la qualità dell'archivio digitale di cui dispone.

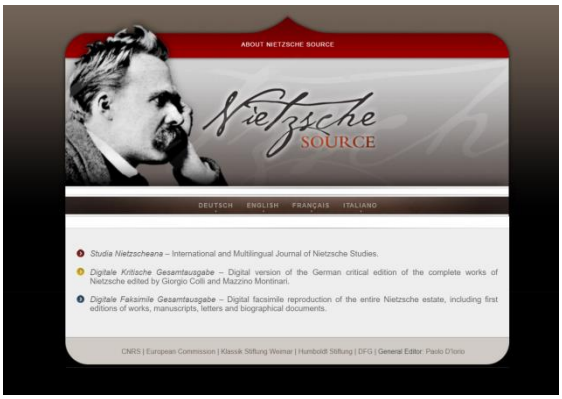

Figura 1.3 Nietzsche Source: pagina di benvenuto

[8] http://pelavicino.labcd.unipi.it/
[9] http://www.nietzschesource.org/

Analizzando nel complesso le piattaforme descritte risaltano prevalentemente due caratteristiche. La prima è la forte disomogeneità da un editor all'altro, che si concretizza nella diversità tra le interfacce, nei modi in cui vengono mostrati e gestiti i testimoni, nel differente utilizzo degli apparatici critici e nella rappresentazione dei confronti. Tutto questo porta alla creazione di una grande confusione ed è un chiaro indicatore della mancanza di uno standard nella creazione di edizioni digitali, che invece è ben consolidato per quanto riguarda quelle a stampa. Una delle conseguenze della mancanza di uno standard è l'aumento della curva di apprendimento da parte dell'utente: diventa cioè difficile usufruire in maniera vantaggiosa delle funzionalità messe a disposizione dai vari editor, senza pensare a quanto sarebbe difficile rendere interoperabili le varie piattaforme. Il secondo fattore che risalta subito all'occhio è la settorialità delle varie piattaforme. Infatti, se da una parte è possibile trovare una o più applicazione che permettano la creazione di un'edizione critica di una determinata opera, dall'altra risulta quasi impossibile trovare un'unica applicazione che permetta lo studio e la realizzazione di edizioni critiche di opere diverse.

1.3.1 Lo standard TEI

La TEI (Text Encoding Initiative[10]) è un consorzio internazionale nato nel 1987 con l'obiettivo principale di sviluppare, mantenere e diffondere uno standard per la rappresentazione di materiali testuali in formato digitale. L'idea è nata in seguito alla diffusione delle edizioni digitali, poiché mancava un metodo che garantisse la conservazione a lungo termine dei dati elettronici. La necessità di creare un standard è nata anche dal fatto che solitamente si desidera una certa interoperabilità per lo scambio di informazioni tra utenti e applicazioni, nonché nello scambio di risorse eterogenee.

TEI si basa sul meta-liguaggio di markup XML e definisce un tagset specifico per la rappresentazione di documenti. Il vantaggio di un documento TEI rispetto ad altri documenti come PDF, DOC e HTML consiste nell'essere completamente indipendente da software esterni, requisito necessario per garantire la manutenibilità del documento nel tempo. Inoltre, essendo fondamentalmente un

[10] http://www.tei-c.org/index.xml

documento XML, il documento TEI è interpretabile su qualsiasi sistema operativo ed è quindi anche machine-readable. Infine il consorzio TEI-C, che è il nome del comitato che mantiene e sviluppa TEI, ne garantisce la manutenzione e gli aggiornamenti, rendendo lo standard preservabile nel futuro. L'ultima versione rilasciata è TEI P5 che è suddivisa in 21 moduli estendibili, ciascuno dei quali approfondisce un aspetto specialistico del testo che si vuole rappresentare.

1.3.2 Tei Critical Apparatus

Il modulo TEI specializzato per la gestione delle informazioni che riguardano l'evoluzione del testo nelle sue varianti e per la codifica dell'edizione critica è il TEI Critical Apparatus[11]. La finalità primaria di tale modulo è quella di soddisfare le esigenze di codifica dei filologi, nate a seguito della nascita e della diffusione dell'edizioni elettroniche. Di seguito vengono spiegati gli elementi più comuni e basilari per la codifica di un apparato critico. TEI infatti ha il grande vantaggio di permettere diverse modalità di codifica di varianti, spesso con alti livelli di specificità.

Il critical apparatus suddivide logicamente il markup in tre classi di elementi:

- L'apparatus entry, che rappresenta il contenuto delle varianti,
- L'apparatus readings, che rappresenta il set di lezioni individuali all'interno di una variante,
- L'apparatus witness che rappresenta l'insieme di testimoni collegati alle singole lezioni.

```
<app>
 <rdg wit="#El">Experience     though     noon     Auctoritee</rdg>
 <rdg wit="#La">Experiment      thouh     noon     Auctoritee</rdg>
 <rdg wit="#Ra2">Eryment       though      none      auctorite</rdg>
</app>
```

Figura 1.4 TEI Critical Apparatus: esempio

Le singole variazioni del testo vengono codificate per mezzo dell'elemento **<app>,** il quale raggruppa insieme tutte le lezioni che costituiscono la variante.

[11]TEI Critical Apparatus: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/TC.html

Le varie lezione sono invece codificate per mezzo dell'elemento **<rdg>** e sono contenute nell'apparato critico di appartenenza. Oltre all'elemento **<rdg>** può essere utilizzato anche il tag **<lem>** per codificare la lezione del testo base o di preferenza. Ad ogni **<rdg>** deve essere assegnato un attributo **@wit** (oppure **@source**, se i testimoni vengono descritti dall'elemento **<listBibl>**) che contiene il riferimento al testimone che attesta quella lezione, o in alternativa può essere utilizzato l'elemento **<wit>**, da inserire immediatamente dopo la lezione a cui fa riferimento. Inoltre possono essere presenti altri attributi come **@type** e **@cause,** che rispettivamente classificano la lezione secondo una tipologia funzionale o in base alla causa che ha portato alla registrazione della variazione, oppure come **@resp** e **@cert** che indicano rispettivamente il responsabile dell'intervento o interpretazione e il grado di certezza dell'intervento. Gli elementi **<rdg>** possono essere raggrupati in **<rdgGrp>** che può a sua volta assumere vari attributi come quelli appena descritti, che verranno ereditati dagli **<rdg>** e **<lem>** contenuti al suo interno. La scelta di utilizzare **<rdgGrp>** può essere dettata da varie motivazione tra cui la presenza di varianti con valori identici su uno o più attributi o varianti che formano una sequenza di variazione autonoma. I raggruppamenti possono essere annidati in maniera ricorsiva , in modo da permettere la classificazione delle varianti a qualsiasi livello di profondità.

I metodi proposti per collegare gli apparati che descrivono le variazioni testuali sono tre:

- *location-referenced method,*
- *double-end-point-attached method,*
- *parallel segmentation*.

Nel *location-referenced method* le variazioni vengono registrate all'esterno del corpo del testo, fornendo un conveniente metodo per la codifica degli apparati tradizionali a stampa. In questo metodo l'apparato è collegato al testo base, indicando esplicitamente solo il blocco del testo in cui è presenta la variante a cui fa riferimento. Il modello *double-end-point-attached method* ha un funzionamento analogo al precedente, anche qui le variazioni vengono registrare in un file separato ma l'inizio e la fine del lemma di riferimento del testo base deve essere

indicato esplicitamente attraverso l'utilizzo dei due attributi **`@form`** (che ne indica l'inizio) e **`@to`** (che ne indica al fine). L'ultimo metodo è il *parallel segmentation method*. Questo modello, diversamente dagli altri due, prevede la registrazione delle varianti all'interno del testo, risultando così di più intuitivo e leggibile e di facile estrazione mediante software appositi ma di contro è sconveniente per la modellazione di apparatici critici più complessi in cui sono presenti casi di variazioni sovrapposte poiché risulta impossibile effettuare un collegamento non ambiguo di ogni variante con il proprio lemma.

Capitolo 2

PhiloEditor 3.0 e le edizioni genetiche analitiche

Esistono numerosi editor testuali che si propongono come validi strumenti sia per lo studio filologico che la pubblicazione e creazione di edizioni critiche digitali. Qualche esempio interessante è stato trattato nel capitolo 1 di questa dissertazione. PhiloEditor 3.0 è un progetto che ambisce ad entrare a far parte di questo insieme, ma introducendo un nuovo modo di studiare i testi. Il progetto mira ad essere un editor che supporti gli studi relativi alla filologia d'autore e introduce la creazione di un nuovo tipo di edizioni che sono state definite dalla prof.ssa Paola Italia con il nome di *edizioni genetiche analitiche* [DIV14]. PhiloEditor 3.0, infatti, supporta principalmente il lavoro del filologo su due livelli: quello puramente filologico, che consiste nella visualizzazione delle due edizioni prese in esame e nel vedere sincronicamente le varianti, e quello interpretativo, in cui PhiloEditor permette di intervenire sul testo digitale marcando le varianti a seconda delle esigenze di studio. Il nome dato a questo nuovo tipo di edizioni vuole appunto sottolineare l'aspetto filologico (genetico) e l'aspetto interpretativo (analitico) del lavoro. Nuovo perché fin ora non è mai esistito questo modo di studiare i testi ed è proprio questo il contributo più significativo del progetto PhiloEditor. Nel corso del capitolo viene illustrato come l'applicazione permette di creare questo nuovo prodotto e alcune delle caratteristiche più importanti che sono state aggiunte nella nuova versione.

2.1 La "*demanzonizzazione*"

Il progetto PhiloEditor è nato dall'interazione nell'ecosistema digitale tra filologia d'autore e informatica umanistica. Il progetto è iniziato prendendo come caso di studio *I promessi sposi* di Alessandro Manzoni. Dal punto di vista filologico la scelta è ricaduta sull'opera di Manzoni poiché è un caso di studio esemplare per la critica delle varianti. La Ventisettana e la Quarantana, che sono state le due versioni utilizzate nel progetto, presentano varianti confrontabili poiché tra i due testimoni non avviene una mutazione a livello strutturale, ma a livello formale e

lessicologico, con molte ma locali differenze. Dal punto di vista informatico questo caso di studio risulta ottimale per l'applicazione del *diffing*, tecnica di confronto tra due versioni diverse di un testo, tradizionalmente utilizzata in progetti informatici per gestire la continua evoluzione di documenti digitali come i codici sorgenti di software. La prima versione del progetto quindi si era concretizzata nell'implementazione di una applicazione sviluppata ad hoc sul modello dei due testimoni dei Promessi Sposi. Dopo il successo di questo esperimento si è deciso di estendere i casi d'uso gestiti da PhiloEditor. Il termine *demanzonizzazione* è nato proprio durante la progettazione della fase successiva del progetto, mirata a rendere PhiloEditor indipendente dall'opera manzoniana e quindi in grado di gestire più opere. Questa scelta è nata dal fatto che attualmente tra le varie applicazioni esistenti per lo studio e la critica di edizioni digitali non sono presenti piattaforme che permettano lo studio di opere di autori diversi. La realizzazione di un progetto di questa portata porterebbe numerosi benefici, poiché verrebbe incrementata ad esempio la possibilità di creare collegamenti tra un autore e l'altro. Basti pensare anche solo all'idea di poter passare in pochi secondi dallo studio di un autore all'altro su uno stesso sistema, che quindi potrebbe mettere in risalto fattori che prima perdevano d'evidenza a causa delle modalità diverse di studio e visualizzazione implementate dalle varie applicazioni. Un altro vantaggio consiste nell'abbassamento della curva di apprendimento: il lettore, infatti, non sarebbe più tenuto ad imparare a utilizzare applicazioni diverse ogni volta che decide di passare alla consultazione di opere di diversi autori.

Questa fase ha coinvolto l'inserimento di alcuni capitoli dell'opera più famosa di Collodi intitolata "Le avventure di pinocchio". La scelta non è stata casuale: PhiloEditor, infatti, è pensato per interagire con opere le cui versioni presentino tra loro differenze formali e non strutturali. Era quindi necessario utilizzare come campione un'opera che fosse in possesso di tali caratteristiche e insieme alla Prof.ssa Paola Italia la scelta è ricaduta proprio su "Le avventure di pinocchio". La nuova opera è stata aggiunta a seguito di quella manzoniana nell'apposito pannello laterale e le modalità di selezione e caricamento del capitolo sono rimaste pressoché identiche a quelle precedenti. L'unica aggiunta prettamente grafica è stata quella di modificare i check presenti in precedenza con delle etichette numeriche verdi con lo scopo di segnalare per ogni capitolo se e quante

sono le edizioni create. I due testimoni non vengono più identificati per mezzo del loro anno di pubblicazione, ma con le etichette *volume* e *rivista*. Le edizioni inserite sono per l'edizione in volume il testo di Castellani Polidoro e per l'edizione in rivista il testo di *Giornale per bambini*. Per l'edizione in rivista, pubblicando parti del racconto a pezzi, non è possibile indentificare un'unica data di pubblicazione ed è quindi risultato più chiaro distinguere i testimoni per mezzo di altre caratteristiche. L'ultima importante novità consiste nel fatto che le metodologie e categorie correttorie ora vengono caricate dinamicamente a seconda dell'opera che si è scelto di aprire. Al momento l'opera di Collodi non è ancora interamente presente sul sistema, poiché la fase di digitalizzazione dei testi è ancora in corso. Inoltre, non sono ancora state scelte le nuove tipologie di categorie e metodologie correttorie reputate necessarie per permettere un adeguato studio dell'opera.

Figura 2.1 PhiloEditor 3.0: interfaccia principale

2.2 Il diffing e lo studio delle varianti

Per la rappresentazione parallele delle varianti del testo PhiloEditor si è appoggiato in fase implementativa alle tecniche di *versioning*, che riguardano, tipicamente nell'ambito dell'ingegneria del software, la rappresentazione e conservazione delle differenti edizioni di un codice sorgente. Nell'ambito del *versioning* una delle operazioni più importanti è il *diffing* (talvolta indicato in letteratura come *differentiating*) che consiste, date due versioni di un testo, nell'individuare le differenze tra di esse e produrre dei delta. Un algoritmo di *diffing* si sostanzia generalmente in due operazioni:

- Identificazione dell'*oggetto minimo* di modifica nel testo, ovvero dell' entità atomica che verrà utilizzata nel rappresentare le differenze tra le versioni,
- Identificazione delle *operazioni di modifica* (inserimenti, cancellazioni, sostituzioni …) avvenute tra le due varianti.

Tipicamente nell'ambito informatico l'unità atomica scelta è la *riga di codice* o più raramente il *carattere*. Entrambe le opzioni risultano però inadatte al diffing di testi letterari, poiché la *riga del testo* è una entità difficile da individuare e poco significativa e il singolo *carattere* risulta eccessivamente specifico e produce differenze semanticamente inappropriate al campo letterario. Per questo motivo è stata utilizzata come unità base la *parola*. Come operazione di modifica invece viene invece identificata l'unica l'operazione *replace*: ogni cambiamento viene visto come una sostituzione, l'inserimento è visto come la sostituzione di un blank-space con una nuova frase o parola e il contrario vale per la cancellazione. Come output si otterrà quindi un unico testo contenente le differenze (varianti) segnalate in maniera opportuna. Sono possibili due tipi di visualizzazione delle varianti: quella verticale, che è la modalità di default del sistema, e quella orizzontale.

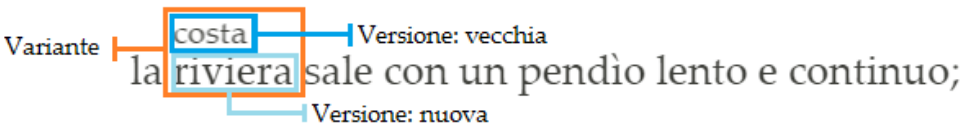

Figura 2.2 Visualizzazione variante: verticale

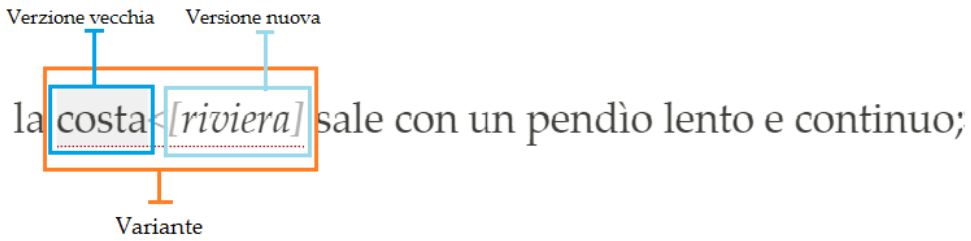

Figura 2.3 Visualizzazione variante: orizzontale

Come evidenziato dalla figura 2.2, nella modalità verticale vengono riportate le variazioni della nuova versione nell'interlinea, discostandosi dalla tradizione che riportava nel rigo la nuova versione e nell'interlinea la vecchia [IV14]. Con questa rappresentazione si sono volute emulare le modalità di correzione manuale dello studioso, che tipicamente annota le correzioni nell'interlinea sovrastante.

L'algoritmo implementato si è rivelato efficace e adatto allo scopo, evidenziando tuttavia alcune problematiche tecniche che hanno richiesto un intervento manuale. Tra di esse una delle principali è stata la gestione delle differenze nella punteggiatura, peraltro piuttosto frequenti nell'opera di Manzoni utilizzata nel prototipo. Nella cultura occidentale, infatti, la punteggiatura viene tipicamente attaccata alla parola precedente. Questo crea non pochi problemi all'algoritmo, poiché utilizzando la parole come entità atomica, identifica la punteggiatura come parte della parola. Il margine di errore è stato diminuito aggiungendo alle versioni degli spazi prima e dopo la punteggiatura, ma sono comunque presenti ancora numerose imprecisioni. Oltre a questi tipi d'errore potrebbero capitare casi in cui il diffing riporti errori più gravi e che richiedano quindi una modifica dei delta e la rigenerazione manuale del testo. Sono quindi stati introdotti l'opzione *unisci*, *separa* e la possibilità di modificare il testo.

Se viene reputato dal filologo che due varianti mostrate separatamente debbano essere invece un'unica variante, è possibile accorparle applicando una selezione che vada dalla prima alla seconda e agendo sul tasto unisci collocato nel pannello laterale *versione e stili*.

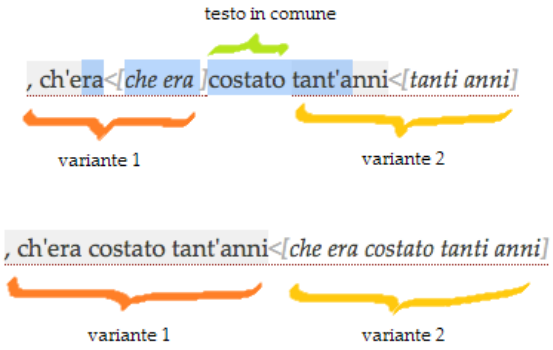

Figura 2.4 Unione di due varianti

L'operazione di separazione viene invece utilizzata per effettuare l'operazione inversa, ovvero per la divisione di una variante. Il primo caso d'uso si ha in presenza di due lezioni appartenenti alla stessa variante che condividono una parte di testo. In questo caso si seleziona arbitrariamente la parte in comune di una delle due lezioni e si agisce sul tasto separa.

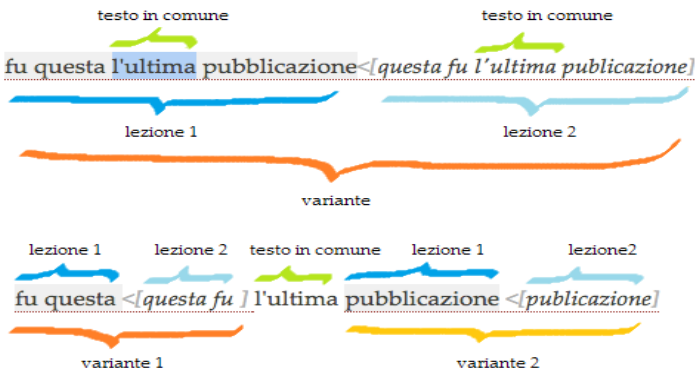

Figura 2.5 Separazione sulla base di testo condiviso

Il secondo caso d'uso si presenta nel momento in cui non ci sia una parte di testo in comune tra le due lezioni, ma si voglia comunque effettuare una divisione. Risulta necessario applicare una selezione che parta dal punto di separazione della prima lezione e arrivi nel punto di separazione della seconda. Agendo sul tasto separa si ottengono due varianti separate e contigue come mostrato dalla figura 2.6.

Figura 2.6 Separazione senza testo condiviso

Infine risulta possibile editare un testo effettuando un *doppio click* sulla parte desiderata. La modifica può riguardare sia il testo di una delle due lezioni che il contenuto di una variante. È anche possibile inserire una nuova variante tenendo premuto il tasto *alt* e cliccando due volte sul punto del testo in cui la si vuole aggiungere.

2.2.1 Critica di varianti

Dal punto di vista filologico la rappresentazione digitale del confronto dei testimoni porta a un superamento dell'edizione critica passando, come appena visto, da una visualizzazione sinottica delle varianti ad una *stratigrafica.* Questo tipo di rappresentazione risulta particolarmente adatta per la critica delle varianti, poiché evidenzia la dinamicità delle versioni e con essa l'evoluzione del pensiero dell'autore. Non esistendo algoritmi per la valutazione delle varianti, è stata realizzata un'interfaccia per la loro classificazione da parte del filologo. La barra laterale *versione e stili* permette di selezionare le categorie di ogni variante da un insieme predefinito. Il primo di questi insiemi è pensato per rappresentare le metodologie correttorie, ovvero le tipologie dei cambiamenti (inserimenti, cancellazione, spostamenti di parole…) slegati dal loro significato letterario. Questo insieme di correzioni è segnalato tramite la colorazione del testo oppure con l'aggiunzione di effetti tipografici come *bold* e *italic*. Il secondo insieme rappresenta invece le categorie correttorie, ovvero le motivazione letterarie dietro un cambiamento. Questo insieme di correzioni è evidenziato colorando lo sfondo della porzione di testo da segnalare. La natura dei due insieme e le categorie che

essi contengono sono strettamente legate all'opera. Come già menzionato, a partire dalla nuova versione PhiloEditor supporta insiemi e categorie diverse per ogni opera che viene inserita.

2.2.2 La visione grafica delle statistiche

La raccolta e la visualizzazione di informazioni riguardanti la frequenza di un certo tipo di modifica o anche soltanto la percentuale di testo variato tra due testimoni è indubbiamente una delle principali innovazioni del progetto PhiloEditor. La raccolta di statistiche, specie su testi molti lunghi, è infatti storicamente un grosso limite della filologia su carta. La raccolta di tali risultati senza l'utilizzo del supporto informatico comporterebbe un lavoro molto dispendioso a livello di tempo e un alto rischio di errore.

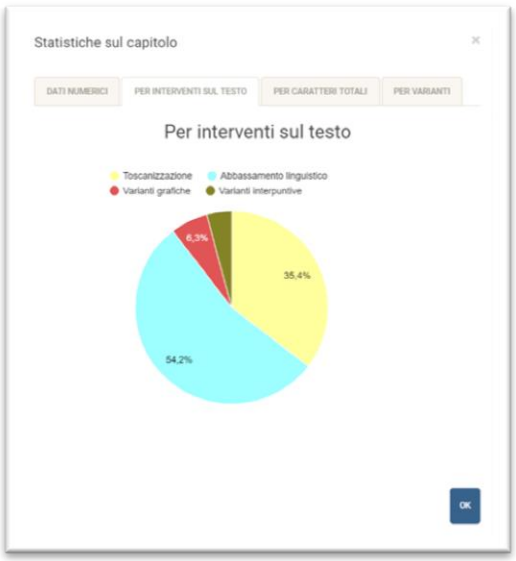

Figura 2.7 Esempio di visualizzazione statistiche per interventi sul testo

L'applicazione permette due tipi di visualizzazione: le statistiche su tutta l'opera o sul singolo capitolo. In entrambi i casi viene mostrata una finestra modale che visualizza i risultati suddivisi in dati numerici, per interventi sul testo, per caratteri totali, per caratteri della singola variante. Queste operazioni servono a stimare la frequenza di utilizzo delle categorie correttorie. Nella sezione *Dati numerici* viene mostrato in una tabella il numero di interventi in cui è stata utilizzata ogni categoria correttoria e il numero dei caratteri che ha coinvolto. La tabella contiene anche il numero di caratteri non modificati e il numero totale di caratteri e di interventi sul capitolo. Le altre sezioni mostrano le percentuali di testo segnalate per ogni categoria, facilitando la lettura dei dati attraverso grafici a torta o istogrammi.

2.3 L'edizione condivisa

Fino alla versione 2.0 PhiloEditor identificava il proprietario di un documento con il suo creatore, impedendo di estendere la proprietà (e conseguentemente i permessi di modifica) di un'edizione. L'idea dell'implementazione di una funzione di condivisione di permessi è nata dalla volontà di sfruttare i vantaggi di una applicazione web, che come è noto diminuisce se non annulla i problemi di distanza geografica, incrementando invece la cooperazione e permettendo la creazione di edizioni collaborative. La funzione implementata per gestire questa casistica è attivata cliccando sul tasto *condividi documento*. Se si è proprietari di un documento, è possibile condividere i permessi con un altro utente registrato, inserendo nell'apposito form il suo username. La gestione delle modifiche è implementata attraverso l'aggiunta al testo modificato di un attributo ***@resp***, che segnala il nome utente dell'utente responsabile.

2.4 La codifica dei testi in formato TEI

La codifica in formato TEI P5 è ormai una caratteristica imprescindibile delle edizioni per la conservazione dei dati elettronici e la loro interoperabilità. È stato quindi reputato necessario aggiungere nella nuova versione di PhiloEditor un sistema che ne garantisse l'esportazione. La codifica delle variazioni viene gestita attraverso il metodo *Parallel segmentation*, che è uno dei tre metodi proposti nel modulo *TEI Crtitical Apparatus*. La scelta è ricaduta sull'utilizzo di questo metodo piuttosto che sugli altri poiché risulta particolarmente adatto per la codifica di variazioni meno complesse che, così come avviene in PhiloEditor, non prevedano overlap di variazioni. Il *Parallel segmentation method* prevede una biforcazione del flusso del testo: ogni segmento di testo in cui è stato registrato un cambiamento è codificato mediante l'utilizzo dell'elemento ***<app>***, avente un attributo ***@type*** che assume il valore delle classi che caratterizzano la variante codificata. Al suo interno ogni lezione viene a sua vota codificata in un elemento ***<rdg>***, che può presentare tre possibili attributi: ***@resp***, ***@wit*** e ***@source***. All'intero dell'attributo ***@resp*** viene memorizzato il nome dell'utente che per ultimo ha modificato la variante. Se la variante non è mai stata modificata, ***@resp*** assume il valore *#void*. Gli attributi ***@wit*** e ***@source*** servono invece a indicare a quale testimone fa riferimento la versione. La descrizione dei testimoni

può essere fatta sia tramite l'utilizzo degli elementi ***<witness>*** che degli elementi ***<bibl>***. L'elemento ***<witness>*** contiene la descrizione del testimone; se il testimone è un testo edito allora può essere anche descritto nell'elemento ***<bibl>***.

L'esportazione avviene semplicemente cliccando sul pulsante *Esporta TEI*, che darà inizio al download del file sul computer. I metadati che popolano il file XML scaricato vengono raccolti in fase di salvataggio. I metadati richiesti nel form e che è stato ritenuto necessario memorizzare per una corretta codifica del documento XML sono i seguenti:

- *Nome autore dell'intervento*
- *Responsabilità dell'autore*
- *Luogo di pubblicazione*
- *Licenza per il riutilizzo dei dati*
- *Interventi nel documento digitale*

Il campo *autore dell'intervento* andrà a riempire il tag ***<publisher>*** all'interno del ***<publicationStmt>***, che contiene informazioni sulla pubblicazione. Il ***<pubPlace>*** viene invece settato al valore indicato come "Luogo di Pubblicazione", mentre la ***<date>*** non viene richiesta ed è automaticamente settata alla data attuale. La responsabilità d'autore raccoglie invece una descrizione della responsabilità intellettuale dell'utente e va a riempire il tag ***<resp>***, che è il primo figlio del ***<respStmt>***. Il nome del contributore all'interno del *respStmt* (indicato dal tag ***<name>***) non viene invece preso in input, ma viene compilato utilizzando il nome e il cognome dell'utente che effettua il salvataggio, così come l'attributo *xml:id* del ***<respStmt>*** stesso che conterrà l'username dell'utente. Il campo *Licenza per l'utilizzo dei dati* va a riempire il tag ***<licence>*** contenuto nell'elemento ***<availability>*** e indica, per l'appunto, la licenza d'uso da allegare al documento pubblicato (generalmente sotto forma di URL). Infine, gli *Interventi nel documento digitale* sono finalizzati a documentare la relazione fra il documento elettronico e la o le fonti da cui è derivato, e andranno a riempire l'***<encodingDesc>*** dell'XML.

Se il documento era privo di queste informazioni, i campi del form saranno mostrati all'utente vuoti. Se invece erano già presenti dei metadati relativi al documento, il form sarà mostrato già popolato e verrà data la possibilità all'utente di modificarli.

Capitolo 3

Dettagli implementativi

In questo capitolo vengono introdotte nel dettaglio le strutture dati e le funzioni utilizzate per la realizzazione dell'applicazione. Per ogni componente vengono inoltre descritte e giustificate nel dettaglio le scelte implementative effettuate e quanto esse si siano rivelate adatte al raggiungimento degli obiettivi preposti.

3.1 File e strutture dati

PhiloEditor 3.0 è un'applicazione web costituita da due entità logiche: un *client* e un *server*. Il client è stato scritto con la classica suite di linguaggi per front-end web: HTML per il layout, CSS per l'aspetto presentazionale (con il supporto della libreria Bootstrap 3.3.1) e Javascript/JQuery per le funzioni di scripting. Gli script server-side di PhiloEditor sono invece interamente scritti in PHP senza il supporto di alcuna libreria o software esterno. Il server si basa sulla piattaforma Apache 2.

3.1.1 File dell'applicazione

L'intero HTML dell'applicazione è contenuto nel file **index.html**, comprensivo anche dei vari template utilizzati. Il codice Javascript si trova nella cartella *js*. Tutte le funzioni implementate per l'applicazione si trovano nel file **main.js**, mentre le varie librerie utilizzate si trovano in vari file separati collocati nella medesima directory che verranno descritti nel dettaglio nel corso del capitolo. Il CSS è invece contenuto nel file **main.css** e collocato nella directory *css* contenente anche i file di Bootstrap.

Gli script server-side si trovano nella directory *php*. Per ogni servizio è stato implementato un file separato, in modo da mantenere chiara l'interfaccia di comunicazione tra client e server. Gli script implementati sono:

- **register.php** che gestisce la registrazione
- **login.php** e **logout.php** che gestiscono le operazioni di login e logout
- **changePwd.php** che gestisce il cambio password di un utente registrato

- **coowner.php** che gestisce la condivisione dei permessi di scrittura su un file
- **getFiles.php** che restituisce la lista dei documenti disponibili sul sistema
- **getOpera.php** che restituisce la lista delle opere presenti
- **getStats.php** che restituisce le statistiche su un'opera.

I file delle opere e dei relativi metadati si trovano nella directory *files*. Ogni opera ha una sottodirectory indipendente contenente i vari capitoli. Una cartella speciale chiamata *"00 – Metadata"* è presente nella directory di ogni opera e contiene tutte le informazioni aggiuntive e i metadati sui file che la costituiscono.
Il file contenente la lista degli utenti e i rispettivi dati è invece **users.json**, collocato nella directory data/hidden protetta da lettura e scrittura. La sua struttura e il suo ruolo nell'applicazione saranno descritti nel dettaglio all'interno del paragrafo 4.1, in cui sono descritte le funzionalità di accounting.

3.1.2 Le strutture dati

Non esistono strutture dati globali client-side se non la struttura *current*, che verrà descritta nel dettaglio nel paragrafo successivo. Le strutture globali del server sono invece memorizzate separatamente per ogni opera in formato JSON, e sono contenute nella directory *00 – Metadata*. Al suo interno sono presenti quattro file json.

- header.json
- infoDoc.josn
- style.json
- stats.json

header.json contiene header significativi sull'opera; al momento l'unico campo di questo tipo memorizzato è il *titolo* dell'opera. *infoDoc.json* contiene i dati di tutti i documenti che appartengono ad un'opera. Si presenta come un array di oggetti, che possono essere strutturati in due possibili modi. Uno serve a descrivere i documenti che contengono il testo originale dei testimoni ed ha questa forma:

```
{
   "id": "1",
   "order": "1827",
   "label": "Capitolo1",
```

```
        "path": "files\/01 - I promessi sposi\/01 -
                 Capitolo 1\/1.txt",
        "version": "1827"
    }
```

La seconda, invece, serve a descrive le edizioni create dagli utenti del sito. Qui il campo *version* della precedente struttura viene sostituito con *versions* che indica quali testimoni sono stati utilizzati nell'operazione di diffing. In aggiunta, si trova il campo *authors* che serve ad indicare l'username o gli username degli autori dell'edizione.

```
 {
       "id": "3",
       "order": "italia",
       "authors": ["italia"],
       "label": "Capitolo1",
       "versions": ["1827","1840"],
       "path": "files\/01 - I promessi sposi\/01 -
                Capitolo 1\/3.html"
 }
```

style.json contiene invece tutti gli stili relativi all'interfaccia grafica per la categorizzazione delle varianti: vengono memorizzate le macrocategorie in cui esse possono essere suddivise e, per ogni categoria, viene descritta la marcatura cromatica corrispondente:

```
{
"categories":{
           "label": "Categorie correttorie",
           "items":{
                 "toscanizzazione": {
                       "label": "Toscanizzazione",
                       "abbr":"Tosc.",
                       "className":"toscanizzazione",
                       "css":"background-color: #99ffff;"
                 },…
           }
     },
     "methodologies": {
           "label": "Metodologie correttorie",
           "items": {
                 "ordine": {
                       "label": "Ordine delle parole",
                       "className": "ordine",
                       "css": "color: #FF00FF;"
                 },…
           }
     }
}
```

L’ultima struttura presente è la struttura *stats.json* che contiene i dati utili per i calcolo delle statistiche, ovvero il numero di caratteri e varianti appartenenti a una categorie correttoria. Per ogni capitolo vengono memorizzati anche il numero di caratteri totali nel campo *total*, di caratteri non modificati nel campo *unmodified* e di varianti presenti nel campo *totalItems*.

```
{
    "01 - Capitolo 1": {
        "32": {
            "chars": {
                "Toscanizzazione": 139,
                "Abbassamento linguistico": 289,
                "Varianti grafiche": 22,
                "Varianti interpuntive": 29
            },
            "items": {
                "Toscanizzazione": 17,
                "Abbassamento linguistico": 26,
                "Varianti grafiche": 3,
                "Varianti interpuntive": 2
            },
            "total": 48199,
            "unmodified": 47720,
            "totalItems": 48
        },…
}
```

3.1.2.1 La struttura current

L’unica struttura dati globale di rilievo all’interno del client è la già menzionata struttura **current**. Essa contiene informazioni importanti riguardanti lo stato dell’applicazione, dell’utente e dei documenti caricati. La struttura è composta dai seguenti campi:

- **edit:** *true* se l’utente sta modificando il testo del documento, *false* altrimenti.
- **mode** che può assumere i valori *edit* e *view*: *edit* se l’utente è in modalità modifica, *view* se è in modalità vista.
- **user:** che contiene informazioni sull’ (eventuale) utente loggato
- **document** che contiene le informazioni sul documento aperto dall’utente
- **documents** che è un array contenente tutti i documenti di un opera. Nello specifico contiene n oggetti di tipo **document**

- **opera** che contiene le informazioni dell'opera a cui appartiene il documento aperto dall'utente
- **operas** che è un array contenente tutte le opere presenti nell'applicazione ed è formato quindi da n oggetti di tipo **opera**
- **stats** che contiene i dettagli statistici dell'opera
- **view** che contiene un vettore delle categorie e metodologie correttorie attualmente visibili.

3.2 Caricamento documenti

L'operazione di caricamento si suddivide in due fasi. In primo luogo viene formata, facendo inferenza sul nome delle directory contenenti le opere, la lista di documenti che andrà a occupare la parte in alto a sinistra del layout di PhiloEditor. Successivamente viene effettuata la procedura di caricamento vero e proprio, che verrà descritta nel dettaglio discriminando il caricamento di un confronto dal caricamento di un'edizione.

3.2.1 Caricamento delle lista documenti

Il caricamento dei documenti è la prima attività che avviene all'avvio di PhiloEditor. Dopo che la pagina è stata caricata (`$(document).ready()`) il sistema effettua una chiama alla funzione **loadOpera()** che si occupa di recuperare la lista delle opere presenti sul sistema effettuando una chiama AJAX con metodo GET al servizio **getOpera.php**. Il servizio restituisce un array di strutture contenenti informazioni sulle opere necessarie al sistema. L'implementazione di queste strutture ha attraversato vari stadi nel suo sviluppo. Un primo approccio, infatti, prevedeva che il sistema supportasse categorie e metodologie "anonime" e numerate. Esistevano cioè quattro classi per le metodologie (met_1 … met_4) e sei per le categorie (cat_1 … cat_6), ed ogni opera istanziava le varianti come necessario (per I promessi Sposi met_1 = "*toscanizzazione*"). Tale implementazione presentava il vantaggio di garantire un maggiore modularità del codice, poiché le regole erano fisse e il database doveva solo memorizzare il collegamento tra i nomi delle classi CSS e i nomi delle varianti. Tuttavia essa presentava due importanti svantaggi:

- L’ipertesto dell’opera (e di conseguenza il documento XML esportato) risultavano privi di semantica, non possedendo informazioni significative circa la natura delle varianti
- Il CSS associato alle varianti era fisso e quindi non dava la possibilità di personalizzare lo stile cromatico e tipografico dei tipi di varianti da un’opera all’altra.

E’ stato dunque deciso di includere esplicitamente le regole CSS da utilizzare nelle varie opere. La struttura risultante ha tre campi:

- **header** che contiene gli header dell’opera. Al momento è presente soltanto un campo *title* che indica il titolo dell’opera

- **path** che indica l’URI relativo della directory contenente i vari documenti che compongono l’opera

- **style** che contiene le informazioni sul tipo di varianti da caricare e le relative regole CSS.

Una volta caricate le informazioni sulle opere, viene chiamata la funzione **loadFileList()**. Essa effettua una chiamata AJAX con metodo GET al servizio **getFiles.php** che restituisce un JSON contenente la rappresentazione delle opere presenti del sistema e della loro posizione.

Il servizio costruisce il JSON analizzando ricorsivamente la directory *files* e costruendo una struttura ad albero. Le informazioni sui singoli documenti sono recuperate dal file **infoDoc.json**. L’algoritmo utilizzato per la creazione della struttura ad albero, che andrà a rappresentare la lista documenti visualizzata sul sito, parte dalla root analizzando ogni elemento:

- se l’elemento è una directory, esso rappresenta una suddivisione intermedia della lista di documenti (es. un’opera, un capitolo…). In tal caso l’algoritmo costruisce un nodo intermedio avente come campi *order* che indica l’ordine in cui l’elemento deve essere parsato rispetto ai suoi fratelli, *label* che indica come quell’entità viene visualizzata nella lista degli elementi, *path* che indica l’URI

relativo dell'entità e *content* che è una array di strutture ottenuto dalle chiamate ricorsive sugli elementi contenuti nella directory.

- Se l'elemento è un file, esso rappresenta un documento dell'opera. In tal caso l'algoritmo prenderà dal file **infoDoc.json** la struttura corrispondente, avente come id il nome del file esaminato e la inserirà come foglia dell'albero.

Una volta ricevuta la lista di documenti così costituita, la funzione **loadFilelist()** chiama la funzione **showFileList()** che ha il compito di costruire la lista di file visualizzata a sinistra della pagina. La prima operazione effettuata è quella di popolare l'array *current.documents* con i documenti caricati. Successivamente la lista è costruita con degli elementi **`<ul>`** annidati per le opere e per i capitoli. Le edizioni degli utenti sono invece collocate in tag **`<li>`** e associate tramite un listener "click" alla funzione **load(i)** dove *i* è l'indice dell'edizione nel vettore *documents*. La gestione dei testimoni è invece più complessa . La showFileList raccoglie tutti i testimoni presenti per ogni capitolo. Dopodiché li raggruppa a due a due creando degli **`<li>`** di confronto. Se sono dunque presenti `n` testimoni ci saranno un totale di `n*(n-1)` **`<li>`** nella forma *confronta versione_i e versione_j*, ciascuno dei quali associato tramite un listener click alla funzione **compare()**.

3.2.2 Caricamento documenti e diffing

Al termine della funzione showFileList tutti i documenti sono caricati nella lista a sinistra e collegati all'opportuno listener. Quando l'utente clicca su uno dei documenti, il sistema lo visualizza nell'area centrale. La visualizzazione delle edizioni create dagli utenti è molto semplice: la funzione **load()** recupera l'HTML dell'edizione, carica il CSS e il contenuto del pannello versioni e stili relativo all'opera desiderata, aggiorna le strutture dati di *current* e mostra l'HTML nel div *#file*. Il caricamento di un confronto tra due testimoni risulta invece più complesso, poiché i testimoni confrontati non sono salvati in un file, ma generati dinamicamente dal client. Il confronto tra i due file avviene nella funzione **compare()**. La prima operazione effettuata dalla funzione è, banalmente, il caricamento dei due file da confrontare. Successivamente viene passato il controllo alla funzione **diffAndShow()** che si occupa del diffing vero e proprio.

Per questa operazione PhiloEditor si appoggia ad una libreria Javascript esterna. Nel corso della fase implementativa (avvenuta già durante la realizzazione di PE 2.0), sono state testate approfonditamente due librerie per lo scopo, *Javascript diff algorithm* e *wikEd diff* [DIV14]. Alla fine è stata scelta la libreria wikEd diff[12], ritenuta più precisa ed affidabile. Il metodo principale della libreria è **WikEdDiffTool**(), che prende in input due testi da confrontare. Il metodo restituisce un HTML così formato: le parti di testo in comune sono prive di markup mentre le differenze sono racchiuse in elementi span. Al primo viene attribuita la classe *oldVersion*, al secondo la *newVersion*. Questo perché il testo passato come primo parametro al metodo WikEdDiffTool viene preso come testo base e confrontato con il secondo. Attraverso l'utilizzo della funzione **fixVersion()** viene controllato che il primo parametro passato sia quello relativo alla versione più vecchia.

Dopo il diffing viene chiamata la funzione **fixDiff()** che effettua alcune modifiche all'HTML restituito dalla WikEdDiffTool, quali:

- Rimozione di interleaving indesiderati tra *newVersion* e *oldVersion*
- Aggiunta di uno span con classe *replace* che racchiuda ogni variazione
- Aggiunta delle classe *insert* e *delete* rispettivamente nel caso di inserimenti o rimozioni
- Inclusione di ogni paragrafo in un tag **`<p>`**

Infine, analogamente alla funzione load, viene caricato il CSS dell'opera, vengono aggiornati i campi della struttura current e il risultato del diffing viene mostrato nel div con id *#file* .

3.2.3 Proprietà e condivisione dei documenti

Novità di PhiloEditor 3.0 rispetto al predecessore è la possibilità di gestire documenti in comproprietà con altri utenti. Ogni documento possiede infatti un campo *authors*, contenente la lista degli utenti che hanno diritto di scrittura su quel documento. Al momento della creazione del file, il vettore *authors* conterrà unicamente l'username del creatore. Egli potrà poi decidere di condividere la proprietà del documento utilizzando la funzione **coowner()** del pannello utente.

[12] Pagina ufficiale del progetto: http://cacycle.altervista.org/wikEd-diff-tool.html

Tale funzione invia una richiesta GET al servizio **coowner.php** specificando l'id del documento e l'username del nuovo proprietario. In caso di successo il vettore viene aggiornato includendo il nuovo proprietario. Gli errori previsti per questa operazione sono: "Utente non esistente" (no_user), "Utente già proprietario" (already_user) e "Operazione non permessa" (no_owner). Per tenere traccia del contributo di ciascun proprietario al documento, ad ogni variante viene aggiunto un attributo `@resp` contenente l'username di chi l'ha segnalata.

3.3 Modifica del documento

La modifica di documenti rientra tra le sezioni del sito accessibili soltanto agli utenti registrati. A questa categoria appartengono le operazioni di segnalazione della tipologia delle varianti, le operazioni di modifica delle varianti (tramite le funzioni **unisci** e **separa**) e le operazioni di modifica del testo stesso. In questa sezione verrà trattata nel dettaglio l'implementazione di ciascuna di esse.

3.3.1 Segnalazione di tipologie di varianti

L'implementazione della segnalazione di tipologie di varianti è stata ampiamente modificata in PhiloEditor 3.0, per gestire le numerose casistiche di conflitto che possono verificarsi tra le segnalazioni. Come già accennato nei capitoli precedenti, ogni variante può avere una categoria correttoria e una metodologia correttoria; metodologie e categorie non possono coesistere tra di loro e la segnalazione di una tipologia già presente risulta nella cancellazione della stessa. Per gestire tutte queste casistiche sono state implementate due funzioni **addCategoria()** e **addMetodologia()**. Le due funzioni operano in maniera simmetrica: controllano che una variante di quel tipo non sia già presente e la aggiungono, attribuendo allo span replace la classe corrispondente al tipo di marcatura che si vuole segnalare. Se la categoria/metodologia era già presente, l'attributo viene rimosso. Per determinare il gruppo di appartenenza delle tipologie presenti, le due funzioni accedono ai campi della struttura *current.opera*, in cui è specificato quali classi rappresentano delle categorie e quali rappresentano delle metodologie.

3.3.2 Editing

I meccanismi di editing possono essere attivati da due comandi: il "doppio click" del mouse o la combinazione "alt+click" sul documento. Il primo comando permette di agire sul testo aggiungendo, modificando o rimuovendo dei frammenti. In questa modalità le parti di testo senza variazioni vengono racchiuse in tag ****** con classe *nochange* e il tag contente il punto selezionato diventerà un area di editing. Tale operazione viene gestita dalla funzione **beginEditing(),** che è attivata dall'evento *dbclick* sullo span a cui verrà attribuita la classe *editing*. Quando le modifiche sono concluse, cliccando in un qualunque punto della pagina esterno all'area di editing, viene chiamata la funzione **endEditing()** che rimuove la classe editing mettendo fine all'operazione di modifica.

Il comando "alt+click", invece, permette di aggiungere una nuova variante nel punto selezionato. L'operazione è gestista dalla funzione **startEditing()** che crea un nuovo span con classe replace contenente una newVersion e una oldVersion, entrambe vuote e con classe editing. A questo punto la gestione è analoga al caso descritto precedentemente: l'utente può inserire il contenuto delle nuove versioni e terminare l'inserimento cliccando in qualunque punto della pagina.

3.3.3 Modifica varianti

L'operazione di modifica varianti risulta essere la più complessa tra le operazione di modifica sopra descritte poiché va a cambiare la struttura HTML del testo e incorre in numerosi possibili conflitti. Le funzioni che gestiscono questo tipo di operazioni sono la **join()** e la **split()** che, come si è spesso ripetuto, sono concepite per effettuare operazioni inverse.

La funzione join viene attivata agendo sul pulsante *unisci* dopo aver effettuato una opportuna selezione sul testo, ovvero può essere utilizzata nei casi in cui:

- si vogliano accorpare due varanti in un'unica,
- si voglia eliminare la versione di uno dei due testimoni perché ad esempio ritenuta un errore (la versione mantenuta è quella in cui finisce la selezione, quindi se la selezione parte dalla versione a sinistra per concludersi in quella di destra, quest'ultima sarà quella mantenuta).

La join inizia recuperando la selezione attuale e controllando che entrambi i suoi estremi si trovino all'interno di un tag *replace*. In tal caso la funzione controlla che il tag che li contiene sia lo stesso. In caso affermativo, ci troviamo nel secondo caso d'uso: lo *span.replace* viene rimosso assieme all'ultima versione selezionata. In caso contrario vengono memorizzate in due variabili *oldV* e *newV* il contenuto rispettivamente delle due *oldVersion* e delle *newVersion* insieme al testo in comune che le separava. A questo punto le due varianti e il testo intermedio vengono rimossi e sostituiti da un nuovo replace, la cui *oldVersion* avrà il contenuto di *oldV* e la newVersion avrà il contenuto di *newV*.
La funzione split viene attivata dal click sul pulsante separa, e può essere utilizzata nei seguenti casi:

- se si vuole estrapolare dalla variante un parte di testo condiviso tra le due versioni che la compongono,
- se si vuole creare una variante partendo da una parte di testo in comune,
- se si vuole dividere una variante.

Nel primo caso, una volta selezionato il testo in comune tra le due versioni esso viene salvato in una variabile *t*. Il contenuto della newVersion è della oldVersion viene invece salvato nelle due variabili *a1* e *a2* e la classe della variante da separare viene salvata nella variabile *c*. Inoltre viene anche creato un template *newText* che rappresenta uno span replace con il relativo contenuto, parametrico rispetto alla classe e ai contenuti delle due versioni.
A questo punto la funzione sostituisce il vecchio span con tre nuovi span:

- uno span replace contenente il frammento di oldVersion e newVersion precedente alla parte comune, ottenuto istanziando il template newText con il contenuto di a1 e a2 e la classe contenuta in c,
- uno span nochange contenente la parte in comune,
- uno span replace contenente il frammento di oldVersion e newVersion successivo alla parte comune, ottenuto istanziando il template newText con il contenuto di a1 e a2 e la classe contenuta in c

Se si vuole invece creare una variante a partite da una parte di testo condiviso , il testo selezionato viene racchiuso all'interno di un tag replace e il suo contenuto viene copiato all'interno dei tag oldVersion e newVersion. Idealmente questa

situazione è stata pensata per gestire la casistica di una variante che non risulti essere presente dove invece è ritenuta necessaria; in tal caso si effettua questo tipo di separazione e si procede alla modifica di una delle due versioni.
L'ultimo caso gestito è la divisione di una variante in cui la selezione parta da un punto della prima versione per terminare in un punto della seconda versione. In questo caso viene creata una struttura *nodes* contenente, sia per la newVersion che per la oldVersion, il nodo in cui sono contenute, il loro contenuto e l'offset della selezione. Successivamente la funzione genera due span:

- uno span con classe replace contenente il frammento di newVersion non selezionato e il frammento di oldVersion selezionato, ottenuto istanziando il template newText con i campi della struttura nodes,
- uno span con classe replace contenente il frammento di oldVersion non selezionato e il frammento di newVersion selezionato, ottenuto istanziando il template newText con i campi della struttura nodes.

3.4 Esportazione in TEI

L'esportazione del TEI associato ad un documento viene effettuata interamente via client: viene generato un XSL a partire da un template, successivamente il documento viene parsato come XML e convertito in TEI utilizzando un XSLT processor.

3.4.1 Generazione del foglio XSL

Il documento XSL per l'esportazione in TEI è stato costruito tenendo conto dei parametri e dei metadati desiderati per l'XML finale e della struttura dell'HTML generato dalla funzione fixDiff**.** Ogni opera ha un file XSL chiamato TEI.xsl collocato nella sottodirectory dei metadati (00 - Metadata). Per garantire una buona flessibilità dell'applicazione il file memorizzato è incompleto: i campi degli header sono infatti dei parametri simili a quelli utilizzati nei template HTML e vengono riempiti dal client prima che la trasformazione avvenga. Il file è suddiviso in due parti: una sezione ***<teiHeader>*** contenente gli header TEI per quella specifica opera, e una sezione ***<text>*** che contiene il corpo dell'edizione.
La generazione dell'XSL definitivo avviene all'interno della procedura **esportaTEI()**, che si attiva quando l'utente clicca sull'omonimo pannello della

navbar. La funzione sostituisce i parametri presenti negli header del file XSL generando dinamicamente il contenuto in base ai metadati disponibili per il documento da esportare, memorizzati nel campo *tei* della struttura current.document. La sezione ***<teiHeader>*** del documento XSL è strutturata come nella figura 3.1.

```
<teiHeader>
   <fileDesc>
      <titleStmt>
         <title>Titolo dell'opera</title>
         <author>Autore dell'opera</author>
      </titleStmt>
      <editionStmt>
         <edition>Edizione collaborativa</edition>
         $resp
      </editionStmt>
      <publicationStmt>
         <publisher>$publisher</publisher>
         <pubPlace>$pubplace</pubPlace>
         <date>$date</date>
         <availability>
             <licence>$URL</licence>
         </availability>
      </publicationStmt>
      <sourceDesc>
        <listWit>
        <!—lista dei testimoni!—>
        </listWit>
      </sourceDesc>
   </fileDesc>
   <encodingDesc>
      <p>$encoding</p>
   </encodingDesc>
   <revisionDesc>
       $change
   </revisionDesc>
</teiHeader>
```

Figura 3.1 Struttura dell'header TEI

In alcuni casi il tag ***<listWit>*** potrebbe essere affiancato o sostituito dal tag ***<bibl>***, che descrive una bibliografia completa dei testimoni. In entrambi i casi tali informazioni sono già codificate nel template e non vengono inserite dinamicamente.

La funzione esportaTEI opera le seguenti sostituzioni:

- Il parametro ***$resp*** viene sostituito con un elenco di *respStmt* ottenuti

riempiendo il template *resp* con le informazioni del campo *document.tei.editionStmts*, nel quale sono memorizzati i contributi dei vari utenti,

- I parametri ***$publisher***, ***$pubplace***, ***$date*** e ***$URL*** vengono riempiti con le informazioni contenute all'interno della struttura *document.tei.publicationStmt*,
- Il parametro ***$encoding*** è sostituito dall'*encoding* memorizzato in *document.tei.encodingDesc*,
- Il parametro ***$change*** viene sostituito con un elenco di tag ***<change>*** ottenuti riempiendo il template *change* con le informazioni del campo *document.tei.revisionDesc*, nel quale sono memorizzati i timestamp dei cambiamenti effettuati dai vari utenti.

La sezione text dell'XSL contiene le trasformazioni da effettuare sull'HTML dell'opera. Sostanzialmente, per ogni paragrafo dell'HTML originale, viene creato un nuovo tag ***<p>*** contenente:

- Gli span con classe nochange dell'HTML, così com'erano.
- Per ogni span con classe replace, un tag ***<app>*** avente un attributo ***@type*** contenente il tipo di variante in considerazione (attributo class dell'HTML originale). All'interno di ogni ***<app>*** ci saranno due ***<rdg>*** relativi alla oldVersion e alla newVersion della variazione, aventi un attributo ***@resp*** contenente l'username dell'utente responsabile di quella variazione (attributo data-responsible dell'HTML originale). Se assente viene attribuito il valore di default *#void*. In più possono essere presenti i due attributi ***@source*** e ***@wit*** che collegato la lezione al testimone a cui appartiene.

3.4.2 Processing dell' XML e salvataggio file

La conversione dell'HTML di un documento in XML non ha richiesto operazioni particolarmente complesse: è stato sufficiente importare l'HTML contenuto nel div con id *#file*, wrappare il contenuto in un tag ***<home>*** (i documenti XML per essere validi non possono avere radici multiple) e aggiungere il tag ***<?xml>*** iniziale, specificando l'encoding e definendo l'entità * .* L'XML così formato è parsato con un oggetto *DOMParser* e salvato nella variabile *xmlDoc*.

Analogamente l'XSL descritto nel paragrafo precedente viene parsato e salvato nella variabile *xslDoc*. La trasformazione a questo punto avviene utilizzando il processore XSLT *Saxon-CE*[13]. La scelta è ricaduta su Saxon piuttosto che sul processore incluso di default nelle librerie di Javascript poiché Saxon supporta lo standard XSLT2.0, necessario all'utilizzo del metodo *replace*. Il processore Saxon è ottenuto istanziando un oggetto di tipo *newXSLT20Processor*. A questo punto viene importato l'XSL utilizzando il metodo *importStylesheet*, e viene trasformato l'XML con il metodo *transformToDocument*. L'XML finale viene poi codificato in una stringa e salvato nel file *Capitolo_n_Edizone_di_authors.xml*, dove *n* è il numero del capitolo e *authors* è la lista degli autori. Il salvataggio è effettuato utilizzando la classe *Blob* della File API di Javascript.

[13] Home page della libreria: http://www.saxonica.com/ce/index.xml

Capitolo 4

Valutazioni

Il passaggio da PhiloEditor 2.0 a PhiloEditor 3.0 ha riguardato, oltre all'implementazione di novità significative come l'esportazione in TEI e la *demanzonizzazione*, anche miglioramenti grafici e funzionali ad alcuni servizi già presenti, una serie di modifiche all'interfaccia utente e l'aggiunta di alcune funzionalità di base come un sistema di accounting.

Le dieci euristiche di Nielsen [Nie95] descrivono punti fondamentali che una buona GUI dovrebbe prendere come linee guida. Molti di questi punti si focalizzano sull'importanza di rendere l'utente consapevole di ciò che lo circonda e dell'ambiente in cui si muove. PhiloEditor 3.0, attraverso l'utilizzo di appositi feedback, guida l'utente verso il corretto utilizzo dell'applicazione e cerca di proporre una GUI pulita e di facile apprendimento, presentandosi con un layout minimalista, un'organizzazione logica dei contenuti e una comunicazione tra utente e applicazione basata sul modello di selezione a menu.

Parte del lavoro che ha portato a questi miglioramenti ha richiesto un refactoring di grandi parti di codice. Di seguito verranno descritte le funzionalità aggiunte e verranno riportati e commentati i risultati di correttezza delle funzioni principali.

4.1 Funzionalità generali

In PhiloEditor 2.0 mancavano un sistema di registrazione degli utenti e una modalità di salvataggio più efficace. Queste funzioni vengono descritte in separata sede rispetto alle altre poiché non sono caratterizzanti per lo scopo ultimo del progetto, ma sono necessarie per rendere l'applicazione utilizzabile.

4.1.1 Accounting

PhiloEditor 3.0 ha un sistema di autenticazione per regolare i permessi di determinate azioni e per tenere traccia delle responsabilità di ciascun contributo, oltre che della proprietà dei vari documenti. Il database degli utenti è memorizzato

nel file **users.json**, collocato a sua volta nella directory data/hidden. Il file è un array di strutture di questo tipo:

```
{
      "nome":"mario",
      "pwd": "pwd",
      "showAs":"Mario Rossi",
      "gender":"m"
}
```

Si è sperimentato anche l'utilizzo di *mysql* per la gestione degli utenti, ma la complessità derivante dall'utilizzo di una applicazione esterna non bilanciava un aumento dell'efficienza, viste le dimensioni ridotte del bacino d'utenza dell'applicazione. Un'ulteriore motivazione che ha portato all'abbandono di questa implementazione è stata la scelta di non appoggiarsi su software esterni.

4.1.1.1 Modalità di registrazione

L'attività di registrazione inizia quando l'utente clicca sull'apposito bottone *Sign Up* del modal di login. Dopo che il form è stato compilato e consegnato viene chiamata la funzione **signUp()** che effettua le seguenti operazioni:

- recupera i dati dal form e li salva in una variabile
- controlla che i campi siano compilati in maniera non banale
- controlla che la conferma password corrisponda alla password inserita
- invia i dati di registrazione al servizio **register.php** tramite una chiamata AJAX con metodo POST.
- In caso di successo notifica l'utente dell'avvenuta registrazione
- In caso di insuccesso notifica all'utente il tipo di errore. Gli errori supportati sono "user_exists" (utente già presente nel database) e "server non disponibile".

Tutti gli errori sia del client che del server sono notificati tramite appositi alert di bootstrap generati con la funzione **error().** In caso di campi non compilati del form, essi vengono colorati di rosso sfruttando i metodi di *form validation* di Bootstrap. Il servizio register.php recupera i dati della richiesta e controlla che lo username inserito non sia già utilizzato da un altro utente. In tal caso aggiunge i dati del nuovo utente al file **users.json** e invia all'amministratore del sito un'email di notifica della registrazione con nome, cognome e username del nuovo utente.

Infine, per notificare il successo delle operazione di registrazione invia come risposta al client un semplice oggetto JSON`{"result":"ok"}`. Nel caso in cui invece lo username non sia disponibile, al campo result dell'oggetto JSON viene assegnato il valore "user_exists".

La password scelta in fase di registrazione potrà eventualmente essere modificata con l'opzione *cambia password* presente nel pannello dell'utente registrato.

4.1.1.2 Modalità di autenticazione

Al click sulla bottone *login* presente nella finestra modale per l'accesso, viene chiamata la funzione **login()**. In questo caso la funzione raccoglie i dati del form senza fare nessun controllo sulla presenza di campi vuoti poiché tale caso è gestito dall'attributo *required* di HTML5. Per ragioni di sicurezza la password viene inviata al server tramite Hash MD5. L'hash è stato fatto utilizzato un'implementazione di MD5 in Javascript ad opera di BlueImp[14] . I dati vengono inviati al servizio **login.php** attraverso una chiamata AJAX con metodo POST. In caso di successo ({"result":"ok"}) viene chiamata la funzione **fixLoginMenu()**, che recupera i dati dal cookie di login e mostra il messaggio di benvenuto e il relativo pannello utente, aggiornando al contempo la struttura *current.user*. In caso di errore viene notificata all'utente la natura dell'errore tramite un messaggio che compare nel form. Gli errori possibili sono "Dati di login errati" (wrong_login) e "Server non disponibile". Il server controlla i dati di login inviati, confrontando l'hash mandato dall'utente con l'hash della password nel database calcolato con la funzione **md5()** di PHP. In caso di successo viene inviato un semplice JSON nella forma {"result":"ok"}. In caso contrario viene notificato l'opportuno errore come illustrato nel paragrafo precedente.

4.1.2 Salvataggio documenti

In PhiloEditor 2.0 era possibile salvare il proprio lavoro soltanto creandone una nuova copia. Per diminuire la quantità di edizioni inutili che potevano così venirsi a creare, si è deciso di aggiungere una nuova modalità di salvataggio che permettesse di sovrascrivere un documento. In questo modo se l'utente ha necessità di apportare modiche a un lavoro precedentemente salvato, agendo sul

[14] Repository del progetto: https://github.com/blueimp/JavaScript-MD5

pulsante *salva* ha la possibilità di sovrascriverlo senza creare una nuova copia. A livello implementativo tale operazione attiva la funzione **save()** che, come prima azione, richiederà all'utente tramite un form i metadati dell'opera. Queste informazioni, come già detto, servono per l'esportazione in TEI. Il testo da salvare viene poi inviato insieme alle informazioni necessarie al salvataggio al servizio **save.php** tramite una chiamata AJAX con metodo POST. Il server accede al file da scrivere e sostituisce il contenuto con quello nuovo inviato dal client. In caso di necessità vengono aggiornati anche gli header e le statistiche sul file. Nel caso in cui sia stata selezionata, invece, l'opzione *salva come nuovo*, il server crea un nuovo file nella directory del file originale, il cui id sarà uguale a *maxId*+1 dove maxId è il numero totale di documenti presenti per l'opera. In questo caso viene anche aggiunta un nuova struttura all'array infoDoc.json i cui campi sono ottenuti facendo inferenza su quelli del documento originale.

4.2 Testing e correttezza del codice

PhiloEditor 2.0 presentava numerosi casi d'errore che hanno richiesto il refactoring di diverse parti di codice. In questa sezione vengono descritti brevemente i bug principali riscontrati e le scelte che hanno portato alla loro risoluzione, dando al contempo una valutazione complessiva della correttezza del codice. Nel corso dell'attività di debugging le procedure di testing della correttezza sono state facilitate dal tool **QUnit**[15], una popolare libreria JavaScript per l'*unit testing*. La prima attività di debug ha riguardato le strutture dati e le variabili globali, che presentavano spesso casi di ridondanza o ambiguità. Casistica particolarmente significativa da questo punto di vista è stata la rifattorizzazione dei meccanismi di selezione del testo. Nella vecchia versione, infatti, il risultato della selezione (ottenuta tramite la funzione **getSelection()**) era salvato in una variabile globale *sel*. Tale implementazione, oltre a rappresentare un caso di ridondanza dell'informazione, risultava problematica e dava vita a numerose *race condition* dovute ai continui accessi e aggiornamenti paralleli da parte di diverse funzioni. La variabile *sel* rappresentava il tipico caso di *lava flow*[16], una informazione ridondante la cui rimozione comprometteva il corretto funzionamento di diverse attività. L'errore che si presentava maggiormente era la

[15] https://qunitjs.com/
[16] https://en.wikipedia.org/wiki/Anti-pattern#Lava flow

mancanza d'aggiornamento della variabile e la conseguente assenza di un *anchorNode.* Il problema è stato risolto rimuovendo tutti i riferimenti alla variabile e calcolando la getSelection quando necessario.

La seconda parte delle attività di debug ha riguardato la rifattorizzazione di alcune funzioni di editing. Per la maggior parte si è trattato di piccole modifiche strutturali o della risoluzione di semplici imprecisioni grafiche. Un caso interessante tuttavia è stato quella della funzione **join()**, che gestisce l'unione di due varianti. Nella precedente versione del software, infatti, tale funzione presentava due gravi bug: le due versioni della variante venivano invertite e il testo in comune veniva eliminato. La figura mostra il fallimento di uno *unit test* a causa di queste problematiche.

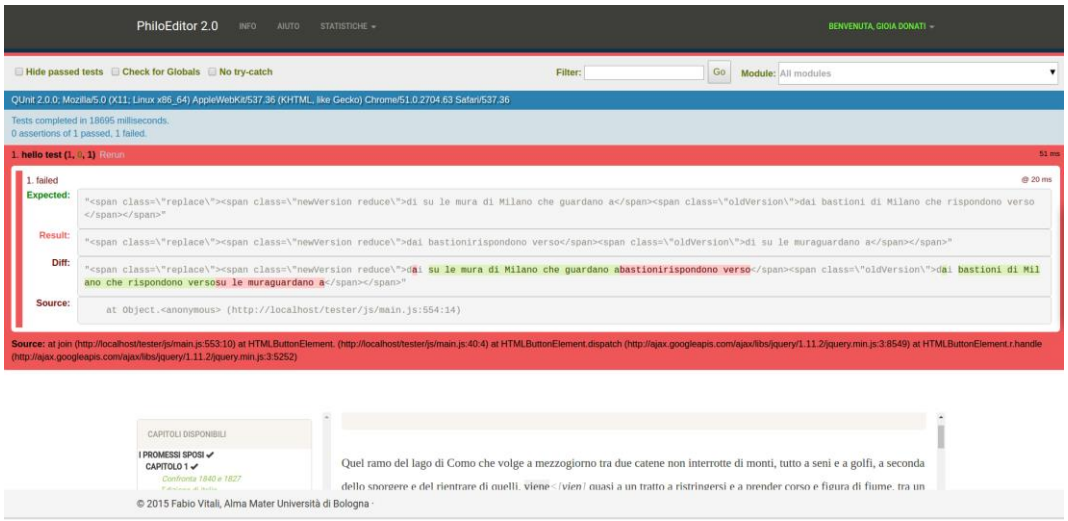

Figura 4.1 PhiloEditor 2.0: unit test della funzione *join*

La soluzione ha richiesto una riscrittura dell'intera procedura. La nuova funzione non prende più in input la struttura della selezione, che viene memorizzata al suo interno al momento della chiamata. Sono stati aggiunti dei nuovi casi d'uso. Ora è possibile utilizzare la join anche per eliminare una variante. Se la selezione inizia nella versione destra e finisce nella versione di sinistra, quella di destra viene cancellata e rimane solo quella di sinistra, mentre per mantenere quella di sinistra la selezione deve essere fatta nella direzione opposta. Vengono distinti due casi: il caso in cui la selezione coinvolga un'unica variante e il caso in cui ne coinvolga più di una. Il primo caso è quello appena descritto che gestisce la cancellazione di una delle due versioni, mentre il secondo è quello che effettua la vera operazione di *join* intesa come operazione inversa della *split* ed è più complicato. Se la

selezione rientra in quest'ultima casistica, la funzione memorizza le strutture DOM degli *span.replace* coinvolti nella selezione. Viene creato un nuovo *span* con classe *replace*, la cui newVersion contiene le newVersion delle varianti selezionate (nell'ordine in cui compaiono) più il testo in comune che le separa, e la cui oldVersion contiene le oldVersion delle varianti selezionate più il testo in comune che le separa. Gli span selezionati vengono infine eliminati e sostituiti con il nuovo span creato che rappresenta per l'appunto *l'unione* delle versioni selezionate dall'utente. Nella figura seguente sono mostrati i risultati dello unit testing con la nuova versione della join: come evidenziato dalla console di QUnit la funzione ha rispettato il comportamento richiesto in tutti i casi testati.

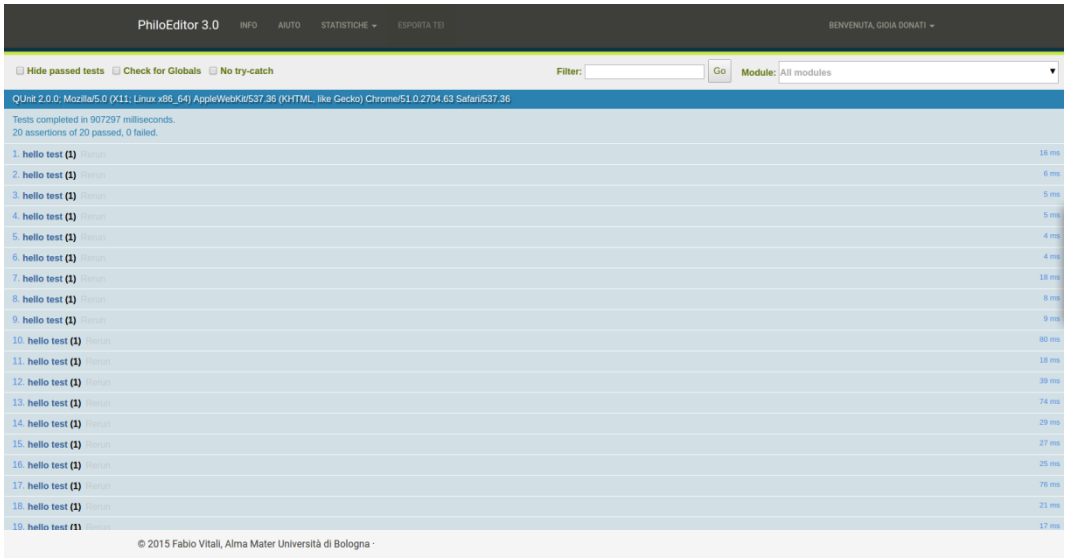

Figura 4.2 PhiloEditor 3.0: unit test della funzione *join*

Infine l'attività di debug si è conclusa con l'aggiunta di opportune segnalazioni d'errore sia nel client, attraverso opportuni alert di Bootstrap, che nel server con l'aggiunta di risposte informative nelle principali casistiche d'errore.

Conclusioni

L’applicativo PhiloEditor 3.0 si propone come un utile strumento per la ricerca filologica e la critica delle varianti. Il progetto è sotto licenza open-source GPL 3.0 e il codice sorgente può essere liberamente visualizzato e modificato all’indirizzo: https://github.com/donatigioia/philoeditor-3.0. Rispetto alla versione 2.0 risulta essere più stabile e presenta alcune funzionalità di base prima non presenti che migliorano l'esperienza utente e l'affidabilità del sistema. Rispetto agli applicativi attualmente disponibili, PhiloEditor 3.0 risulta essere l’unico che permetta la consultazione e lo studio di più opere e che mette a disposizione degli strumenti interpretativi specifici da opera a opera. Inoltre, la nuova possibilità di esportare i documenti in formato TEI colma un'importante lacuna, andando ad aggiungere all'applicazione una caratteristica ormai ritenuta di fondamentale importanza nel settore e presente in quasi tutti gli altri applicativi web di questo genere.

Allo stato attuale non risulta ancora caricata l’intera opera di Collodi, perché la fase di digitalizzazione dei capitoli rimanenti è ancora in corso d’opera. Tuttavia, la struttura del sistema è modificata, come è stato descritto nel corso della dissertazione, in modo da incorporare correttamente nuove risorse. I test effettuati sugli attuali capitoli caricati sul sistema hanno avuto riscontri positivi e un ottimo banco di prova per l’applicazione potrebbe essere il suo utilizzo da parte di studenti di filologia per esercitarsi con l’interpretazione delle varianti d’autore.

Il progetto potrebbe essere facilmente espanso includendo nuove risorse che presentino le caratteristiche necessarie per essere correttamente gestiste da PhiloEditor, ovvero che non presentino cambiamenti strutturali ma solo formali. Un’ulteriore estensione del progetto può essere l’aggiunta di digitalizzazione di manoscritti o di edizioni a stampa, affiancate da strumenti di manipolazione come zoom in modo da permettere di cogliere dettagli altrimenti poco visibili.

Infine, una prospettiva interessante può essere indubbiamente quella di rendere PhiloEditor 3.0 responsive e utilizzabile su piattaforme mobili, utilizzando le funzionalità del framework Bootstrap già adottato dall'applicazione per la componente grafica.

Bibliografia

[BDIV16] C. Bonsi, A. Di Iorio, P. Italia, F. Vitali – Manzoni's Electronic Interpretations. Semicerchio vol.2015/2: pp.91-99

[Boz06] Bozzi Andrea – Edizione elettronica dei testi e filologia computazionale. Fondamenti di critica testuale, Bologna pp. 207-232

[DP15] Chiara Di Pietro – EVT per le edizioni critiche digitali: progettazione e sviluppo di una nuova GUI basata sullo schema progettuale MVC. 2015. Tesi magistrale, Università di Pisa.

[DI98] Paolo D'Iorio – L'edizione elettronica, in "Annali della Scuola normale Superiore di Pisa" .v.III, s IV, 1998, n.1-2, pp. 257

[DIV14] A. Di Iorio, P. Italia, F. Vitali – Variants and Versioning between Textual Bibliography and Computer Science .Conferenza,AIUCD'14, settembre 18-19, 2014, Bologna

[IR10]P. Italia, G. Raboni – Che cos'è la filologia d'autore. 2010.Roma. Carocci pp. 9-11

[IV14] "P. Italia e F. Vitali - Metodologie a confronto:Variantistica" . YouTube video. Terzo convegno annuale AICUD 2014 - La metodologia della ricerca umanistica nell'ecosistema digitale, 18-19 Settembre 2014, Bologna. Postato da Università di Bologna. 12/12/14 . URL: https://www.youtube.com/watch?v=YqP53CzO8n4

[Nie95]Nielsen, Jakob – 10 Heuristics for User Interface .1/01/1995, URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (visitato il 10/05/2016).

[Ros03] Luca Carlo Rossi – Errori e varianti. 27/03/03. Unità didattica per il consorzio. Italian culture on the net, Pisa.
URL: http://www00.unibg.it/dati/corsi/24038/44814-Errori%20e%20varianti%20-%20L.C.%20Rossi.pdf

[Tom01] Francesca Tomasi – Le nuove frontiere della filologia: metodi conservativi e nuove tecnologie informatiche. in "Argo", Anno II, Numero II, 2001. URL: http://www.argonline.it/territori/territorio_due/tomasi_filologia.html (visitato il 3/05/06)