Alma Mater Studiorum · Università di Bologna

**FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI**

**Corso di Laurea in Informatica triennale**

# APPROCCIO DICHIARATIVO ALLA GENERAZIONE AUTOMATICA DEL LAYOUT PER TESTO A FRONTE SU PIATTAFORMA WEB

**Relatore:**
**Angelo Di Iorio**

**Presentata da:**
**Luca Antinori**

**Seconda Sessione**
**Anno Accademico 2014/2015**

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Introduzione

Negli ultimi anni con la sempre maggiore diffusione del web, la crescente diversificazione dei dispositivi usati per la visualizzazione di contenuti (smartphone, PDA, laptop, e-book reader, etc..) e la nascita di tecniche per la personalizzazione dei contenuti in base alle esigenze specifiche di ogni utente(come RSS e le sue variazioni) è nata l'esigenza di metodologie pratiche per la generazione automatica di presentazioni efficaci e adatte alla rappresentazione dei loro contenuti cercando al tempo stesso di minimizzare l'intervento umano richiesto per tale scopo.

Ad oggi le principali soluzioni esistenti richiedono la presenza di un documento di input che immagazzini i contenuti in maniera astratta e computabile tramite la presenza di metadati descriventi la semantica delle vari parti del documento (come per esempio un documento xml). Successivamente possono essere usati *template* per la generazione del layout finale e per riversare i contenuti del documento di input al suo interno oppure algoritmi specializzati di generazione del layout basati sull'ottimizzazione di vincoli geometrici.

Il problema con il primo approccio basato su template[5, 6, 7] è la carenza di generalità e la complessità di scrittura dello stesso, poiché in questo caso si necessità che il designer scriva direttamente il template in base alle specifiche esigenze di presentazione del contenuto e per fare questo deve avere un ampia conoscenza del linguaggio usato dal template in questione.

D'altra parte un approccio al layout basato su algoritmi a ottimizzazione geometrica vincolata[4, 9, 11, 12, 13] è computazionalmente molto onerosa, non adatta alla generazione frequente di layout diversi su dispositivi con

scarse risorse di calcolo (come gli smartphone ad esempio), e anche se producono layout molto più generici e adattabili di quelli ottenuti tramite template presentano anche essi carenza di generalità poiché allo stato dell'arte questi algoritmi generano solo un determinato tipo di layout (come per esempio quello stile quotidiano[11]) ed inoltre la qualità finale del layout prodotto è decisamente inferiore rispetto a quello ottenuto con altri approcci e non tiene conto di esigenze tipografiche specifiche.

Un terzo tipo di paradigma, a metà strada tra i due precedentemente elencati è quello dichiarativo, in questo tipo di approccio il designer specifica direttamente le caratteristiche che vuole per il suo layout in termini di proprietà generalmente ad alto livello di astrazione invece che in termini di proprietà geometriche, in modo tale da permettere l'uso di questo tipo di motori anche ad utenti inesperti e contemporaneamente mantenere anche un certo grado di controllo sul layout finale. Seguendo questo approccio non si elimina totalmente il fattore umano come avviene con gli algoritmi a ottimizzazione vincolata, ma si può ridurre allo stretto indispensabile rispetto al paradigma procedurale basato su template. In tal modo si ottengono layout di qualità accettabile in tempi ridotti, soddisfacendo allo stesso tempo le esigenze tipografiche specifiche del designer.

Le tecniche di layout sopra descritte trattate in letteratura e nelle implementazione di motori esistenti si occupano di generare un ampia gamma di tipologie di macro layout che vanno dall'impaginazione automatica di quotidiani a formattazioni automatiche ancora più generiche tramite l'espressione di proprietà di layout ad alto livello da parte del designer; ma nessuna degli approcci ad oggi esistenti propone un approccio valido per la generazione automatica di layout specifico per il testo a fronte. Questo è dovuto al fatto che l'impaginazione a fronte (e quindi il layout finale) è legata direttamente alla semantica delle parti dei contenuti stessi da sincronizzare, ed è impossibile quantificare programmaticamente la semantica contenuti senza un intervento umano esterno che aggiunga metadati appositi.

Il testo a fronte è una branca del macro layout, ed è una tipologia di

impaginazione usata spesso in editoria per il raffronto parallelo di due o più testi, in cui la corretta formattazione e allineamento di questi ultimi è cruciale per una corretta comprensione dei contenuti stessi. Le soluzioni tradizionalmente usate per questo tipo di formattazione implicano generalmente l'intervento diretto di un designer che formatti manualmente le varie parti di contenuti da allineare con il linguaggio di formattazione usato o tramite l'ausilio di software per il *desktop publishing* poiché nella maggior parte dei casi le parti devono essere allineate secondo criteri semantici ben precisi non completamente gestibili programmaticamente; in alternativa si può ricorrere a un approccio procedurale basato su template in cui il designer deve scrivere comunque del codice che generi la formattazione desiderata nel linguaggio di formattazione usato, ma anche in questo caso oltre all'onere di scrivere un template ad hoc per la formattazione richiesta, i contenuti del file di input usato dal template devono essere comunque marcati tramite un linguaggio di *markup*(tipicamente xml) in maniera molto scrupolosa, in maniera tale da aggiungere le informazioni e i metadati necessari per un elaborazione programmatica dei contenuti, ed è comunque un processo molto lungo e oneroso che va bene solo per tipi di layout a fronte molto specifici. Per questo si è pensato a una soluzione dedicata a questa tipologia di layout che fino ad ora è stata trascurata.

Il contributo di questo lavoro è stata la realizzazione di un motore di layout specializzato per il testo a fronte basato su paradigma dichiarativo chiamato Sider, che permette all'utente di generare automaticamente layout a fronte anche complessi specificando direttamente le regole da utilizzare per l'allineamento, riducendo drasticamente i problemi legati agli approcci sopra citati. La scelta di seguire un approccio dichiarativo invece, è stata dettata dal fatto che è il paradigma che permette di ottenere il miglior compromesso tra automatizzazione del layout e interazione utente poiché per natura stessa del testo a fronte l'intervento umano non è totalmente eliminabile.

Successivamente nel capitolo 1 si presenta una panoramica globale sul mondo del layout e sulle metodologie ad oggi esistenti per la sua generazione

automatica. Nel capitolo 2 si introdurranno le caratteristiche principali di Sider, seguite da alcuni esempi, casi d'uso e modalità di utilizzo. Nel capitolo 3 verranno elencati i dettagli implementativi e architetturali del motore.
newpage

# Capitolo 1

# Il Layout e Testo a fronte

## 1.1 Layout e testo a fronte

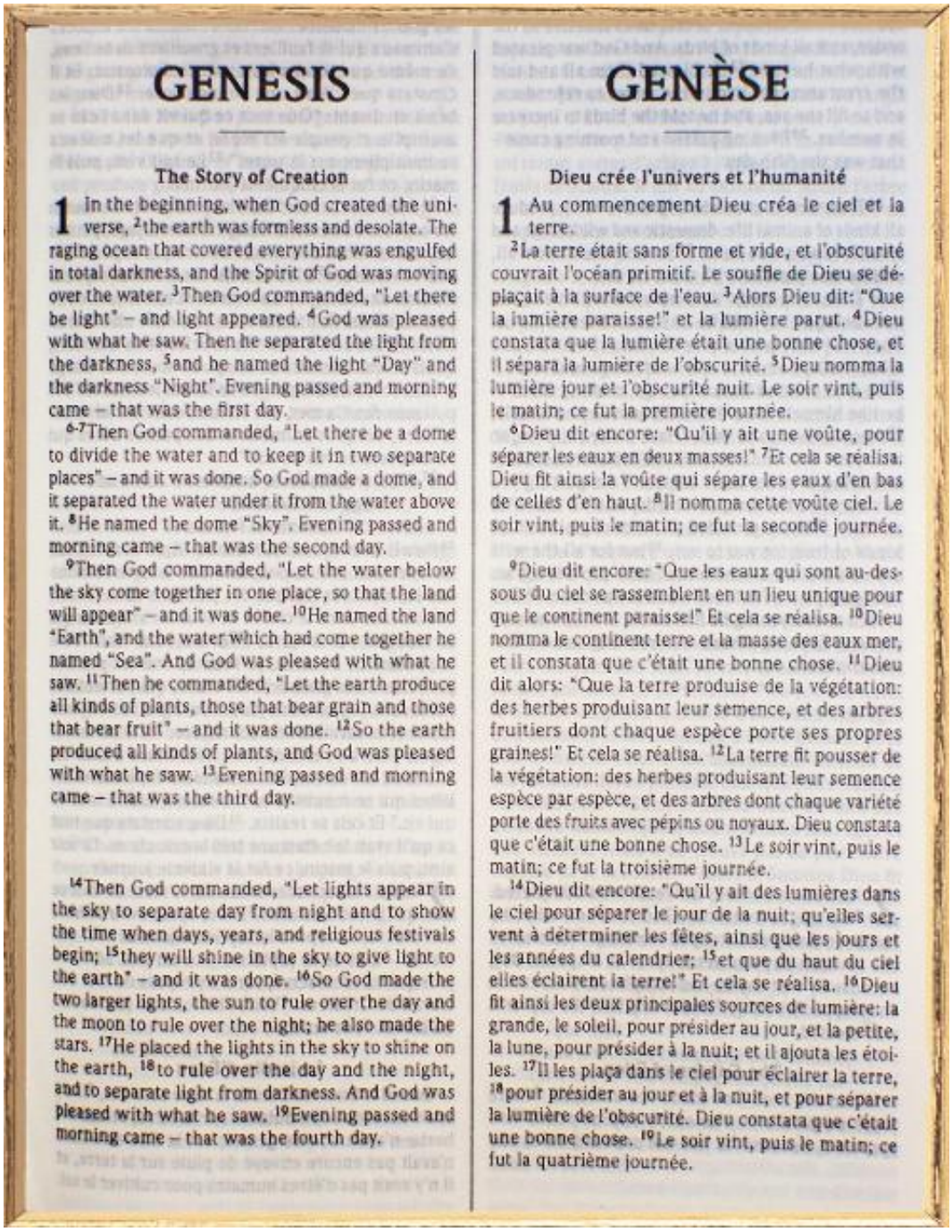
Fin dalla nascita della tipografia il layout è sempre stato una componente fondamentale se non imprescindibile dei contenuti stessi trasmessi con la parola scritta. Il layout (impaginazione in italiano) è una branca del design grafico e si occupa della disposizione visuale degli elementi tipografici su pagina o su un qualsiasi altro tipo di supporto usato per la trasmissione delle informazione in formato digitale e non. Più genericamente comprende l'insieme dei principi organizzativi nella composizione dei contenuti (o elementi tipografici) sul supporto di destinazione atta al raggiungimento di specifici obbiettivi comunicativi che sono strettamente legati alla semantica dei contenuti in oggetto.
Il layout si può suddividere in due macro aree: il layout ad alto livello consiste nella disposizione generale dei vari elementi tipografici su pagina e eventualmente anche nella decisione sulla forma, grandezza e tipo di supporto di destinazione e il layout di basso livello che comprende la paginazione e la composizione tipografica dei singoli elementi. Prima dell'avvento del desktop publishing sia la parte di alto livello si quello di basso livello erano interamente costruite manualmente dall'editore, ora invece la parte di basso livello è totalmente automatizzabile e richiede solo l'inserimento di alcuni

paramenti come per esempio i margini delle caselle di testo o le direttive di giustificazione del testo, che comunque una volta generate richiedono solo un marginale intervento manuale per eventuali rifiniture in base all'esigenza dell'editore.
Quello ad alto livello invece è tuttora un problema aperto poiché può variare anche notevolmente da caso a caso in base al tipo di contenuti da trasmettere al lettore e non esistono regole empiriche che ne permettano un automazione efficace per tutte le casistiche esistenti (ammesso che siano finite).
Questo ha portato col tempo allo sviluppo di template ad alto livello, ovvero di modelli predefiniti e ottimizzati per uno specifico tipo di contenuto, stile di impaginazione e supporto di destinazione; l'esempio per eccellenza è il layout ad alto livello usato per i comuni quotidiani, generalmente tutti hanno lo stesso stile tipografico ma diversificato per ogni pagina, per esempio il layout della prima pagina ha una disposizione molto diversa rispetto alle successive,stessa cosa con riviste, libri, volantini, ecc... E spesso anche questo tipo di approccio da solo non basta e richiede quasi sempre l'ausilio di designer grafici specializzati a supervisionare e modificare ad hoc i template esistenti secondo le esigenze specifiche degli editori, tipologie di elementi e contenuti da impaginare.


Figura 1.1: Tipico prototipo di testata di un quotidiano


Figura 1.2: Tipico layout di una pagina generica di un quotidiano

Entrando più nello specifico un sotto problema del layout di alto livello è la gestione macroscopica della disposizione dei singoli elementi tipografici,in maniera particolarmente accentuata il testo a fronte.
La formattazione canonica generalmente usata per il testo a fronte, consiste nel disporre parallelamente su una pagina o in una sotto sezione di essa due o più colonne contenenti testo o altri tipi di contenuto multimediale al fine di agevolare e ottimizzare la lettura di testi che richiedono un rapido raffronto con gli altri per la comprensione del contenuto in oggetto.
Nella maggior parte dei casi non basta semplicemente disporre i vari contenuti da raffrontare su colonne diverse, ma si ha anche l'esigenza di un allineamento orizzontale tra gli elementi costituenti le colonne stesse che può essere anche molto articolato a seconda delle specifiche esigenze dell'editore.
Un esempio classico di testo a fronte sono le traduzioni multilingue di un testo che in genere è costituito da due o più colonne ognuna in una lingua diversa, in cui vengono allineati orizzontalmente gli elementi del contenuto in oggetto, questi elementi in genere possono essere costituiti sia dalle singole parole che da frasi o da interi periodi. Nella figura 3 per esempio le unità di allineamento usate sono interi periodi.
Il testo a fronte o allineamento parallelo di uno o pi testi sugli elementi costituenti le sue colonne sono un prerequisito fondamentale per molte aree della linguistica, legislazione e anche sceneggiature.
Durante la composizione di un testo a fronte gli elementi da allineare possono essere frammentati, uniti, cancellati, inseriti o riordinati dall'autore in maniera arbitraria o in base alla semantica dei contenuti, per questo la generazione automatica del layout ad alto livello di queste tipologie di layout non è un problema non triviale. Questo tipo di problema è particolarmente difficile da automatizzare poiché il layout finale dei singoli elementi costituenti i contenuti è strettamente correlato alla semantica degli altri costituenti le colonne da allineare e in genere richiede per la sua risoluzione un intervento umano diretto.



# GENESIS

### The Story of Creation

1 In the beginning, when God created the universe, 2 the earth was formless and desolate. The raging ocean that covered everything was engulfed in total darkness, and the Spirit of God was moving over the water. 3 Then God commanded, "Let there be light" – and light appeared. 4 God was pleased with what he saw. Then he separated the light from the darkness, 5 and he named the light "Day" and the darkness "Night". Evening passed and morning came – that was the first day.

6-7 Then God commanded, "Let there be a dome to divide the water and to keep it in two separate places" – and it was done. So God made a dome, and it separated the water under it from the water above it. 8 He named the dome "Sky". Evening passed and morning came – that was the second day.

9 Then God commanded, "Let the water below the sky come together in one place, so that the land will appear" – and it was done. 10 He named the land "Earth", and the water which had come together he named "Sea". And God was pleased with what he saw. 11 Then he commanded, "Let the earth produce all kinds of plants, those that bear grain and those that bear fruit" – and it was done. 12 So the earth produced all kinds of plants, and God was pleased with what he saw. 13 Evening passed and morning came – that was the third day.

14 Then God commanded, "Let lights appear in the sky to separate day from night and to show the time when days, years, and religious festivals begin; 15 they will shine in the sky to give light to the earth" – and it was done. 16 So God made the two larger lights, the sun to rule over the day and the moon to rule over the night; he also made the stars. 17 He placed the lights in the sky to shine on the earth, 18 to rule over the day and the night, and to separate light from darkness. And God was pleased with what he saw. 19 Evening passed and morning came – that was the fourth day.

# GENÈSE

### Dieu crée l'univers et l'humanité

1 Au commencement Dieu créa le ciel et la terre.

2 La terre était sans forme et vide, et l'obscurité couvrait l'océan primitif. Le souffle de Dieu se déplaçait à la surface de l'eau. 3 Alors Dieu dit: "Que la lumière paraisse!" et la lumière parut. 4 Dieu constata que la lumière était une bonne chose, et il sépara la lumière de l'obscurité. 5 Dieu nomma la lumière jour et l'obscurité nuit. Le soir vint, puis le matin; ce fut la première journée.

6 Dieu dit encore: "Qu'il y ait une voûte, pour séparer les eaux en deux masses!" 7 Et cela se réalisa. Dieu fit ainsi la voûte qui sépare les eaux d'en bas de celles d'en haut. 8 Il nomma cette voûte ciel. Le soir vint, puis le matin; ce fut la seconde journée.

9 Dieu dit encore: "Que les eaux qui sont au-dessous du ciel se rassemblent en un lieu unique pour que le continent paraisse!" Et cela se réalisa. 10 Dieu nomma le continent terre et la masse des eaux mer, et il constata que c'était une bonne chose. 11 Dieu dit alors: "Que la terre produise de la végétation: des herbes produisant leur semence, et des arbres fruitiers dont chaque espèce porte ses propres graines!" Et cela se réalisa. 12 La terre fit pousser de la végétation: des herbes produisant leur semence espèce par espèce, et des arbres dont chaque variété porte des fruits avec pépins ou noyaux. Dieu constata que c'était une bonne chose. 13 Le soir vint, puis le matin; ce fut la troisième journée.

14 Dieu dit encore: "Qu'il y ait des lumières dans le ciel pour séparer le jour de la nuit; qu'elles servent à déterminer les fêtes, ainsi que les jours et les années du calendrier; 15 et que du haut du ciel elles éclairent la terre!" Et cela se réalisa. 16 Dieu fit ainsi les deux principales sources de lumière: la grande, le soleil, pour présider au jour, et la petite, la lune, pour présider à la nuit; et il ajouta les étoiles. 17 Il les plaça dans le ciel pour éclairer la terre, 18 pour présider au jour et à la nuit, et pour séparer la lumière de l'obscurité. Dieu constata que c'était une bonne chose. 19 Le soir vint, puis le matin; ce fut la quatrième journée.

Figura 1.3: Esempio classico di testo a fronte per periodi

## 1.2 Layout su digitale e Desktop publishing

Ad oggi l'insieme di modalità de facto usate per l'impaginazione e la formattazione di contenuti in maniera professionale e non, vengono comunemente raggruppate sotto il termine Desktop publishing (o DTP). Il DTP è il processo di creazione del layout appositamente studiato per il contenuto in oggetto tramite l'uso di software specializzati su personal computer. Questi programmi usati da un designer esperto possono generare contenuti impaginati e formattati ad hoc per le esigenze specifiche dell'editore e produrre testi e immagini di qualità tipografica anche molto elevata comparabili qualitativamente ai metodi tradizionali di tipografia e di stampa. Questi software possono emulare qualsiasi tipo di supporto di destinazione sia esso digitale che fisico quindi in genere sono usati per la creazione di layout per tutti i tipi di supporti, anche se certi software sono orientati e/o specializzati per specifici supporti rispetto che altri.[15]

I paradigmi usati dai software specializzati per il DTP si dividono principalmente in due macro categorie:I *WYSIWYG* ("what you see is what you get") e i *WYSIWYM*("what you see is what you mean").

I *WYSIWYG* consistono in un editor ad interfaccia grafica che mostra direttamente i contenuti formattati sul supporto di destinazione emulato ,in cui il designer ottiene il layout desiderato senza conoscere il linguaggio di formattazione usato dall'editor stesso interagendo direttamente con i componenti del layout su schermo. Tale linguaggio di formattazione viene automaticamente generato dal motore interno del software usato; un esempio di questo tipo di software è Adobe InDesign oppure Scribus nel mondo open source insieme a molti altri. Il vantaggio principale di questa tecnica software è che permette di visualizzare direttamente un documento molto simile se non uguale al risultato finale durante tutto il processo di creazione, e la possibilità di manipolare il layout direttamente da interfaccia senza scrivere direttamente le direttive di layout al motore di formattazione interno.

Dall'altra parte abbiamo i software *WYSIWYM* in cui il documento formattato creato viene assemblato e visualizzato solo dopo che l'utente ha

specificato direttamente il markup desiderato sui contenuti, la peculiarità fondamentale di questo paradigma è che durante lo sviluppo il designer si interfaccia solo su una fattispecie di presentazione simile a quella finale ma non esattamente uguale. Negli editor *WYSIWYM* l'utente dispone i contenuti in maniera strutturata, marcandoli secondo il loro significato specifico all'interno del documento lasciando la presentazione finale del documento a uno o più fogli di stile separati. Per esempio in un documento di questo tipo il designer marca manualmente le sezioni del documento con una determinata semantica come un titolo, un nome di una sezione, un indirizzo o il nome dell'autore e questo permette successivamente di attribuire direttive di visualizzazione su un foglio di stile esterno come per esempio specificare il colore, l'allineamento o la grandezza del titolo limitando fortemente un futuro intervento manuale per eventuali ritocchi. Questo tipo di DTP richiede quindi di impostare la struttura semantica del documento prima della sua stesura inoltre richiede funzionalità integrate per l'esportazione dei contenuti così strutturati in formati intermedi alla presentazione per permettere l'intercambiabilità dei fogli di stili ad esso associati. Il vantaggio principale del *WYSIWYM* è la separazione tra presentazione e contenuti poiché l'editore tipicamente scrive e struttura il documento una sola volta lasciando i vari tipi di presentazione a fogli di stile modulari esterni . Un esempio di software di questo tipo è LyX o Texworks che sono più incentrati sulla semantica del documento piuttosto che sulla presentazione finale.

Degno di nota inoltre sono anche i "word processors" e i software da ufficio che ad oggi forniscono tutti almeno un supporto sufficiente per le funzionalità di layout di base di tipo *WYSIWYG*.

Le caratteristiche fondamentali che tutti i software di DTP hanno in comune sono un interfaccia grafica con cui l'utente puï¿½ interagire direttamente, un linguaggio di markup per descrivere il layout su cui vengono salvate le direttive vere e proprie (come per esempio Latex per Lyx) e un motore interno che traduce il linguaggio interno nell'output finale. Inoltre fondamentale per questi tipi di motori è la capacità di importazione ed espor-

tazione in diversi formati di input e di output come ad esempio in Pdf o Dvi. C'è da dire inoltre che nel paradigma *WYSIWYM* l'utente ha il totale controllo sul linguaggio di markup del layout interno cosa non sempre vera nei *WYSIWYG* che non sempre offrono un capillare controllo su di esso.

Parlando di layout su web molti software DTP hanno motori interni con un supporto sempre maggiore per l'esportazione su piattaforma web e sono concepiti appositamente per agevolare l'utente alla creazione di pagine web senza il prerequisito di conoscenza del loro specifico linguaggio di markup (html e css), anche se i documenti creati tramite questi ultimi tendono ad avere un overhead di markup molto elevato e riducono anche di parecchio il controllo dell'utente sul documento finale e quindi sul suo linguaggio di formattazione (premettendo di interagire solo tramite interfaccia grafica). La tecnologia di DTP su web presenta dei *trede-off* ancora troppo alti e quindi molti utenti preferiscono scrivere direttamente nel linguaggio di formattazione web senza l'ausilio di software terzi.

Negli ultimi anni grazie allo sviluppo sempre maggiore delle tecnologie per il layout il confine che separa il layout cartaceo, quello paged su digitale (pdf, epub, dvi, etc..) e quello digitale su web sta sbiadendo sempre di più e si andrà verso l'utilizzo di motori, linguaggi e software sempre più a largo spettro di formati supportati e quindi a un unificazione globale del modo di concepire il layout. Degno di nota a riguardo è il tentativo del w3c di unificare web e gli altri supporti esistenti, proponendo uno standard appositamente studiato per trasformare il layout da web a paged-media e cartaceo A4. Tramite un approccio modulare su css Paged Media Module[1] e il suo sotto modulo Print Profile[2], il w3c propone uno standard per partizionare dei flussi di contenuto in pagine statiche di dimensione fissa su supporto digitale e/o cartaceo come nel caso specifico di Print profile. Questi due moduli mettono a disposizione dell'utente tutta una serie di meccanismi sviluppati ad hoc per disporre il flusso di contenuti su supporto a pagina:

- Regole di interruzione dei contenuti
- Proprietà specifiche della pagina

- Regole di inclusione header e footer nella pagina
- Modalità di numerazione delle pagine

Questo standard è ancora in fase embrionale e manca di parecchie funzionalità per rivaleggiare contro le tecniche e gli strumenti di DTP esistenti per gli altri supporti ma è un primo passo per un unificazione globale del layout su web con gli altri supporti e formati.

## 1.3 VDP e generazione automatica del layout

Nel paragrafo precedente sono state introdotte le principali tecniche moderne per la formattazione e l'impaginazione classiche più usate nel campo della tipografia moderna, nel seguente paragrafo si passerà in rassegna il concetto di VDP, la generazione automatica del layout, gli strumenti e le tecnologie emergenti usate per tale scopo. Un documento statico per definizione è un documento dove ogni istanza di esso è esattamente la stessa, un documento variabile si può intendere invece come un "framework" dal design prefissato in cui sia i contenuti statici che quelli dinamici vengono assemblati assieme come in un collage. I contenuti dinamici (variabili) nei documenti possono variare nel tempo, tra un processo di stampa del documento e un altro oppure variano in base a uno flusso di dati che riflette una selezione logica di parti di contenuto mirata a generare delle stampe (o documenti digitali) uniche per un determinato destinatario, argomento o scopo. Ci sono molti tipi di documenti classificabili come "variabili" che possono essere prodotti con tecniche di assemblaggio dinamico di contenuto, per esempio:

- biglietti di auguri e inviti matrimoniali
- cataloghi e mail di marketing
- documenti e contratti per il business
- Reports e articoli accademici

- antologie di poesie, libri di cucina e atlanti
- Newsletters, volantini pubblicitari e menù per ristoranti

Indipendentemente dai contenuti, dagli obbiettivi e dal "look and feel" i suddetti documenti differiscono principalmente da quando e da come variano; ma in termini di design, tecnologia e processo creativo sono molto simili. Possiamo quindi usare il termine VDP (Variable Data Printing) per riferirci alle suddette tecniche e metodologie di composizione e assemblaggio di documenti dinamici[3]. Per completare il quadro con il paragrafo 1.2 bisogna citare per completezza che molti software moderni di DTP posseggono supporti di qualche tipo per il VDP, in forma di funzionalità integrate o come plugin opzionali, come ad esempio la funzionalità di Database Publishing che permette di riversare flussi di contenuti da un Database selezionati dall'editore in uno o più componenti del framework nelle pagine di destinazione e quindi nel documento finale; mettendo a disposizione dell'editore anche funzionalità semi-automatiche per la generazione del layout.

Una branca di studio del VDP è la formattazione automatica di documenti o generazione automatica del layout, che si sovrappone per molti aspetti concettuali e tecnologici ai temi trattati dal VDP. Negli ultimi anni c'è stato un interesse sempre maggiore verso il layout automatico a causa dell'avvento del WWW e del diffondersi del VDP in campo industriale, e questo ha portato a uno slittamento di interesse da aspetti micro tipografici a macro tipografici. Uno degli obbiettivi di design dell'architettura web moderna è separare i contenuti del documento dalla sua presentazione, in maniera tale da permettere al layout di adattarsi alla lettura su diversi dispositivi e ai bisogni dei singoli utenti. Oltretutto con l'avvento dei contenuti dinamici si è reso impossibile per l'autore stabilire completamente il layout finale dei documenti. Questo è un problema sia per il Web ma anche per i documenti generati con tecniche di VDP le quali si sono diffuse negli ultimi anni come standard de facto per l'industria tipografica perché hanno permesso agli editori e alle aziende di tagliare anche molto su i costi di stampa sui documenti personalizzati per ogni specifico destinatario. Per questo la richiesta di layout

generato automaticamente di qualità sarà sempre in crescendo. I dispositivi usati per la visualizzazione di documenti si stanno diversificando sempre di più (stampe, monitor, eBook, PDA, etc.) rendendo la questione di adattamento automatico del layout sempre più appetibile, e la sempre maggiore disponibilità di informazioni sulle preferenze e dei bisogni degli utenti permettono una miglior personalizzazione della presentazione dei contenuti dei documenti. Perciò lo sviluppo di tecniche per la generazione automatica di un layout di qualità e ad hoc continuerà ad essere un tema centrale nei prossimi anni avvenire. Purtroppo la questione della generazione automatica del macro layout è tutt'altro che triviale; per capire la complessità del problema si fa riferimento al lavoro di Hurst-Li-Marriott nell'articolo "Review of Automatic Document Formatting": "La formattazione automatica di documenti ad alta qualità di layout è un problema estremamente difficile. Le difficoltà sono principalmente tre. La prima, è la difficoltà di quantificare cosa rende un determinato tipo di layout di qualità.Mentre è impensabile aspettarsi che i sistemi automatici di layout rivaleggino con la creatività di un designer grafico esperto, si ha bisogno di fissare dei metodi per identificare quando un layout è palesemente sbagliato in tal modo da generare un layout ragionevole. Secondo,il layout è computazionalmente difficile. Persino dei sotto task a prima vista semplici come layout tabellare o certi tipi disposizioni fluttuanti sono NP-complessi. La terza difficoltà è la complessità di progettazione e implementazione di strumenti layout-dedicati. Questo perché è difficile separare il processo di layout in sotto operazioni ben definite, anche perché le decisioni prese sul layout interagiscono fra di loro - lo spezzamento di una linea ha effetto sul piazzamento di una figura - e le convenzioni tipografiche sono estremamente diversificate. Per questo è facile ritrovarsi con un sistema software molto complesso e specializzato per il settore in questione."[4].

L'articolo passa in rassegna i principali algoritmi esistenti al 2009 per il micro e macro layout presentandoli in base alla loro bontà computazionale espressa secondo un problema di ottimizzazione vincolata ovvero utilizzando delle variabili contenenti i valori di posizionamento degli elementi costituenti

la pagina e dei vincoli che forzano la disposizione degli stessi secondo relazioni geometriche precise, come l'allineamento o il contenimento di una pagina, e successivamente date in input a una funzione oggettiva che misura la bontà del layout finale di un determinato problema o sotto problema. L'articolo conclude dicendo che al 2009 le tecniche per la generazione automatica del micro layout sono abbastanza efficienti e robuste da essere usate in applicazioni pratiche ma dall'altra parte invece quelle riguardanti il macro layout sono ancora acerbe e non ancora adatte all'uso in applicazioni pratiche[4].

Ora si passeranno in rassegna alcune delle proposte accademiche, linguaggi e motori ad oggi usati per la generazione del layout.

Una soluzione proposta dal w3c per il supporto alla formattazione automatica di contenuti è XSL-FO , la specifica dedicata esclusivamente alla formattazione e alla resa grafica di XSL.XSL-FO (XSL Formatting Objects) è una sottospecifica di xslt ed è un formato intermedio che descrive le proprietà di resa grafica di uno o più flussi di contenuto e non presenta la dicotomia fra contenuto e presentazione (come avviene invece con html/css) poiché sono tutti e due miscelati assieme in un unica struttura. E' un linguaggio concepito appositamente per i media su supporto pagina ed in genere viene usato per la generazione di documenti PDF anche se può generare in output altri formati in base al motore di formattazione usato, ma attualmente nessuno motore supporta a pieno il formato web(html/css). FO è usato per generare a partire da dati e metadati contenuti in un XML una sua corrispondente descrizione grafica ibrida che successivamente in base al motore utilizzato può essere convertita in uno specifico *PDL* (Page Description Language) come ad esempio PDF o PS. Il genere di layout prodotto con questo tipo di processo è tipicamente compreso fra la tipologia in formato pagina e indicizzato come ad esempio libri, manuali o documenti . XSL-FO nella sua ultima versione esistente fornisce supporto per flussi multipli di contenuti e supporto di base per layout basato su regioni rettangolari, rendendo cosi questo linguaggio inadatto per layout più complessi e orientati alla grafica come quelli di riviste o giornali[5].

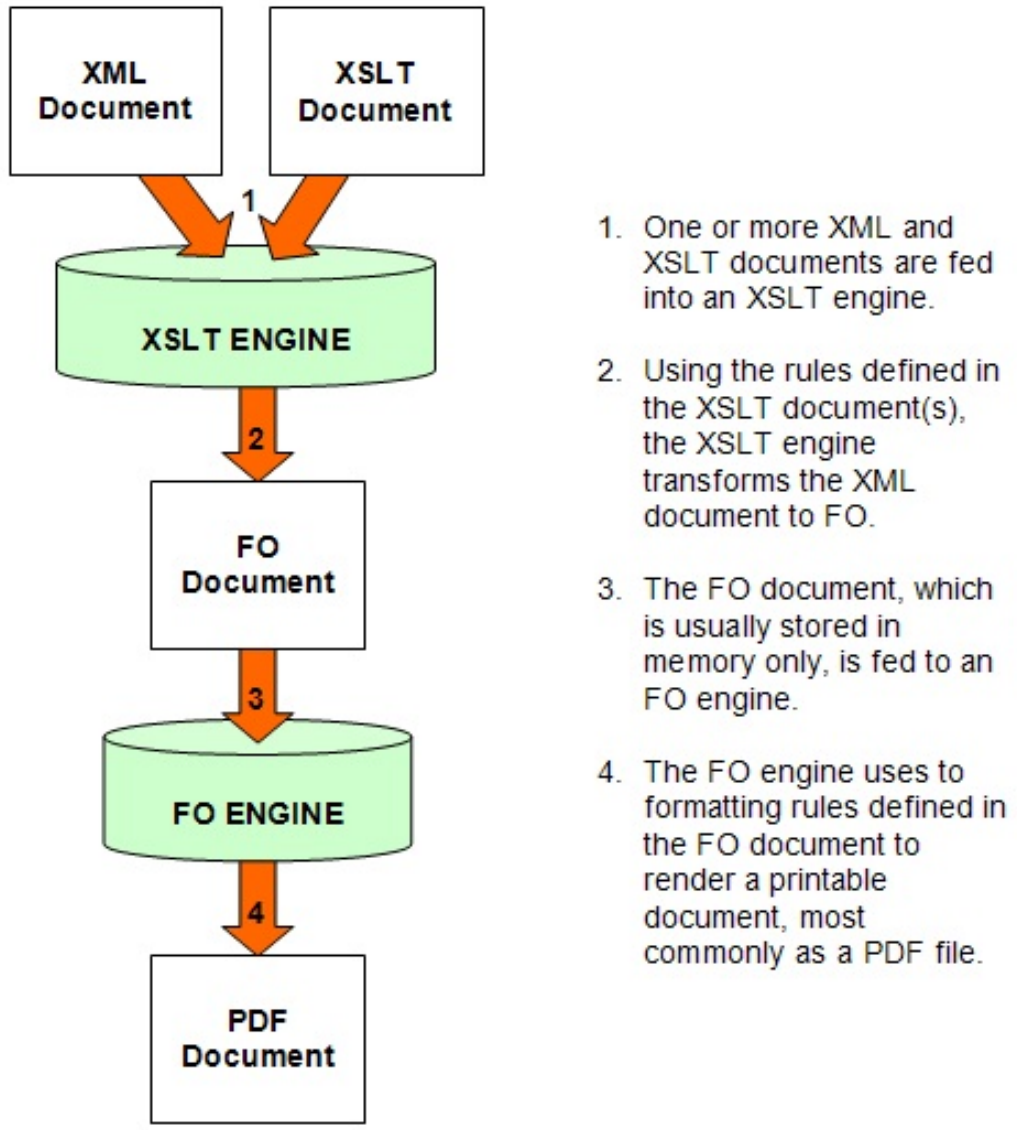



Figura 1.4: Processo di generazione di un documento in FO

Nella sua versione 2.0 sono state aggiunte funzionalità avanzate per l'internazionalizzazione e l'integrazione con le tecnologie SVG e di MathML oltre a ulteriore supporto per formattazione grafica come la funzionalità di "Text Run-around", contenitori non-rettangolari per contenuti, spezzamento automatico di tabelle su pagine multiple e aggiustamento automatico del

layout di pagina tramite "interazione intra-regionale" con regole specificate dall'autore[6, 7]. Comunque la versione 2.0 di FO è solo in forma di specifica e non esiste nessuna implementazione del suo motore di formattazione e inoltre il suo working group è stato chiuso poiché recentemente si è deciso di passare a un approccio più a largo spettro usando come base il motori di formattazione web di html e css, aggiungendo supporto modulare per i formati "paged" e cartacei tramite moduli css esterni come accennato nel paragrafo 1.2 con "Paged Media" e "Print Profile". Con questo approccio al layout automatico il designer deve scrivere direttamente un template XSLT che converta il file XML nel formato di FO rendendo questo metodo procedurale ed estremamente verboso nonché difficilmente aggiornabile/personalizzabile.

In opposizione all'approccio procedurale usato con XSLT e FO c'è quello proposto con TALL. TALL è un linguaggio basato su astrazione topologica sui componenti di pagina, proposto nell'articolo "Higher-level Layout through Topological Abstraction". Si basa sull'idea di proporre un paradigma alternativo alla concezione del layout finora usata, esprimendolo anziché tramite proprietà geometriche pure, tramite proprietà astratte di alto livello con cui gli utenti hanno naturalmente più familiarità. TALL è un linguaggio basato su XML che mette a disposizione del designer svariate proprietà ad alto livello di astrazione da attribuire ai componenti della pagina come somiglianza, prossimità, importanza e raggruppamento, queste proprietà vengono definite nell'articolo proprietà topologiche da cui il nome TALL (Topological Abstract Layout Language). L'articolo sostiene che l'utente conosce il documento e la struttura da dare ai contenuti; l'utente sa quali sono i componenti e sotto componenti dai quali il documento è composto e anche quali sono le correlazioni fra di essi: certe parti sono più importanti di altre mentre altre possono essere omesse se necessario; delle parti sono simili, alcune devono essere elencate in un determinato ordine, mentre altre possono essere disposte liberamente. L'articolo sostiene inoltre che queste informazioni sotto forma di proprietà delle varie parti del contenuto sono indipendenti dalla geometria di output e che infatti fanno parte di un più alto livello di astra-

zione. Si sostiene inoltre che un linguaggio basata sulle suddette proprietà sia un linguaggio ideale per descrivere il layout, poiché permette all'utente di "parlare" direttamente al motore di layout in termini di concetti già familiari invece che in termini di parametri geometrici, lasciando questo onere al motore interno. Molto importante per gli autori è infine sottolineare che le proprietà topologiche non sono concepite per rimpiazzare totalmente quelle geometriche utilizzate comunemente, bensì sottolineano una coesistenza tra le due al fine di arricchire le possibilità con cui i vari layout possono essere descritti. Nell'articolo inoltre viene proposta anche un implementazione del motore di TALL anche se parziale[8].

Un altra forma di generazione automatica del layout è l'adattamento di un layout già esistente ai diversi tipi di media, che è l'approccio proposto nell'articolo di Nebeling-Matulic-Norrie "Adaptive Layout Template for Effective Web Content Presentation in Large-Screen Contexts", in cui viene proposta una metodologia per l'adattamento automatico del layout su pagine web in base alla grandezza dello schermo del dispositivo su cui è visualizzata. Gli autori propongono un insieme di template css e librerie javascript che collegati a una pagina web standard scalano automaticamente i contenuti in essa presenti ottimizzandoli per la visualizzazione sul dispositivo corrente. Per ottenere questo risultato i moduli aggiungono allo standard di HTML5 e CSS3 le seguenti funzionalità:

- Scalabilità automatica dei font per mantenere la grandezza fisica del testo in differenti contesti di visualizzazione.

- Alternabilità controllata tra layout a colonna singola e multi colonna per produrre un layout testuale e una lunghezza di linea proporzionale alle condizioni di visualizzazione.

- Paginazione automatica dei contenuti con supporto per il design multi colonna in combinazione al modello a scorrimento verticale del web.

- Adattamento e sostituzione automatica dei media su pagina scalando la grandezza e il dettaglio di immagini o video se disponibili in diverse risoluzioni.

Gli autori concludono auspicandosi che tali funzionalità di adattamento automatico siano integrate e meglio supportate in future versioni di CSS e HTML sostenendo che tramite esse si potrebbe evitare la generazione ad hoc della parte grafica delle pagine web per ogni specifico supporto, pratica usata ancora oggi[9].

Da notare inoltre che l'idea di fondo proposta dall'articolo sopra citato è quella su cui si basano molti "framework" web moderni come *Bootstrap*[10] e altri.

Un altro approccio interessante al layout automatico è quello proposto nell'articolo da Piccoli-Oliveira, che prendendo spunto dal lavoro precedente di Oliveira[12] propongono un algoritmo chiamato APL (Automatic Page Layout) in grado di produrre automaticamente pagine in formato di layout ispirato a quello comunemente usato dai quotidiani, dove la pagina è ricorsivamente decomposta tramite bisezioni successive verticali e orizzontali. Questo approccio è chiamato partizionamento a ghigliottina[4] e consiste nel dividere un area di una pagina di grandezza data in regioni sempre più piccole tramite bisezioni ricorsive fino a quando si hanno abbastanza aree da riempire con tutti gli elementi di input, in modo tale che ogni area ottenuta riesca a contenere tutti gli elementi di input in maniera più omogenea possibile. A differenza degli approcci precedenti che usano il partizionamento a ghigliottina, in questo algoritmo il layout prodotto non è vincolato da uno schema di divisione a righe orizzontale bensì utilizza come presupposto base l'utilizzo di un partizionamento della pagina in un numero discreto di colonne permettendo un calcolo delle aree partizionate libero da vincoli in altezza che secondo gli autori permette un layout di output di qualità superiore rispetto agli approcci precedenti. Per ottimizzare ulteriormente la qualità del layout generato gli autori impostano alcuni vincoli a cui l'algoritmo da loro proposto si attiene:

- Le pagine di output devono essere interamente coperte in maniera omogenea da tutti gli elementi di input.
- L'ordine degli elementi di input deve essere preservato.
- Tutti gli elementi di input devono essere disposti su pagina e non possono essere omessi.
- Le regioni rettangolari dedicate agli elementi di input non possono essere ruotate e non possono intersecarsi
- Per questioni di leggibilità ed estetica il testo nelle regioni non deve essere troppo grande o troppo piccolo. Quindi i *font* devono rispettare una coppia di vincoli reali rappresentanti la tolleranza minima e quella massima in proporzione a quelli delle altre regioni di testo.
- Possono essere disposti su pagina un numero indefinito di elementi, quindi si deve permettere al testo contenuto nelle varie regioni di espandersi o contrarsi per poter essere contenuto nella regione ad esso assegnata.

Il contributo maggiore di questo lavoro è l'algoritmo e l'idea che sta alla base di esso di permettere ai *font* nelle regioni di testo di espandersi e contrarsi in maniera dinamica in base agli elementi di input senza intaccare troppo la qualità del layout finale; l'algoritmo usa il motore di rendering di TeX per fare una previsione delle aree occupate dai vari elementi di input aggiustando le aree e la grandezza dei font di conseguenza; gli autori inoltre propongono un algoritmo ausiliario per simulare e fare una previsione dell'area occupata dalle regioni testuali senza TeX, a causa del problema di ottimizzazione che subentra richiamando ogni volta il motore tipografico di TeX per fare il calcolo delle aree[11].

Da un analisi degli articoli sopra citati e da altri lavori accademici di settore possiamo concludere che i metodi di generazione del layout oscillano perlopiù tra approcci basati su template e quelli basati su generazione

geometrica vincolata o a meta strada tra questi due paradigmi. L'approccio che si decide di utilizzare quindi dipende strettamente dal trade-off di tra quanto spesso o quanto velocemente un layout deve essere prodotto e le sue caratteristiche intrinseche di design richieste (unicità di design).

## 1.4 Modalità di layout per il testo a fronte

Passando in rassegna il materiale esistente sulla composizione automatica del macro layout si nota che le modalità e tecnologie proposte sull'argomento sono tante e diversificate e al contempo si delinea una preponderanza di approccia a questo genere di problema di tipo procedurale o con algoritmi basati su ottimizzazione vincolata, con la sola eccezione di TALL[8] . Parlando di testo a fronte, non sono stati trovati lavori antecedenti a questo a riguardo, questo è dovuto al fatto che come accennato nel capitolo precedente il testo a fronte è una nicchia del mondo del layout molto specializzata che oltre a presentare tutti i problemi legati alla realizzazione di un motore di layout automatico sopra citati presenta anche nella maggior parte dei casi un requisito addizionale di interdipendenza fra la disposizione degli elementi all'interno del layout di pagina e il contenuto semantico degli elementi stessi, per esempio in una traduzione a fronte gli elementi contenenti la stessa frase tradotta nelle varie lingue devono essere disposti sulla stessa riga per ragioni imprescindibili di chiarezza e comprensione. Per questo le soluzioni fino ad ora utilizzate per aggirare il problema si basano perlopiù su un approccio di tipo manuale o procedurale.

Quelle di tipo manuale di regola richiedono l'uso di un software di DTP in cui il designer deve disegnare direttamente la "tabella" di layout in base agli elementi da disporre a fronte; questo su cartaceo e *paged media*, mentre su piattaforma web le soluzioni "manuali" comunemente utilizzate sono principalmente 3 (e si basano tutte sulle stesse tecnologie di fondo):

- Allineamento tramite l'uso di tabelle in html o la proprietà table del css

- Allineamento tramite l'uso della proprietà float in css

- Allineamento usando l'uso di *framework* con supporto per layout a griglia

Per chiarezza ogni soluzione sarà seguita da un semplice esempio di codice con medesimi contenuti ed il suo relativo output.

L'allineamento tramite tabelle è il più usato, specialmente in passato e consiste nell'incapsulare ogni blocco di contenuto da allineare in una cella diversa, contenute dell'elemento html "table".

```
<!--Allineamento con tabelle-->
        <h1>Allineamento con tabelle</h1>
        <table border="0" style="width:100%">
                <tr>
                    <th><h3>Di voi mi innamorai (1829)</h3></th>
                    <th><h3>? ??? ?????</h3></th>
                    <th><h3>I Loved You</h3></th>
                </tr>
                <tr
        </table>
```

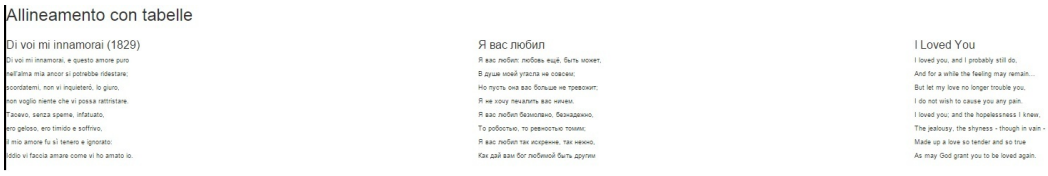



Figura 1.5: Allineamento con tabella html

In quello tramite css, si incapsula ogni riga di blocchi di contenuto da allineare all'interno di un contenitore "div" e tramite le proprietà "float" e "clear" si ottiene una formattazione a fronte. C'è anche da citare per completezza l'allineamento tramite la proprietà "display: table" e affini che non è elencato negli esempi perché molto simile alle soluzioni sopra citate,e si rimanda al sito w3c per ulteriori approfondimenti.

```
<!--Allineamento standard css-->
        <h1>Allineamento con css standard</h1>
        <div>
            <div class="riga">
                <div class="colonna"><h3>Di voi mi innamorai (1829)</h3></div>
                <div class="colonna"><h3>? ??? ?????</h3></div>
                <div class="colonna"><h3>I Loved You</h3></div>
            </div>
        </div>
```

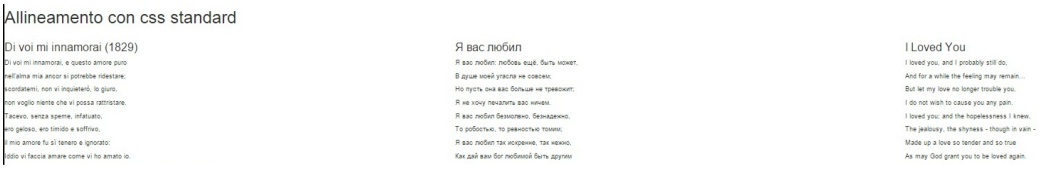



Figura 1.6: Allineamento con css

La terza soluzione è quella basata su framework css/javascript con supporto per layout a griglia che è quella su cui si basa il lavoro di questo elaborato; questa consiste sempre nell'incapsulare i vari blocchi di contenuto da allineare in elementi specifici del framework che rappresentano rispettivamente le righe e le singole celle delle colonne del layout finale desiderato, il tutto incapsulato a sua volta nel contenitore generico del framework. Da notare che esistono molti tipi di framework diversi ma con funzionalità più o meno simili , qui viene usato il framework bootstrap[10] come esempio ma il layout desiderato potrebbe essere realizzato con la maggior parte dei framework in circolazione semplicemente sostituendo i nomi degli elementi delle righe e/o colonne mantenendo inalterata la struttura dell'albero html.

```
<!--Allineamento con framework-->
        <h1>Allineamento con framework</h1>
        <div class="container-fluid">
            <div class="row">
                <div class="col-xs-4"><h3>Di voi mi innamorai (1829)</h3></div>
                <div class="col-xs-4"><h3>? ??? ?????</h3></div>
                <div class="col-xs-4"><h3>I Loved You</h3></div>
            </div>
        </div>
```

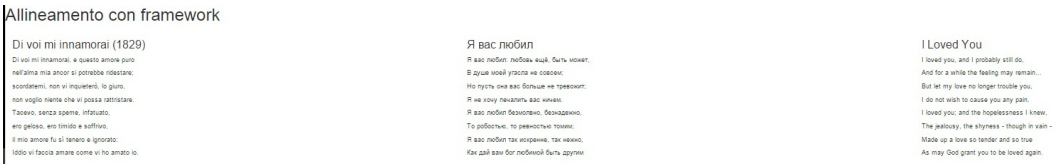



Figura 1.7: Allineamento con framework

Passando all'approccio procedurale riprendiamo il discorso del paragrafo 1.3 su XSL-FO[5] . Con questo tipo di soluzione il designer deve scrivere

direttamente le istruzioni di formattazione del documento in XSLT (o in gergo template):

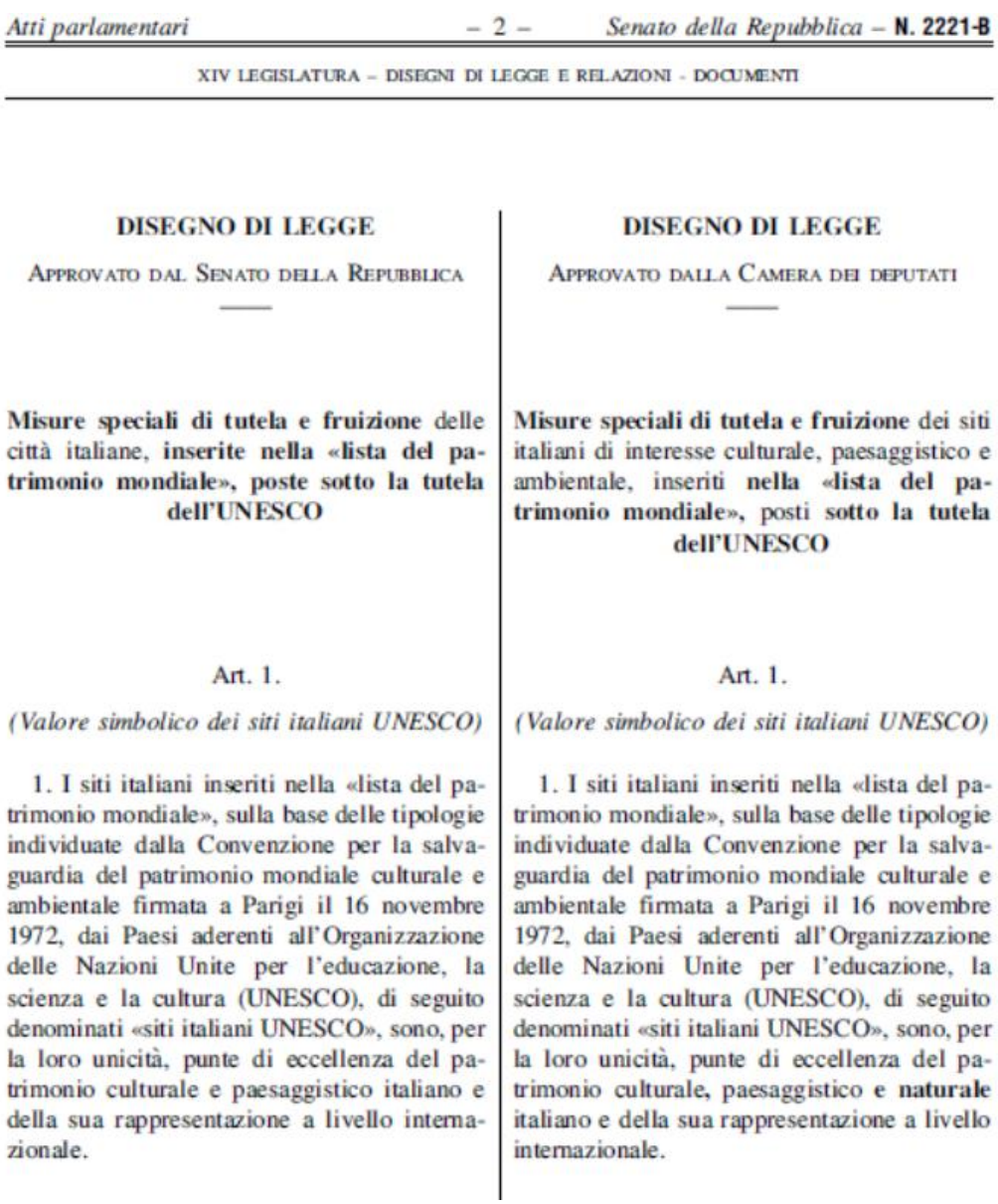
```
<xsl:template match="DisegnoLegge">
<html>
<head>
<title>
<xsl:value-of select="//emanante"/> - Disegno di Legge n. <xsl:value-of select="//numDoc"/>
</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<link href="../css/style.css" rel="stylesheet"/>
<!-- Stylesheet usato in locale da Angelo:
<link href="file://C:\mydata\Projects\NDiff\Repository\trunk\libra\default\css\style.css" rel="stylesheet"/>
-->
</head>
<body>
<table cellpadding="0" cellspacing="5" border="0" width="100%">
<!-- NOTA: momentaneamente intestazioni e metainformazioni non sono gestite. Ci concentriamo sull'articolato. -->
<!--
<tr style="margin:top:30px; margin-bottom:30px;">
<td colspan="2">
<xsl:apply-templates select="intestazione"/>
</td>
</tr>
-->
<tr valign="top">
<xsl:if test="(articolato | annessi) and ($indice = 'true')">
<td width="20%" style="border: thin solid #bbbbbb; background-color: #ffffcc;">
<div class="toc">Sommario</div>
<xsl:apply-templates select="articolato|annessi" mode="Link"/>
</td>
</xsl:if>
<td style="padding-top:15px;">
<xsl:choose>
<xsl:when test="($indice = 'true')">
<xsl:attribute name="width">80%</xsl:attribute>
</xsl:when>
<xsl:otherwise><xsl:attribute name="colspan">2</xsl:attribute></xsl:otherwise>
</xsl:choose>
<xsl:apply-templates select="relazione"/>
<xsl:apply-templates select="articolato | contenitore"/>
<xsl:apply-templates select="conclusione"/>
<xsl:apply-templates select="annessi"/>
<!--<xsl:apply-templates select="meta"/>-->
</td>
</tr>
</table>
</body>
</html>
</xsl:template>
```

Questo template si applica su uno o più documenti xml strutturati secondo determinati canoni:

```
<intestazione>
<h:p h:class="emananteddlmessaggio">
<emanante>SENATO DELLA REPUBBLICA</emanante>
</h:p>
<h:p h:class="attestazioneinmessaggio"> Attesto che la
<emanante>Commissione permanente</emanante>,
```

```
<h:br/>(Istruzione pubblica, beni culturali, ricerca scientifica,
 spettacolo e sport), il 9 febbraio 2006, ha approvato il seguente disegno di legge,
 d&#x2019;iniziativa dei senatori Asciutti, Viviani, Togni, Alberti Casellati,
  Eufemi, Delogu, Acciarini, Travaglia, D&#x2019;Ippolito, Fabbri, Falcier,
   Balboni, Battaglia Antonio, Ulivi, Tunis, Cortiana, Comincioli, Bianconi,
   Bettamio, Cavallaro, Compagna, Trematerra, Tomassini, Consolo, Monticone,
   Gubetti, Manieri, Vicini, Tredese, Favaro, Bevilacqua, Sudano,
   Danzi, D&#x2019;Andrea e Gaburro, gia' approvato dal Senato e modificato
   dalla Camera dei deputati:
<h:br/>
</h:p>
<h:p h:class="titoloinmessaggio">
<titoloDoc>Misure speciali di tutela e fruizione dei siti italiani
di interesse culturale, paesaggistico e ambientale, inseriti nella
&#xAB;lista del patrimonio mondiale&#xBB;, posti sotto la tutela
dell&#x2019;UNESCO </titoloDoc>
</h:p>
</intestazione>
```

E produce in output un documento formattato in XSL-FO che successivamente può essere usato per generare direttamente un file di *PDL* in base al motore usato (per esempio pdf) e infine può finalmente essere "renderizzato".



*Atti parlamentari* – 2 – *Senato della Repubblica* – **N. 2221-B**

XIV LEGISLATURA – DISEGNI DI LEGGE E RELAZIONI - DOCUMENTI

**DISEGNO DI LEGGE**

APPROVATO DAL SENATO DELLA REPUBBLICA

**Misure speciali di tutela e fruizione** delle città italiane, **inserite nella «lista del patrimonio mondiale», poste sotto la tutela dell'UNESCO**

Art. 1.

*(Valore simbolico dei siti italiani UNESCO)*

1. I siti italiani inseriti nella «lista del patrimonio mondiale», sulla base delle tipologie individuate dalla Convenzione per la salvaguardia del patrimonio mondiale culturale e ambientale firmata a Parigi il 16 novembre 1972, dai Paesi aderenti all'Organizzazione delle Nazioni Unite per l'educazione, la scienza e la cultura (UNESCO), di seguito denominati «siti italiani UNESCO», sono, per la loro unicità, punte di eccellenza del patrimonio culturale e paesaggistico italiano e della sua rappresentazione a livello internazionale.

**DISEGNO DI LEGGE**

APPROVATO DALLA CAMERA DEI DEPUTATI

**Misure speciali di tutela e fruizione** dei siti italiani di interesse culturale, paesaggistico e ambientale, inseriti **nella «lista del patrimonio mondiale», posti sotto la tutela dell'UNESCO**

Art. 1.

*(Valore simbolico dei siti italiani UNESCO)*

1. I siti italiani inseriti nella «lista del patrimonio mondiale», sulla base delle tipologie individuate dalla Convenzione per la salvaguardia del patrimonio mondiale culturale e ambientale firmata a Parigi il 16 novembre 1972, dai Paesi aderenti all'Organizzazione delle Nazioni Unite per l'educazione, la scienza e la cultura (UNESCO), di seguito denominati «siti italiani UNESCO», sono, per la loro unicità, punte di eccellenza del patrimonio culturale, paesaggistico **e naturale** italiano e della sua rappresentazione a livello internazionale.

Figura 1.8: Allineamento risultante con xslt-fo

Degno di nota è anche una funzionalità offerta da XSL-FO nella versione 2.0[6, 7], che permette la visualizzazione parallela di due o più flussi di contenuto, specificando esplicitamente un flusso principale, permette di sincronizzare le parti desiderate degli altri flussi tramite specifiche regole di sintassi e permette anche di specificare come i *gap* tra blocchi di contenuto sincronizzato devono essere trattati, tramite le meccaniche "Widow and Orphan". Le direttive di formattazione parallela si dividono in 2: La parte dove

si specifica la mappa dei contenuti (ovvero quanti flussi sa allineare,il loro nome univoco all'interno della mappa e la colonna su cui disporre il flusso):

```
<fo:flow-map>
   <fo:flow-assignment>
      <fo:flow-source-list>
        <fo:flow-name-specifier
           flow-name-reference="Arabic-Flow"/>
      </fo:flow-source-list>
      <fo:flow-target-list>
         <fo:region-name-specifier
         region-name-reference="Region-Middle"/>
      </fo:flow-target-list>
   </fo:flow-assignment>
   <fo:flow-assignment flow-map-dependency-reference=?Arabic-Flow?>
      <fo:flow-source-list>
         <fo:flow-name-specifier
            flow-name-reference="Armenian-Flow"/>
      </fo:flow-source-list>
      <fo:flow-target-list>
         <fo:region-name-specifier
           region-name-reference="Region-Right"/>
      </fo:flow-target-list>
   </fo:flow-assignment>
   <fo:flow-assignment flow-map-dependency-reference="Arabic-Flow">
      <fo:flow-source-list>
         <fo:flow-name-specifier
          flow-name-reference="Syriac-Flow"/>
      </fo:flow-source-list>
   <fo:flow-target-list>
      <fo:region-name-specifier
        region-name-reference="Region-Left"/>
   </fo:flow-target-list>
 </fo:flow-assignment>
 </fo:flow-map>
```

e la parte dove si specifica tramite markup le parti vere e proprie di flussi da sincronizzare:

```
<fo:flow flow-name=?Arabic-Flow?>
   ?
   <fo:block dependency-content-begin=?Key_A?>
       And Nadan went to Haiqâr, and said to him,
       ?W?allah, O my uncle! The king verily
       rejoiceth in thee for having done what he
       commanded thee.
   </fo:block>
   <fo:block>
       And now he hath sent me to thee that thou
       mayst dismiss the soldiers to their duties
       and come thyself to him with thy handsbound
       behind thee, and thy feet chained, that the
       ambassadors of Pharaoh may see this, and
       that the king may be feared by them and by
       their king?.
   </fo:block>
   <fo:block>
       And Nadan took him and went with him to the
       king.
   </fo:block>
   <fo:block dependency-content-end=?Key_A?>
       Then answered Haiqâr and said, ?To hear is
       to obey.?
   </fo:block>
```

```
  ?
</fo:flow>
```

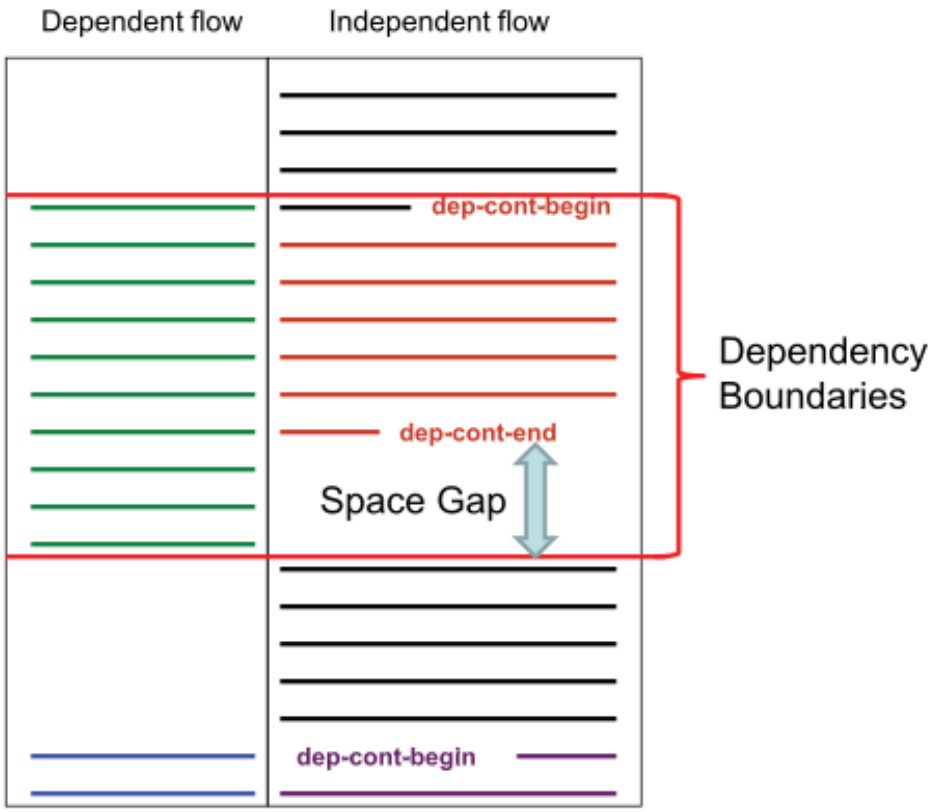



Figura 1.9: Output teorico di XSL-FO 2.0

Detto questo comunque la versione 2.0 di XSL-FO è solo in forma di specifica come accennato nei paragrafi precedenti. L'approccio pratico descritto precedentemente basato su XSL-FO 1.0 invece, è tutt'ora usato e mentre da una parte offre un ampia gamma di possibilità per la generazione del layout (compreso quello a fronte) presenta notevoli *trade-off*, questi si manifestano nella difficile aggiornabilità e personalizzabilità dei template XSL poiché apportare una modifica ai template non è molto differente dal riscriverli completamente da capo,inoltre essi sono strettamente dipendenti dal tipo di layout che si vuole ottenere e quindi anche dal dominio dei contenuti

da formattare quindi si necessita di template ad hoc per soddisfare i requisiti di layout di ogni campo di contenuti compreso il testo a fronte. Infine il designer nello scrivere i suddetti template oltre alle abilità di design richieste dalla sua professione necessita anche di avere conoscenze specialistiche di XSL e di FO, che non è un requisito trascurabile. Per questo si è pensato di rivedere l'approccio al problema da un altra angolazione, passando da un approccio procedurale, basato su template come quello usato con XSL-FO a uno di tipo dichiarativo che va a minimizzare proprio gli aspetti negativi sopra citati legati all'approccio procedurale.

Il contributo di questo lavoro intende essere duplice proponendo un approccio dichiarativo al problema del layout automatico e al contempo andare a coprire la carenza in questo settore di supporti specifici per la gestione e generazione di layout specifico per il testo a fronte, il tutto raccolto e nella libreria javascript sviluppata in questo lavoro, Sider.

# Capitolo 2

# Sider

Nel seguente capitolo si introdurrà la soluzione proposta con Sider e le sue principali caratteristiche di funzionamento seguite da alcuni esempi di applicazione del tool, per poi passare a un breve raffronto delle sue varie applicazioni.Infine verrà descritta esaustivamente la sintassi vera e propria per il suo utilizzo.

## 2.1 Caratteristiche

Sider è uno strumento per la formattazione automatizzata del testo a fronte, che mette a disposizione dell'utente varie funzionalità aggiuntive al fine di fornire un approccio dichiarativo e quindi minimizzare la ridondanza di markup nel caso specifico in cui si necessita di una gestione specifica per il raffronto di due o più testi.

Prima di passare a elencare le funzionalità del tool è necessario introdurre alcuni concetti di base per la comprensione delle sue funzionalità. Sider prende in input un contenitore primario che contiene un elenco di alberi (o sotto contenitore) ogni albero è composto dalla sequenza di elementi contenenti i blocchi di testo e/o il contenuto vero e proprio da allineare, e ognuno di essi rappresenta una delle diverse sorgenti di contenuti elaborate dal tool. Sider sincronizza gli elementi delle varie sorgenti sulle righe di una griglia prodotta

in output, questa "griglia" presenta un numero di colonne pari al numero di sorgenti di input e un numero di righe pari alla colonna (o sorgente di input) che ha il maggior numero di elementi. Un altro concetto fondamentale per la comprensione delle funzionalità del tool sono i livelli di allineamento, con questi ci si riferisce a tutte le righe della griglia costituite dagli elementi allineati corrispondenti ai requisiti richiesti dall'utente; mentre un sottolivello sono tutti gli elementi di ogni colonna compresi tra una riga di allineamento e l'altra del livello precedente.


Figura 2.1: Rappresentazione dei livelli di allineamento

Le principali funzionalità di Sider sono:

1. Allineamento parallelo di testi o alberi di contenuti misti, da 2 fino a un massimo di 12 contemporaneamente.

2. Possibilità di scelta tra tre criteri per l'allineamento tra gli elementi all'interno degli alberi:

   - Allineamento per nome degli elementi(o il nome del tag xml).
   - Allineamento per valore attributo interno agli elementi.

- Allineamento per attributo specificato da utente contenente una numerazione progressiva in base alla quale eseguire la sincronizzazione degli elementi.

Questi parametri sono specificabili nell'intestazione dell'elemento contenente gli alberi da allineare sotto forma di attributi specifici di Sider descritti in dettaglio più avanti in questo capitolo, inoltre si possono specificare regole di allineamento avanzate combinando a coppie i criteri di allineamento di base sopra elencati.

3. Supporto per allineamento su più livelli, che permette di allineare a loro volta tutti gli elementi compresi tra le righe del livello precedente che non hanno avuto riscontri di allineamento.

4. Possibilità di specificare l'allineamento verticale e orizzontale degli elementi disposti all'interno delle celle della griglia.

5. Possibilità di scelta fra due tipi diversi di layout della griglia:

   - Disposizione degli elementi per cella ovvero ogni set di elementi corrispondenti alle direttive di allineamento vengono disposti su una riga a se stante.
   - Disposizione degli elementi per blocco,in cui tutti gli elementi contigui corrispondenti allo stesso livello di allineamento vengono disposti in blocco in una singola cella della griglia formando cosi una riga di celle contenenti tutti gli elementi contigui associati con il corrente livello di allineamento.

6. Possibilità di eseguire l'allineamento ricorsivamente all'interno di ogni elemento delle sorgenti di input a patto che sia formattato correttamente per l'input di Sider. Ottenendo così delle griglie allineate e annidate l'una dentro l'altra.

7. Funzionalità di debugging integrata, che permette di visualizzare informazioni sull'esecuzione del tool al fine di agevolare l'utente nell'interazione con esso.

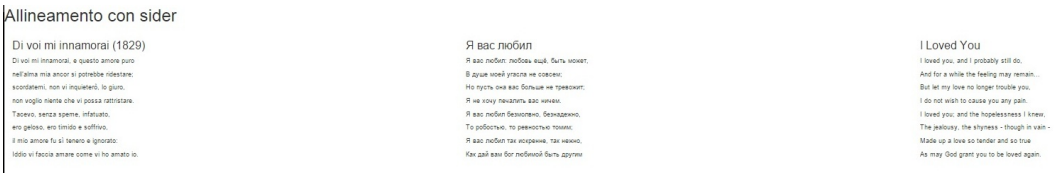



Figura 2.2: risultato dell'output di Sider visualizzato su browser

Con Sider si incapsula tutto il contenuto di ogni singola colonna (composta da blocchi di testo o quant'altro) da allineare all'interno di un singolo contenitore "div", successivamente si specifica nel contenitore principale (che è quello che contiene tutti i contenitori colonna da allineare) tramite alcuni attributi usati come parametri di ingresso da Sider le modalità di allineamento desiderate e il tool genera automaticamente in output al posto del codice html sorgente del codice html automaticamente formattato in modo da ottenere il layout a fronte allineato sulle parti di contenuto desiderato con strutture anche complesse e annidate.

```
<div
     sbs:containersSelectors="h3,p"
     sbs:align="center-left"
     sbs:margin-collapse="true">
    <div>
        <h3>Di voi mi innamorai (1829)</h3>
        <p>Di voi mi innamorai, e questo amore puro</p>
        <p>nell?alma mia ancor si potrebbe ridestare;</p>
        <p>scordatemi, non vi inquieterò, lo giuro,</p>
        <p>non voglio niente che vi possa rattristare.</p>
        <p>Tacevo, senza speme, infatuato,</p>
        <p>ero geloso, ero timido e soffrivo,</p>
        <p>il mio amore fu sì tenero e ignorato:</p>
        <p>Iddio vi faccia amare come vi ho amato io.</p>
    </div>
    <div>
        <h3>? ??? ?????</h3>
        <p>? ??? ?????: ?????? ???, ???? ?????,</p>
        <p>? ???? ???? ?????? ?? ??????;</p>
        <p>?? ????? ??? ??? ?????? ?? ????????;</p>
        <p>? ?? ???? ???????? ??? ?????.</p>
        <p>? ??? ????? ?????????, ??????????,</p>
        <p>?? ????????, ?? ????????? ?????;</p>
```

```
        <p>? ??? ????? ??? ????????, ??? ?????,</p>
        <p>??? ??? ??? ??? ??????? ???? ??????</p>
    </div>
    <div>
        <h3>I Loved You</h3>
        <p>I loved you, and I probably still do,</p>
        <p>And for a while the feeling may remain...</p>
        <p>But let my love no longer trouble you,</p>
        <p>I do not wish to cause you any pain.</p>
        <p>I loved you; and the hopelessness I knew,</p>
        <p>The jealousy, the shyness - though in vain -</p>
        <p>Made up a love so tender and so true</p>
        <p>As may God grant you to be loved again.</p>
    </div>
</div>
```

Il tool prende in input un insieme di elementi html con una struttura ad albero canonizzata in base a dei criteri prefissati e degli attributi contenenti una sintassi concepita ad hoc per il testo a fronte. Entrando più nello specifico l'albero di input è costituito da un contenitore (un "div") globale contenente gli attributi usati per le direttive di formattazione e molteplici sotto contenitori (altri "div") figli diretti del contenitore sopra citato ,uno per ogni colonna di contenuto da allineare fino a un massimo di 12 testi. Da notare che l'esempio sopra citato parte dal presupposto di non avere un input già preformattato secondo una struttura ad albero e si sottintende che l'utente debba prendere i blocchi di contenuti uno ad uno e formattarli adeguatamente a mano, invece partendo da un file xml, formattare i contenuti tramite un xslt in maniera compatibile con l'input di Sider è immediato. Successivamente Sider applica le direttive di allineamento tra i figli diretti dei sotto contenitori in questione lasciando inalterati eventuali elementi annidati, modificando l'ordine solo degli elementi sul primo livello di annidamento. Infine restituisce in output sulla pagina html in oggetto al posto dell'albero di input un altro albero html rielaborato in modo tale da ottenere un allineamento dei contenuti dell'albero di input secondo i parametri specificati negli attributi di partenza nel modo meno invasivo possibile per il documento e del layout finale desiderato(ovvero del markup html e css pregresso).

Come si può notare nelle funzionalità sopra introdotte, il contributo che il tool fornisce rispetto agli approcci tradizionali per il testo a fronte sono molteplici, derivanti dai vantaggi correlati ad un approccio dichiarativo al

problema con un drastico snellimento del codice richiesto per una presentazione dei contenuti, e risultante in un notevole decremento del carico di lavoro per l'utente finale e molte meno conoscenze prerequisito richieste al designer per l'utilizzo del tool rispetto all'approccio basato su xsl.

## 2.2 Esempi e casi d'uso

Per chiarire ulteriormente i vari utilizzi possibili del tool segue una rassegna dei principali casi d'uso realizzati con le diverse varianti sintattiche disponibili, essi sono costituiti dall'output visualizzato dal browser, le direttive di Sider nell'intestazione del contenitore principale ed una loro descrizione.

### 2.2.1 Testo di legge

In questo esempio si raffronta un pdf di un testo di legge(figura 2.3) e l'output di Sider su pagina web(figura 2.4).La pagina pdf è il risultato della renderizzazione di xsl-fo tramite un motore di formattazione,che viene generato in output dalla conversione di un xml sorgente tramite un file xslt, mentre l'output di Sider è stato generato a partire da un albero html formattato manualmente dai contenuti del pdf. Il markup usato per l'allineamento dell'albero html con Sider è il seguente (l'albero dei contenuti verrà omesso per brevità):

```
<div
                              sbs:containersSelectors="@center,@segue"
                              sbs:align="bottom-center,top-left,top-left"
                              sbs:margin-collapse="true">
                    <div>
                                  <!--ELEMENTI COLONNA 1-->
            </div>
             <div>
                                <!--ELEMENTI COLONNA 2-->
            </div>
< /div>
```

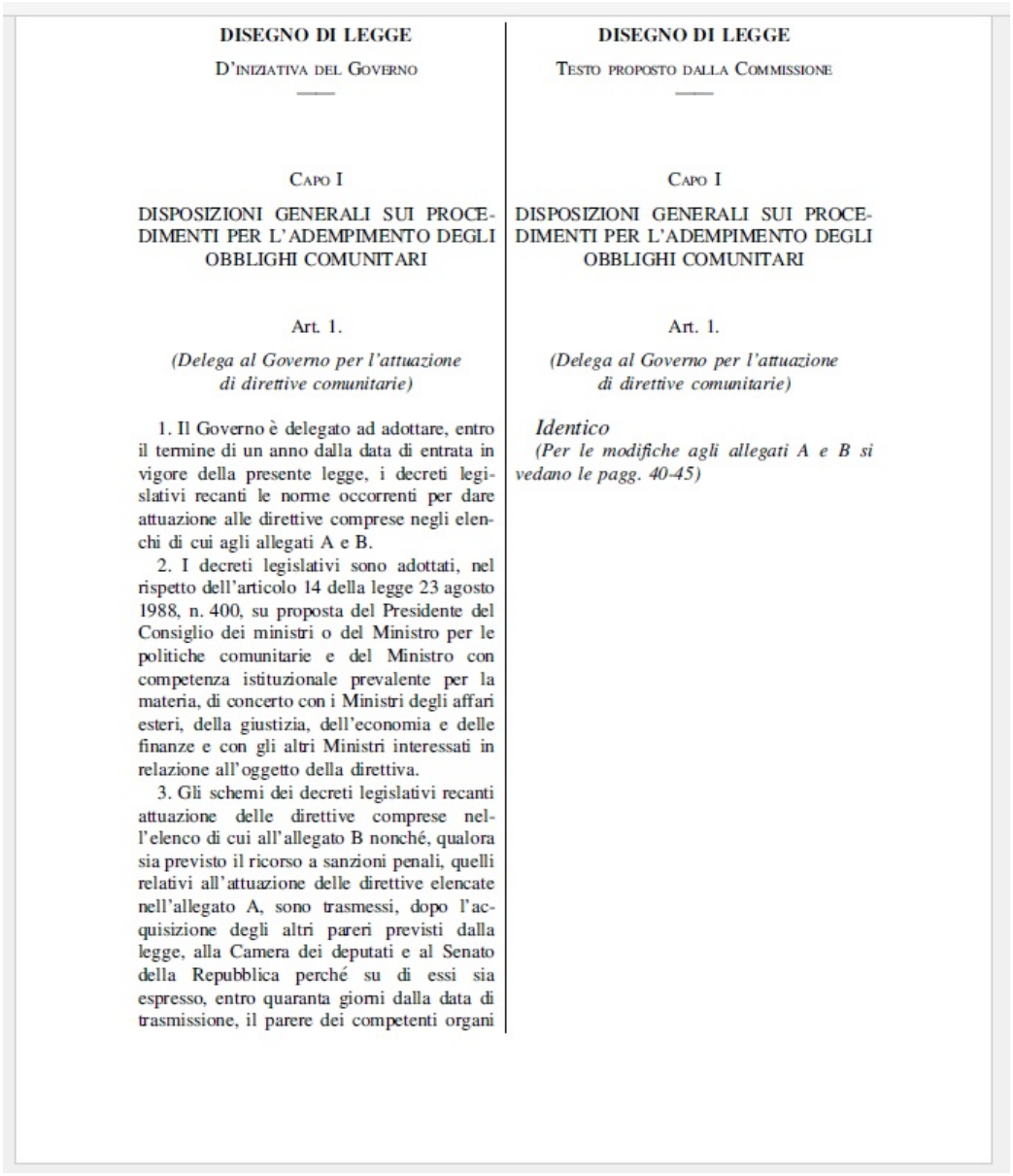
In questo caso per semplicità si sono usati solo due livelli di allineamento il primo attribuito agli elementi centrali come per esempio gli elementi di tipo "Capo x" e quelli di tipo "Art. x" che essendo nell'albero sorgente speculari

sono stati usati come ancora principale per l'allineamento, il secondo livello ovvero gli elementi di classe "segue" rappresentano i punti di allineamento sulla rottura della pagina ("(Segue: Testo d?iniziativa del Governo)") che è stato volutamente inserito per emulare il più possibile il layout originale, ma su pagina web non avendo l'esigenza di spezzare le pagine si potrebbe benissimo omettere direttamente l'elemento. C'è anche un terzo livello di allineamento che non è esplicitamente dichiarato nella sintassi e viene applicato in ogni caso da Sider per parificare l'allineamento dei contenuti, dove vengono allineati tra di loro tutti gli elementi compresi tra quelli dei precedenti livelli di allineamento solo in base al loro ordine di comparizione nell'albero anche se in questo caso avendo specificato la compressione dei blocchi contigui sullo stesso livello tramite la direttiva "margin-collapse" l'allineamento avviene per incapsulamento di questi ultimi in contenitori unici che contengono tutti gli elementi del livello corrente per quella colonna, a loro volta inclusi nei contenitori di tipo riga ("row") di Bootstrap.

Per ogni livello di allineamento è stato specificato tramite "align" il posizionamento interno degli elementi su ogni cella quindi in basso al centro per gli elementi sul livello "center" ,in alto a sinistra per tutti gli altri.

Avendo a disposizione un albero di ingresso formattato in maniera più elaborata si potrebbero ottenere layout finali molto più articolati dell'esempio suddetto che è solo a fini dimostrativi.

C'è da dire infine che è stato fatto qualche ritocco manuale in css per la parte tipografica del documento ma non è rilevante per l'allineamento finale dell'output.



**DISEGNO DI LEGGE**

D'INIZIATIVA DEL GOVERNO

CAPO I

DISPOSIZIONI GENERALI SUI PROCEDIMENTI PER L'ADEMPIMENTO DEGLI OBBLIGHI COMUNITARI

Art. 1.

*(Delega al Governo per l'attuazione di direttive comunitarie)*

1. Il Governo è delegato ad adottare, entro il termine di un anno dalla data di entrata in vigore della presente legge, i decreti legislativi recanti le norme occorrenti per dare attuazione alle direttive comprese negli elenchi di cui agli allegati A e B.

2. I decreti legislativi sono adottati, nel rispetto dell'articolo 14 della legge 23 agosto 1988, n. 400, su proposta del Presidente del Consiglio dei ministri o del Ministro per le politiche comunitarie e del Ministro con competenza istituzionale prevalente per la materia, di concerto con i Ministri degli affari esteri, della giustizia, dell'economia e delle finanze e con gli altri Ministri interessati in relazione all'oggetto della direttiva.

3. Gli schemi dei decreti legislativi recanti attuazione delle direttive comprese nell'elenco di cui all'allegato B nonché, qualora sia previsto il ricorso a sanzioni penali, quelli relativi all'attuazione delle direttive elencate nell'allegato A, sono trasmessi, dopo l'acquisizione degli altri pareri previsti dalla legge, alla Camera dei deputati e al Senato della Repubblica perché su di essi sia espresso, entro quaranta giorni dalla data di trasmissione, il parere dei competenti organi

**DISEGNO DI LEGGE**

TESTO PROPOSTO DALLA COMMISSIONE

CAPO I

DISPOSIZIONI GENERALI SUI PROCEDIMENTI PER L'ADEMPIMENTO DEGLI OBBLIGHI COMUNITARI

Art. 1.

*(Delega al Governo per l'attuazione di direttive comunitarie)*

*Identico*

*(Per le modifiche agli allegati A e B si vedano le pagg. 40-45)*

Figura 2.3: Testo di legge originale formattato con XSL-FO

**DISEGNO DI LEGGE**

D'iniziativa del Governo

---

Capo I

**DISPOSIZIONI GENERALI SUI PROCEDIMENTI PER L'ADEMPIMENTO DEGLI OBBLIGHI COMUNITARI**

Art. 1.

*(Delega al Governo per l'attuazione di direttive comunitarie)*

1. Il Governo e' delegato ad adottare, entro il termine di un anno dalla data di entrata in vigore della presente legge, i decreti legislativi recanti le norme occorrenti per dare attuazione alle direttive comprese negli elenchi di cui agli allegati A e B.

2. I decreti legislativi sono adottati, nel rispetto dell'articolo 14 della legge 23 agosto 1988, n. 400, su proposta del Presidente del Consiglio dei ministri o del Ministro per le politiche comunitarie e del Ministro con competenza istituzionale prevalente per la materia, di concerto con i Ministri degli affari esteri, della giustizia, dell'economia e delle finanze e con gli altri Ministri interessati in relazione all'oggetto della direttiva.

3. Gli schemi dei decreti legislativi recanti attuazione delle direttive comprese nell'elenco di cui all'allegato B nonche', qualora sia previsto il ricorso a sanzioni penali, quelli relativi all'attuazione delle direttive elencate nell'allegato A, sono trasmessi, dopo l'acquisizione degli altri pareri previsti dalla legge, alla Camera dei deputati e al Senato della Repubblica perche' su di essi sia espresso, entro quaranta giorni dalla data di trasmissione, il parere dei competenti organi parlamentari. Decorso tale termine, i decreti sono emanati anche in mancanza del parere. Qualora il termine previsto per il parere dei competenti organi parlamentari scada nei trenta giorni che precedono la scadenza dei termini previsti ai commi 1 o 4 o successivamente, questi ultimi sono prorogati di novanta giorni.

*(Segue: Testo d'iniziativa del Governo)*

4. Entro un anno dalla data di entrata in vigore di ciascuno dei decreti legislativi di cui al comma 1, nel rispetto dei princi'pi e criteri direttivi fissati dalla presente legge, il Governo puo' emanare, con la procedura indicata nei commi 2 e 3, disposizioni integrative e correttive dei decreti legislativi emanati ai sensi del comma 1.

5. In relazione a quanto disposto dall'articolo 117, quinto comma, della Costituzione, i decreti legislativi eventualmente adottati nelle materie di competenza legislativa delle regioni e delle province autonome di Trento e di Bolzano, entrano in vigore, per le regioni e le province autonome nelle quali non sia ancora in vigore la propria normativa di attuazione, alla data di scadenza del termine stabilito per l'attuazione della normativa comunitaria e perdono comunque efficacia a decorrere dalla data di entrata in vigore della normativa di attuazione adottata da ciascuna regione e provincia autonoma nel rispetto dei vincoli derivanti dall'ordinamento comunitario e, nelle materie di competenza concorrente, dei princi'pi fondamentali stabiliti dalla legge.

Art. 2.

*(Principi e criteri direttivi generali della delega legislativa)*

1. Salvi gli specifici princi'pi e criteri direttivi stabiliti dalle disposizioni di cui al capo II ed in aggiunta a quelli contenuti nelle direttive da attuare, i decreti legislativi di cui

*(Segue: Testo d'iniziativa del Governo)*

all'articolo 1 sono informati ai seguenti princi'pi e criteri direttivi generali: a) le amministrazioni direttamente interessate provvedono all'attuazione dei decreti legislativi con le ordinarie strutture amministrative; b) per evitare disarmonie con le discipline vigenti per i singoli settori interessati dalla normativa da attuare, sono introdotte le occorrenti modifiche o integrazioni alle discipline stesse, fatte salve le materie oggetto di delegificazione ovvero i procedimenti oggetto di semplificazione amministrativa; c) salva l'applicazione delle norme penali vigenti, ove necessario per assicurare l'osservanza delle disposizioni contenute nei decreti legislativi, sono previste sanzioni amministrative e penali per le infrazioni alle disposizioni dei decreti stessi. Le sanzioni penali, nei limiti, rispettivamente, dell'ammenda fino a 103.291 euro e dell'arresto fino a tre anni, sono previste, in via alternativa o congiunta, solo nei casi in cui le infrazioni ledano o espongano a pericolo interessi generali dell'ordinamento interno, ivi compreso l'ecosistema. In tali casi sono previste: la pena dell'ammenda alternativa all'arresto per le infrazioni che espongano a pericolo o danneggino l'interesse protetto; la pena dell'arresto congiunta a quella dell'ammenda per le infrazioni che rechino un danno di particolare gravita'. La sanzione

**DISEGNO DI LEGGE**

Testo proposto dalla Commissione

---

Capo I

**DISPOSIZIONI GENERALI SUI PROCEDIMENTI PER L'ADEMPIMENTO DEGLI OBBLIGHI COMUNITARI**

Art. 1.

*(Delega al Governo per l'attuazione di direttive comunitarie)*

Identico

*(Per le modifiche agli allegati A e B si vedano le pagg. 40-45)*

*(Segue: Testo proposto dalla Commissione)*

Art. 2.

*(Principi e criteri direttivi generali della delega legislativa)*

Identico

*(Segue: Testo proposto dalla Commissione)*

Figura 2.4: Testo di legge formattato con Sider su Browser

### 2.2.2 Poesie


Figura 2.5: Poesia allineata e formattata con Sider

```
<div class="example"
             sbs:containersSelectors="h3,p"
             sbs:align="center-left"
             sbs:margin-collapse="true">
```

Questo caso è lo stesso usato per l'esempio introduttivo del tool e qui avviene un allineamento su due livelli, il primo allinea tra di loro tutti gli elementi titolo "h3" in questo caso ogni colonna ne ha solo uno, ma ci possono essere casi di colonne contenenti poesie multiple dove può essere utile allineare prima per un elemento che funge da delimitatore dei contenuti delle varie colonne. Nel secondo livello si sincronizzano sulla stessa riga tutti gli elementi di classe tipo "p" in ordine di comparizione nell'albero di input. Da notare che essendo questo un esempio dimostrativo, l'albero di ingresso è molto semplice e si potrebbero avere gli stessi identici risultati solo specificando il parametro "*" che allinea tutti gli elementi su una stessa riga solo secondo il loro ordine di comparizione all'interno del contenitore colonna. L'allineamento di ogni elemento in ogni cella è al centro a sinistra su tutti i livelli e anche qui viene specificata una compressione dei blocchi sul medesimo livello di allineamento.

Nel prossimo caso di poesia si vuole evidenziare la differenza del layout finale che si ottiene tramite l'uso della direttiva "margin-collapse".

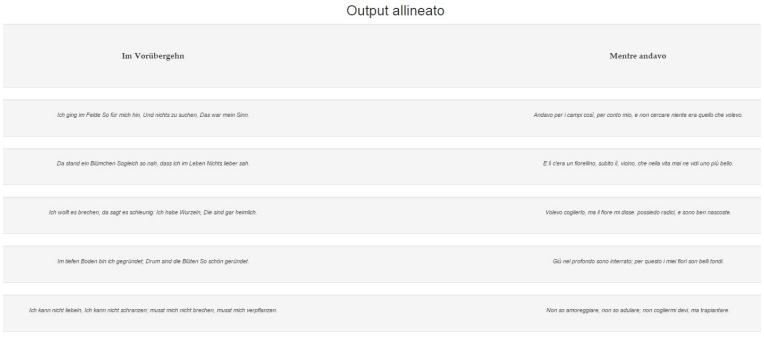



Figura 2.6: Poesia allineata e formattata senza compressione elementi


Figura 2.7: Poesia allineata e formattata con compressione elementi

| *Figura*2.6 | *Figura*2.7 |
|---|---|
| `<div class="example"`<br>`sbs:containersSelectors="strong,p"`<br>`sbs:align="center-center"`<br>`sbs:margin-collapse="false"`<br>`sbs:markerClasses="titolo,paragrafo">` | `<div class="example"`<br>`sbs:containersSelectors="strong,p"`<br>`sbs:align="center-center"`<br>`sbs:margin-collapse="true"`<br>`sbs:markerClasses="titolo,paragrafo">` |

Tabella 2.1: Raffronto intestazione per gli allineamenti delle due poesie

In ambedue i casi si sono usate le stesse direttive di allineamento che sono su due livelli, il primo allinea gli elementi di tipo "strong" tra di loro e il secondo gli elementi di tipo "p" compresi tra quelli del livello precedente ovvero gli elementi di tipo "strong". E' stato specificato inoltre che in ambedue i casi gli elementi in ogni cella siano centrati. L'unica e fondamentale differenza sta nel fatto che è stata cambiata la direttiva "margin-collapse" da valore "true" a "false", quindi mentre impostando l'attributo a valore "true" si ha una compressione dei blocchi sullo stesso livello in una riga sola in cui

ogni cella della riga contiene tutti gli elementi di quel livello per quella colonna, la direttiva "false" dice al tool di creare una riga per ogni set di elementi corrisposti dai criteri di allineamento, in cui ogni cella della riga contiene solo un elemento della colonna di origine. In questo esempio inoltre è stato introdotto un nuovo attributo :"markerClasses", che di perse non ha nessuna interazione con il layout di output su browser, ma permette di marcare ogni livello di allineamento associato secondo l'ordine di comparizione (per esempio agli elementi allineati del livello "strong" viene associata la *keyword* titolo mentre a quelli sul livello "p" la *keywork* paragrafo) con una classe specifica scelta dall'utente; agevolando in questo modo ulteriori ritocchi in css specifici per ogni livello di allineamento, tramite l'uso dei selettori di classe precedentemente specificati.

### 2.2.3 Canzoni


Figura 2.8: Canzone formattata con Sider

```
<div class="example"
          sbs:containersSelectors="#AncoraNumericaProgressiva"
          sbs:align="top-left"
          sbs:margin-collapse="false">
```

In questo esempio è stata usata una canzone e la sua traduzione in italiano, qui l'allineamento è stato realizzato tramite un attributo personaliz-

zato chiamato "AncoraNumericaProgressiva" aggiunto manualmente a ogni elemento da allineare ed usato come identificatore numerico univoco all'interno della rispettiva colonna di appartenenza. Questo permette all'utente di specificare direttamente a Sider in un preciso ordine quali blocchi allineare, questa funzionalità è progressiva, cioè parte dall'elemento con numero più basso presente in quel livello di allineamento e allinea progressivamente i contenuti sugli elementi delle colonne che hanno lo stesso identificatore. Qui gli elementi sullo stesso livello non sono stati collassati quindi c'è una riga per ogni set di elementi e l'allineamento interno alle celle è in alto a sinistra.

### 2.2.4 Sceneggiature

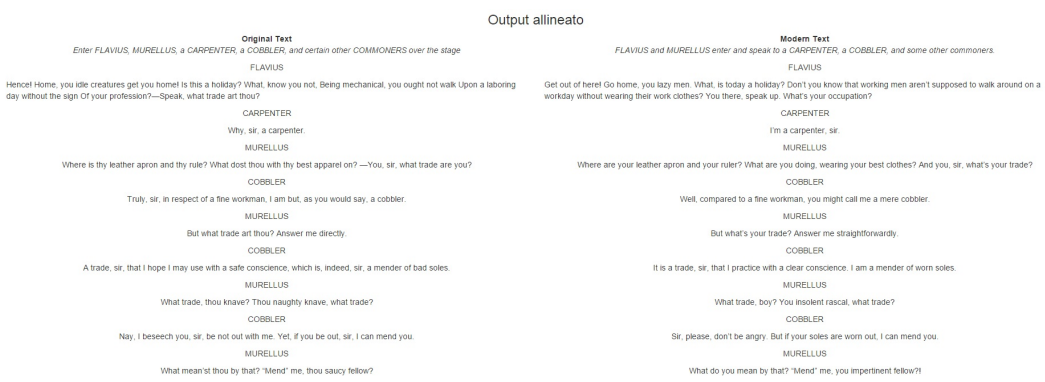



Figura 2.9: Sceneggiatura formattata con Sider

```
<div class="example"
             sbs:containersSelectors="@title,@azione"
             sbs:margin-collapse="true">
            <div>
                <strong class="title">Original Text</strong>
                <p class="azione">Enter FLAVIUS, MURELLUS, a CARPENTER, a COBBLER,
                 and certain other COMMONERS over the stage</p>
                <p class="character">FLAVIUS</p>
                <p>Hence! Home, you idle creatures get you home! Is this a holiday? What, know you not,
                Being mechanical, you ought not walk Upon a laboring day
                without the sign Of your profession??Speak, what trade art thou?</p>
                <p class="character">CARPENTER</p>
                <p>Why, sir, a carpenter.</p>
                <p class="character">MURELLUS</p>
                <p>Where is thy leather apron and thy rule? What dost thou with thy best apparel on?
                 ?You, sir, what trade are you?</p>
                <p class="character">COBBLER</p>
                <p>Truly, sir, in respect of a fine workman, I am but, as you would say, a cobbler.</p>
                <p class="character">MURELLUS</p>
                <p>But what trade art thou? Answer me directly.</p>
                <p class="character">COBBLER</p>
                <p>A trade, sir, that I hope I may use with a safe conscience,
                which is, indeed, sir, a mender of bad soles.</p>
                <p class="character">MURELLUS</p>
                <p>What trade, thou knave? Thou naughty knave, what trade?</p>
                <p class="character">COBBLER</p>
                <p>Nay, I beseech you, sir, be not out with me. Yet, if you be out, sir, I can mend you.</p>
                <p class="character">MURELLUS</p>
                <p>What mean?st thou by that? ?Mend? me, thou saucy fellow?</p>
            </div>
            <div>
                <strong class="title">Modern Text</strong>
                <p class="azione">FLAVIUS and MURELLUS enter and speak to a CARPENTER,
                 a COBBLER, and some other commoners.</p>
                <p class="character">FLAVIUS</p>
                <p>Get out of here! Go home, you lazy men. What, is today a holiday?
                  Don?t you know that working men aren?t supposed to walk around on a
                  workday without wearing their work clothes? You there, speak up.
                  What?s your occupation?</p>
                <p class="character">CARPENTER</p>
```

```
        <p>I?m a carpenter, sir.</p>
        <p class="character">MURELLUS</p>
        <p>Where are your leather apron and your ruler?
         What are you doing, wearing your best clothes?
         And you, sir, what?s your trade?</p>
        <p class="character">COBBLER</p>
        <p>Well, compared to a fine workman, you might call me a mere cobbler.</p>
        <p class="character">MURELLUS</p>
        <p>But what?s your trade? Answer me straightforwardly.</p>
        <p class="character">COBBLER</p>
        <p>It is a trade, sir, that I practice with a clear conscience. I am a mender of worn soles.</p>
        <p class="character">MURELLUS</p>
        <p>What trade, boy? You insolent rascal, what trade?</p>
        <p class="character">COBBLER</p>
        <p>Sir, please, don?t be angry. But if your soles are worn out, I can mend you.</p>
        <p class="character">MURELLUS</p>
        <p>What do you mean by that? ?Mend? me, you impertinent fellow?!</p>
    </div>
</div>
```

In questo caso è stato ritenuto rilevante includere tutto il sorgente dell'albero di input perché in questo esempio viene fatto uso esplicito del livello di allineamento globale "nascosto", che è quello che viene processato implicitamente da Sider dopo l'elaborazione di quelli esplicitamente dichiarati dall'utente. Qui avviene prima l'allineamento per elementi di classe "title" poi all'interno di questi vengono sincronizzati gli elementi di classe "azione" e successivamente gli elementi rimanenti tra questi ultimi vengono tutti allineati tra di loro in ordine di comparizione all'interno dell'albero (questo è il terzo livello processato implicitamente). In questo esempio ci sono anche elementi di classe "character" che rappresentano i nomi degli attori in questione, ma non vengono utilizzati per l'allineamento perché non necessario, poiché ogni colonna presenta una struttura speculare e non c'è bisogno di ulteriori direttive di allineamento che sono invece necessarie in casi dove la struttura delle colonne è più eterogenea. Inoltre si può notare che la direttiva di allineamento interno celle "align" è stata tralasciata ed il tool ha quindi applicato l'allineamento di default centrato su tutti gli elementi delle celle.

### 2.2.5 Parafrasi


Figura 2.10: Parafrasi formattata con Sider

```
<div class="example"
            sbs:containersSelectors="#canto"
            sbs:align="top-left,center-center"
            sbs:margin-collapse="true"
            sbs:markerClasses="Mark1,centrato">
```

Qui l'allineamento primario avviene tramite identificatore numerico progressivo rappresentato dall'attributo "canto" mentre il resto degli elementi

sono lasciati in gestione al livello di allineamento implicito di Sider (che è quello che allinea gli elementi restanti non corrispondenti a nessuna direttiva impartita dai selettori) che comprendono il titolo del canto e la tipologia di contenuto della colonna. Quindi poiché è stato specificato il collassamento delle righe sullo stesso livello gli elementi del secondo livello "nascosto" sono stati compressi sulla stessa riga, cosa che non si nota sul primo livello (quello che contiene i testi veri e propri) perché tra ogni set di elementi selezionati c'è sempre qualche altro elemento del secondo livello che li separa e quindi non ci sono elementi contigui da collassare. L'allineamento interno delle celle del primo livello è in alto a sinistra mentre quello interno del secondo è centrato sia verticalmente che orizzontalmente.

### 2.2.6 Traduzioni

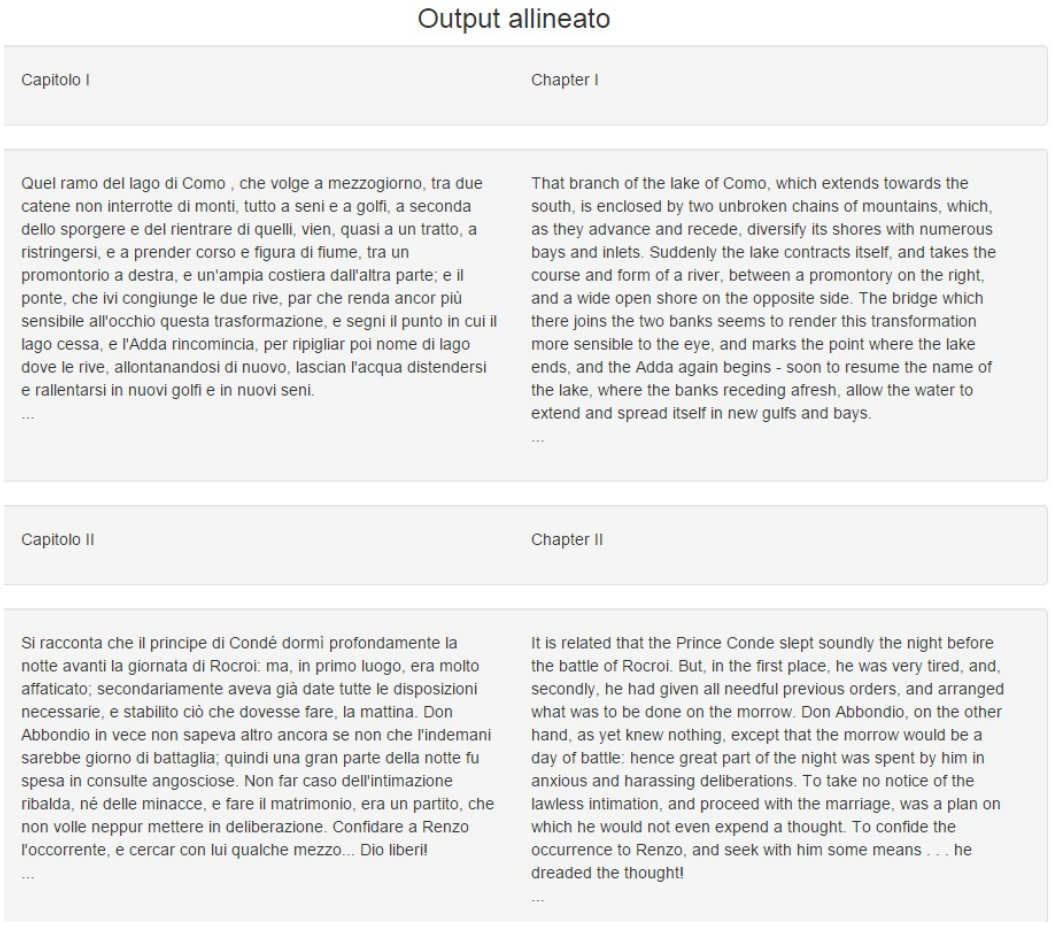



Figura 2.11: Traduzione formattata con Sider

```
<div class="example"
             sbs:containersSelectors="@capitolo"
             sbs:align="top-left"
             sbs:margin-collapse="false"
             sbs:markerClasses="Mark1">
```

In questo esempio si usa la classe "capitolo" per l'allineamento, gli elementi marcati con questa classe rappresentano le intestazioni di inizio capitolo che suddivide i vari contenuti, e tutto il resto degli elementi compresi tra questi sono sincronizzati tra di loro elemento per elemento dato che è stato specificato l'allineamento senza il collasso di questi ultimi. L'allineamento interno delle celle è in alto a sinistra per tutti i livelli, poiché se si specifica

solo il primo livello di allineamento interno esso viene ereditato da tutti i livelli restanti, compreso quello "nascosto".

### 2.2.7 Funzionalità del Debugger


Figura 2.12: Esempio di schermata di debug di Sider

Riprendiamo l'esempio usato delle poesie per illustrare le funzionalità di debug. In questo esempio sono state attivate le funzionalità di debug di Sider passando alla funzione chiamante del tool una combinazione di 3 caratteri sotto forma di stringa che rappresentano a loro volta la verbosità del debugger stesso:

```
AlignDocument("iwe");
```

Si nota che ci sono 2 differenze principali dal normale output di Sider:

La prima è la colorazione degli elementi sulla griglia di output, gli elementi del medesimo colore rappresentano gli elementi appartenenti allo stesso livello di allineamento o quelli che Sider processa come tali.

La seconda cosa è la finestra collassabile in basso al documento, questa contiene un riquadro per ogni albero allineato da Sider, il quale contiene a sua volta un link ipertestuale cliccabile con il nome dell'indirizzo *Xpath* associato

all'albero di appartenenza del riquadro e una lista di righe di testo colorate a seconda della tipologia di avviso in esse contenute.

## 2.3 Valutazione

La prima cosa in evidenza tra le varie soluzioni è che il layout che si ottiene è molto simile, si tiene a specificare che non c'è nulla di più che non si possa ottenere con Sider rispetto a una formattazione manuale dei contenuti o a una soluzione procedurale, anzi inevitabilmente a causa dell'impossibilità di una gestione capillare dei vari casi con un approccio dichiarativo, si perde parte del controllo finale sul layout(come per esempio l'impossibilità di creare un separatore tra le colonne). Quindi si può dire che la qualità del layout finale con l'utilizzo di Sider tende a degradarsi quanto più il layout desiderato si allontana dalla casistica di soluzioni coperte dal tool.

In seconda, l'*overhead* di markup da aggiungere ai contenuti o di codice esterno al documento è drasticamente ridotta con Sider, dato che la parte ridondante (e più onerosa per l'utente) è generata automaticamente in Output, questa differenza diventa sempre più marcata con l'aumentare e la diversificazione dei contenuti. Purtroppo in qualsiasi caso è richiesta sempre un interazione da parte dell'utente non eliminabile, come per esempio l'incapsulamento delle varie parti di contenuto nei tag html e/o l'aggiunta di attributi o identificatori univoci per elemento; a meno che non si parta da contenuti già preformattati, ma in genere anche in quel caso si necessita comunque di un aggiustamento manuale o programmatico da parte dell'utente finale che può essere più o meno oneroso a seconda del layout finale che si vuole ottenere. Comunque sia, la "distanza" del documento originale e il documento di input preformattazione è drasticamente ridotta con l'approccio dichiarativo usato rispetto a quello procedurale o a quello manuale.

Infine parlando di complessità di allineamento, Sider supporta allineamenti su livelli multipli che permettono la realizzazione di layout a fronte anche molto complessi con relativamente poco sforzo da parte dell'utente ri-

spetto all'utilizzo dei metodi tradizionali che richiedono di parecchio markup di contorno addizionale o di ulteriore codice negli approcci procedurali in cui l'utente deve avere già presente un quadro globale approssimativo del layout di output desiderato. Senza contare che se si vogliono provare diverse configurazioni per l'allineamento di uno stesso documento l'utente è costretto a riformulare totalmente (o quasi) il markup e/o il codice, mentre con Sider basta cambiare solo qualche parametro e anche in questo caso il carico di lavoro risparmiato all'utente non è trascurabile.

## 2.4 Utilizzo

Per utilizzare Sider bisogna includere nel documento in questione 3 librerie:

- La libreria css di Bootstrap nell'intestazione ovvero:

  ```
  <link rel="stylesheet" href="../../lib/bootstrap-3.3.4-dist/css/bootstrap.min.css">
  ```

  dove il collegamento "href" può essere sia un uri che un file contenente la libreria css versione 3.3.4.

- La libreria di Jquery per le funzionalità aggiuntive di javascript nella sezione script del documento su cui si basa Sider:

  ```
  <script src="../../lib/jquery-1.11.3.min.js"></script>
  ```

- La libreria di Sider contenente le funzionalità vere e proprie descritte in questo elaborato, da includesi sempre nella sezione script:

  ```
  <script src="../../siderJS.js"></script>
  ```

Una volta aggiunte queste 3 librerie si procede al markup del documento, specificando a Sider quali sono i contenitori preformattati adeguatamente per l'elaborazione. Successivamente va chiamata dalla sezione script la funzione di Sider che si desidera utilizzare in base alle proprie esigenze, Sider ha 3 tipi diversi di chiamata:

- `AlignDocument(Debug);`

  Allinea tutti gli elementi nel documento contenenti gli attributi direttive di Sider

- `NestedElementsAlign(element,Debug);`

  Allinea in maniera ricorsiva tutti gli elementi contenenti gli attributi direttive di Sider annidati dentro l'elemento DOM "element" passato da parametro e l'elemento stesso.

- `AlignElement(DivContainer,Debug);`

  Allinea solo l'elemento DOM "element" passato da parametro.

Il parametro Debug comune a tutte 3 le chiamate specifica se attivare o no le funzionalità di Debug (Vedere il paragrafo 2.4.2 per approfondimenti).

Il tool processa solo i contenitori che posseggono l'attributo "sbs:containersSelectors" e scarta tutti gli altri, anche se attivando le funzionalità di debugging fornisce degli avvisi per gli errori di sintassi più comuni ovvero dove viene rilevato un intento da parte dell'utente di utilizzo del tool ma in maniera impropriamente formulata.

Sider mette a disposizione 4 attributi html per le direttive utente di cui una obbligatoria per il funzionamento del tool e le altre 3 opzionali poiché in caso di una loro omissione verrà applicato un comportamento di default.

### 2.4.1 Sintassi

**Namespace** Per questioni di interoperabilità con altri domini e/o tools ogni attributo usato per le direttive è preceduto dal namespace specifico di Sider: "sbs", seguito dal carattere ":" per esempio:

```
sbs:containersSelectors=''@center,@segue''
```

Il prefisso del namespace "sbs:" è obbligatorio per ogni attributo usato da Sider; le direttive senza namespace vengono totalmente ignorate per l'elaborazione dell'output (ma non dal debugger). Viene aggiunta inoltre nell'intestazione della pagina la definizione del namespace usato. Per chiarezza in seguito il prefisso del namespace per le direttive sintattiche verrà omesso.

**ContainersSelectors** L'attributo principale contenente le direttive di allineamento (o selettore), è costituita da un numero indefinito di regole di allineamento singole o a coppie separate dal carattere "," (o livelli di allineamento). Le singole regole di base possono essere di 3 tipi:

- Allineamento secondo il nome del tag degli elementi da allineare, in cui vengono allineati tutti gli elementi aventi lo stesso tipo di tag. Per esempio specificando containersSelectors="p" verranno allineati su quel livello tutti gli elementi di tipo "p".
- Allineamento per classe, ovvero specificando il carattere "@" seguito da un valore arbitrario verranno allineati tra di loro tutti gli elementi che presentano la classe specificata. Per esempio containersSelectors="@capitolo" sincronizzerà tra loro tutti gli elementi aventi la classe "capitolo" nella loro lista delle classi.
- Allineamento per identificatore univoco, in cui specificando il carattere "#" seguito da un nome arbitrario verranno allineati tutti gli elementi aventi come attributo il valore specificato in maniera progressiva secondo il valore contenuto nell'attributo stesso che viene usato quindi come ancora numerica progressiva. Per esempio containersSelectors="#sezione" sincronizzerà tra loro tutti gli

elementi con l'attributo sezione che hanno lo stesso valore numerico partendo dal valore in comune più basso. Quindi la sincronizzazione avverrà tra tutti gli elementi di ogni colonna che hanno l'attributo sezione="1", poi quelli con sezione="2" e così via; se c'è un valore dell'attributo mancante su una colonna questo verrà saltato e Sider passerà direttamente ad allineare il prossimo valore in comune a tutte le colonne, questo procedimento viene ripetuto fino al raggiungimento del valore massimo in una delle colonne.

Inoltre le singole regole possono essere combinate a coppie di 2 sullo stesso livello permettendo all'utente di esprimere criteri di selezione degli elementi avanzati tramite l'uso dei caratteri "[" e "]" contenenti il secondo parametro di allineamento. Per esempio con la direttiva: containersSelectors="#sezione[div]" verranno sincronizzati tra di loro gli elementi delle colonne di tipo "div" che presentano l'attributo "sezione" secondo i valori contenuti in quest'ultimo, oppure con containersSelectors="@capitolo[div]" verranno sincronizzati gli elementi di tipo "div" ma che possiedono contemporaneamente la classe "capitolo" nella loro lista classi. L'ordine in cui i parametri vengono specificati è indifferente mentre non si possono specificare alcune combinazioni su una stessa regola come come per esempio div[p] che costituisce fondamentalmente un errore semantico. Per concatenare i livelli di allineamento si usa il carattere ",". Un livello di allineamento è l'insieme degli elementi selezionati con la regola associata a quel determinato livello, compresi tra i due elementi contigui selezionati nel livello precedente.

Quindi per esempio volendo allineare tutti gli elementi classe capitolo sul primo livello e successivamente all'interno di questi si vuole allineare sul secondo livello tra di loro tutti gli elementi progressivamente per l'attributo sezione di avrà:

```
containersSelectors="@capitolo,#sezione"
```

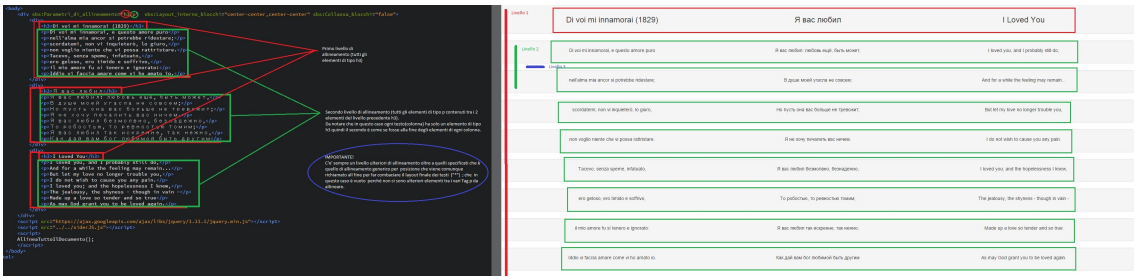



Figura 2.13: Livelli di allineamento

**Align** Questo attributo contiene le direttive per l'allineamento interno degli elementi nelle celle interne della griglia di output, ogni regola è associata al corrispondente livello di layout delle direttive di allineamento separate dal carattere ",". La singola regola contiene il parametro di allineamento verticale e quello di allineamento orizzontale separati dal carattere "-". Per il parametro verticale può assumere 3 valori:

- "top" per il posizionamento in alto nell'interno cella.
- "center" per il posizionamento al centro.
- "bottom" per il posizionamento in basso.

Il parametro orizzontale invece:

- "left" per il posizionamento a sinistra nell'interno cella.
- "center" per il posizionamento al centro.
- "right" per il posizionamento a destra.

Quindi avendo 2 livelli di allineamento per il selettore di allineamento e volendo allineare gli elementi del primo livello al centro e quelli del secondo in alto a sinistra si avrà:

```
align="center-center,top-left"
```

Se l'attributo align viene omesso gli elementi verranno centrati di default su tutti i livelli.

Inoltre se si specifica una sola regola per un solo livello questa verrà ereditata da tutti gli altri livelli di allineamento della griglia.

**Margin-collapse** L'attributo margin-collapse può assumere solo 2 valori "true" o "false" e rappresenta la modalità di compressione dei livelli.

- Con "false" ogni set di elementi corrisposti dal selettore vengono disposti su una riga diversa sul layout di output.
- Se invece viene usato "true" tutti gli elementi contigui appartenenti allo stesso livello di allineamento vengono collassati su una stessa riga ottenendo un layout finale molto differente dal precedente.

**markerClasses** L'attributo markerClasses non ha effetti diretti sul layout finale di output, ma aggiunge alle celle contenitori degli elementi appartenenti a un determinato livello di allineamento una classe con nome a discrezione dell'utente per successivi ritocchi grafici tramite css.

Anche qui come per il selettore e l'attributo align ogni classe che si vuole assegnare è separata da un carattere ",", ed è associata al corrispondente livello in base all'ordine di comparizione nell'attributo. Per esempio se si vogliono marcare 2 livelli, il primo con la classe "livello1" e il secondo con "livello2" si avrà:

```
markerClasses="livello1,livello2"
```

Come per align se si specifica la classe di un solo livello questa verrà applicata in maniera ereditaria su tutti i restanti livelli.

### 2.4.2 Debugger

Sider dispone di una funzionalità di debugging integrate che permette all'utente una migliore interazione con il tool. Si attiva passando come parametro alla funzione Javascript chiamante di Sider almeno uno dei caratteri usati per specificare la verbosità del tool. I caratteri del parametro sono 3:

- i - Mostra gli avvisi informativi generali del funzionamento interno del tool e sono rappresentati con le righe in blu.
- w - Mostra gli avvisi di avvertimento che non pregiudicano il funzionamento del tool ma solo che c'è qualche anomalia, sono rappresentati con le righe in giallo.
- e - Mostra i messaggi di errore che sono critici per il funzionamento del tool e non sono trascurabili, sono rappresentati con le righe di colore rosso.

Una volta attivato con uno o più dei suddetti parametri, nella finestra del browser compare in basso un riquadro collassabile(comprimibile/espandibile a seconda se il cursore sia nell'area della finestra) che contiene un contenitore intitolato con l'indirizzo *Xpath* dell'albero di allineamento associato con all'interno listati gli avvisi relativi all'albero in questione. L'indirizzo Xpath di ogni contenitore è cliccabile e centra la pagina del browser direttamente sull'output corrispondente di quel riquadro, evidenziandolo con un contorno rosso.

I controlli effettuati dal Debugger sull'istanza del tool sono i seguenti:

- Controllo correttezza sintattica sugli attributi usati da Sider e sui loro contenuti usati come direttive dal tool.
- Controllo direttive semantiche errate su selettore.
- Controllo correttezza sintattica dei namepace.
- Controllo degli stessi parametri di debug.
- Controllo sugli errori di funzionamento interno tramite il ridirezionamento delle eccezioni del javascript nei vari segmenti di codice direttamente su un avviso all'interno del riquadro di debug ad esso associato

Si faccia riferimento al paragrafo 2.2.7 per esempio di schermata di Debug.

# Capitolo 3

# Architettura

## 3.1 Tecnologie utilizzate

Le tecnologie utilizzate per lo sviluppo e su cui si basa il tool sono principalmente 3:

- Html + Css
- Javascript + Jquery
- Bootstrap

Sider è stato concepito appositamente per avere una resa grafica del testo a fronte su browser web, quindi l'uso di html e css è stata una scelta obbligatoria. Html(HyperText Markup Language) è usato per la formattazione e impaginazione di documenti ipertestuali disponibili nel World Wide Web sotto forma di pagine web mentre il css(Cascading Style Sheets ) definisce la formattazione grafica avanzata.

Per lo scopo che si prefigge il tool si ha bisogno di funzionalità di elaborazione dati non presenti in Html e css ed è stato quindi scelto di usare Javascript con Jquery. Javascript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione web lato client per la creazione, di siti web e applicazioni web, ed effetti dinamici

interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso (mouse, tastiera ecc...); Jquery invece è una sua libreria che mette a disposizione una serie di meccanismi sintattici che permettono di semplificare notevolmente la selezione, la manipolazione e la gestione degli eventi di oggetti *DOM* su html.

Bootstrap è un framework per lo sviluppo web ed è stato scelto tra i tanti disponibili in rete perché è largamente supportato, ricco di funzionalità e ben documentato; in particolare è stato scelto per il suo supporto di layout a griglia (una funzionalità fondamentale per Sider) e il supporto per l'annidamento di contenuti.

## 3.2 Struttura

Sider possiede 3 funzioni di *entry point*:

- AlignDocument(Debug).
- NestedElementsAlign(element,Debug).
- AlignElement(DivContainer,Debug).

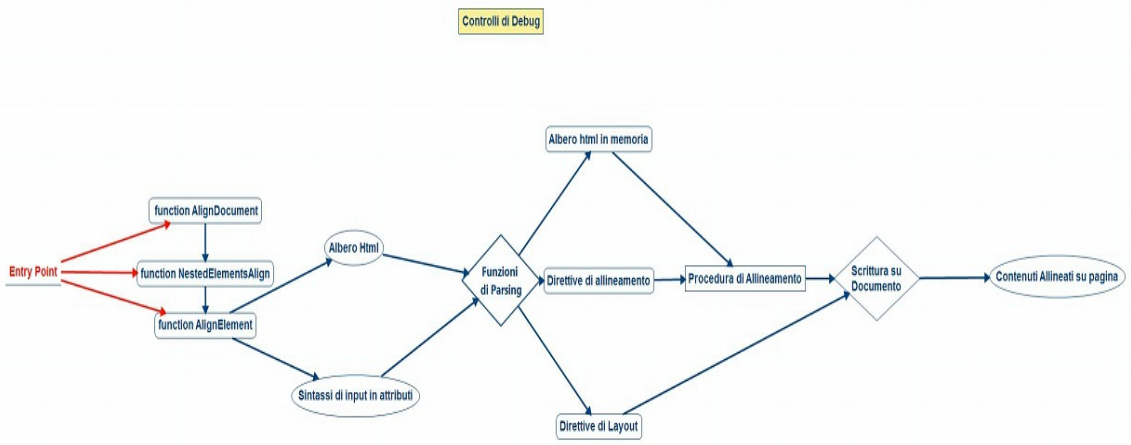



Figura 3.1: Struttura concettuale di Sider

La principale è AlignElement che è quella su cui si riversa il flusso di esecuzione delle altre 2. La funzione AlignElement prende in ingresso un elemento DOM che rappresenta l'albero contenente il materiale da elaborare e la stringa che specifica le modalità di debug e sostituisce sulla pagina in oggetto l'elemento passato da parametro con quello allineato. AlignElement è scomponibile in 4 parti principali che a loro volta fanno uso delle varie sotto funzioni del tool che verranno descritte dettagliatamente più avanti. La prima parte è quella dedicata alle funzionalità di debug che si occupa della creazione del frame di debug, i sottoframe dei contenitori e dei messaggi di debugging ad essi associati; questa parte non è da intendersi in sequenza perché le sue funzionalità per natura parallela del debugger sono sparse su un po' tutte le varie parti del tool anche se la creazione delle componenti chiave avviene nella parte iniziale di AlignElement.

La seconda si occupa del parsing delle direttive necessarie per l'allineamento dei contenuti e dei contenuti stessi dell'albero in questione passato da parametro.

La terza esegue l'allineamento vero e proprio in memoria .

E la quarta infine si occupa della scrittura su pagina dei contenuti restituiti in output dalla funzione precedente sostituendoli all'elemento d'ingresso.

La funzione d'ingresso NestedElementsAlign(element,Debug) controlla ricorsivamente all'interno dell'albero originato da element se ci sono contenitori marcati per l'elaborazione da parte di Sider tramite la funzione ausiliaria NestedCheck che restituisce un array contenente un puntatore a tutti gli elementi da elaborare ordinato per livello di profondità in cui si trovano nel sottoalbero associato a element; successivamente viene richiamata AlignElement per ogni elemento dell'array in questione fino a esaurimento dei puntatori, passando a ogni chiamata i parametri di debug a AlignElement.

La terza e ultima funzione di entrata è AlignDocument la quale chiama semplicemente la funzione NestedElementsAlign passandogli come parametro l'elemento "body" del documento, ciò equivale ad allineare tutti gli elementi marcati per l'elaborazione in esso contenuti.

Nel seguito si analizzeranno uno ad uno i 4 componenti (o moduli) costituenti Sider.

### 3.2.1 Sezione di Parsing

La componente di parsing della sintassi del tool è costituita da 3 funzioni principali:

- parseSyntax(SbSTextValue,AlignParameters)

E' la funzione che esegue il parsing delle direttive di allineamento, è chiamata direttamente dalla funzione AlignElement e prende in ingresso la stringa contenuta nell'attributo "sbs:containersSelectors" e restituisce dentro AlignParameters un array di record per ogni livello di allineamento specificato, ognuno contenente il numero associato alla procedura di allineamento da effettuare in quel livello (utilizzata in seguito dalla funzione di allineamento), il nome del tag, classe o identificatore su cui lavorare e un campo nullo in seguito utilizzato dalla procedura di allineamento. Si avvale anche della funzione checkToken che stabilisce le associazioni numeriche da dare alle direttive sotto forma di stringa.

- parseLayoutSyntax(SbsLayoutText)
- parseClassNames(SbsNames)

Queste due funzioni sono richiamate dal componente di output writeAlignedArrayonPage.

La funzione parseLayoutSyntax restituisce un array di record contenente le direttive di allineamento verticale e orizzontale interne alle celle del livello associato sotto forma di stringa, prese a partire dal contenuto dell'attributo "sbs:align".

Mentre parseClassNames restituisce semplicemente un array con le classi da associare a ogni livello di allineamento nell'albero finale di output a partire dal contenuto dell'attributo "sbs:markerClasses".

L'associazione dei record dei 3 array ai rispettivi livelli viene fatta implicitamente in base all'ordine di caricamento nella struttura dati.

C'è anche una quarta sezione che è quella che si occupa del parsing dell'albero dei contenuti su array, avente un array per ogni sotto albero(o colonna) da allineare, questa parte è integrata nel codice della funzione AlignElement.

### 3.2.2 Sezione di Allineamento

E' il cuore di Sider, la funzione dove avviene l'allineamento vero e proprio. Questo componente è costituto principalmente dalla funzione:

- recursiveAlign(PartialSubTree, currentlevel,sortinglevel,OutputTrees)

In breve questa funzione prende in ingresso rispettivamente l'array contenente i contenitori (colonne) di elementi html caricati in memoria precedentemente dal modulo di parsing, il livello di allineamento corrente su cui sta lavorando, l'array contenente i record delle direttive di allineamento precedentemente parsati e un puntatore vuoto che viene usato per la restituzione dell'albero di output allineato secondo specifica utente. In pratica l'allineamento viene realizzato tramite la parificazione degli elementi delle colonne secondo i loro livelli di appartenenza, concettualmente la procedura di allineamento si può dividere in 3 parti principali eseguite secondo l'ordine di citazione:

1. Il controllo e la gestione del caso base, ovvero dove la funzione non effettua più le ricorsioni e torna all'esecuzione delle funzioni precedenti sullo stack di chiamata. Il caso base consiste nell'allineamento totale degli elementi senza nessun parametro, qui gli elementi vengono ordinati solo in base alla loro sequenza di comparizione all'interno del sotto contenitore di appartenenza e si verifica quando il livello di allineamento corrente contiene la direttiva "*" oppure sono finiti i record delle direttive di allineamento ovvero quando è stata chiamata la funzione sull'ultimo record delle direttive di allineamento e si ha bisogno di una

parificazione dei contenuti non selezionati dall'utente per far quadrare l'allineamento finale.

2. La seconda sezione è dedicata al calcolo e all'elaborazione del sotto albero di ricorsione ovvero l'individuazione parallela sulle varie colonne degli elementi compresi tra quelli selezionati da direttiva utente(nel corrispondente record di allineamento) nel livello corrente, per poi passarli come parametro a un altra istanza di recursiveAlign che si occupa della gestione di quel livello specifico.

3. La parte finale si occupa della parificazione degli elementi selezionati tramite il record di allineamento corrispondente al livello in questione (che avviene dopo la chiamata ricorsiva) e l'avanzamento parallelo in maniera iterativa sul livello corrente negli array da allineare fino all'esaurimento degli elementi in essi contenuti.

RecursiveAlign fa uso di 2 funzioni ausiliarie scritte ad hoc:

- getFilter(FilterRecord,Colums)
  Traduce il record di allineamento delle direttive utente passato da parametro in una stringa di selezione Jquery e la ritorna in output.

- getnextAlignElement(primaryFilter,secondaryFilter,Column,sortingRecord)
  Prende le stringhe di selezione in Jquery primaryFilter e secondaryFilter, gli array contenenti le colonne da allineare e un record di allineamento (quello corrispondente al livello delle stringhe tradotte) e restituisce in output il primo elemento corrispondente ai criteri di selezione passati da parametro.

### 3.2.3 Sezione di Output

In questa sezione che è l'ultima in ordine di esecuzione, viene sostituito all'elemento "DivContainer" passato come parametro a AlignElement(DivContainer,Debug) l'albero html allineato di output generato da recursiveAlign.

Il modulo di output di Sider è implementato principalmente dalla funzione writeAlignedArrayonPage(Divs, context,DebugRecord) e in misura minore nella parte finale di AlignElement.

La funzione writeAlignedArrayonPage(Divs, context,DebugRecord) si occupa della traduzione del gruppo di array ("Divs") contenente rispettivamente gli elementi delle varie colonne, nell'albero html vero e proprio formattato secondo direttive utente.

La prima parte di questa funzione si occupa del parsing delle direttive di layout usando le funzioni della sezione di parsing sugli attributi dell'elemento "context" passato da parametro(che è l'elemento contenitore da allineare originale associato a quello allineato in memoria "Divs").

Successivamente il flusso di esecuzione ha una biforcazione in base al valore dell'attributo "margin-collapse" contenuto nell'elemento context; nella prima se questo assume valore "false" tutti gli elementi di ogni array contenuto in "Divs" vengono posizionati all'interno della stessa riga del contenitore finale ovvero ogni elemento con lo stesso indice all'interno di ogni array viene incapsulato all'interno di un elemento colonna di bootstrap ("col-xs-*") di uguale ampiezza e poi tutti quanti vengono a loro volta incapsulati in un elemento ("row") di bootstrap. Nel secondo flusso di esecuzione invece tutti gli elementi contigui appartenenti allo stesso livello di allineamento su ogni array (identificati tramite un attributo aggiunto da recursiveAlign contenente un numero rappresentante il livello a cui appartengono) vengono incapsulati nello stesso elemento colonna ("col-xs-*") uno per ogni array del contenitore "Divs" di ingresso; e immessi nella stessa riga (elemento "row"). Oltre a questo ogni elemento di ogni array viene incapsulato in un "div" a se stante aventi specifiche proprietà css che permettono a ogni elemento appartenente a una data cella della griglia di output di spandersi omogeneamente sull'altezza massima della cella stessa.

Infine in entrambe i flussi di esecuzione viene chiamata su ogni elemento allineato la funzione insideAlign(element,align) che prende in ingresso il loro "div" contenitore e gli aggiunge le proprietà css necessarie per rispecchia-

re l'allineamento interno alle celle della griglia voluto su quel livello che è contenuto dentro il parametro "align" .

La funzione ritorna un elemento contenitore principale, contenente gli elementi allineati con il markup di bootstrap che viene infine sostituito all'elemento contenitore originale nel documento dalla funzione AlignElement.

### 3.2.4 Sezione di Debug

La sezione di debug al contrario delle altre è delocalizzata su tutto il codice di Sider, comunque c'è un punto in particolare che svolge le funzionalità cardine di questa sezione che si trova nella parte iniziale della funzione AlignElement, dove viene stabilito in base al parametro delle funzioni di ingresso se le funzionalità di debug devono essere attivate o meno, la creazione del frame principale di debug, la creazione del sotto contenitore associato all'istanza, i controlli di base su sintassi e semantica delle direttive di ingresso e ridirezione delle eccezioni di runtime del javascript sui messaggi di debug.

Questa parte fa uso delle principali funzioni ausiliarie della sezione che comprendono le funzioni:

- createDebugFrame()
  Crea il frame principale di Debug.

- CreateDebugBox(ID)
  Crea la scatola contenente i messaggi di debug associata all'istanza di AlignElement.

- DebugChecks(ContainerContext,DebugRecord)
  Esegue i controlli di base sulla sintassi e sulla semantica delle direttive di ingresso.

- MistypeCheck(container)
  Controlla eventuali errori di sintassi sui nomi stessi degli attributi delle direttive e del namespace.

- validateDebugString(Dstring)
  Valida la stringa di debug passata come parametro nelle funzioni di ingresso.

## 3.3 Limiti

Durante lo sviluppo e il testing di Sider sono emersi alcuni principali limiti dell'attuale implementazione legate perlopiù ai limiti stessi delle tecnologie utilizzate per lo sviluppo:

- La parte in css usata per il posizionamento degli elementi allineati all'interno delle celle e la parte per l'espansione dei div contenenti gli elementi stessi nel layout in modalità collassato presenta problemi di compatibilità con il browser di *explorer* e *safari* poiché si ottengono formattazioni di layout indesiderate a causa di interpretazioni differenti del codice css da parte di questi ultimi, mentre su *chrome* e *firefox* il risultato è quello desiderato. Bisognerebbe aggiungere del codice ad hoc per ogni browser sulle parti che presentano tali problemi.

- Anche se l'approccio dichiarativo del tool riduce di molto la distanza tra i contenuti di base e gli stessi formattati in maniera comprensibile dai vari strumenti per l'allineamento, si ha comunque una parte di interazione umana non del tutto eliminabile per ottenere una formattazione compatibile con l'input del tool.

- Una parte del controllo dell'utente sul layout finale è sacrificata a vantaggio dell'automatizzazione di quest'ultimo. Un esempio per eccellenza è l'impossibilità di tracciare una linea continua tra una colonna di contenuti e un altra, a causa della struttura intrinseca del markup di output ed altre piccole limitazioni.

- Non vi è un supporto specifico per l'allineamento interno e la sincronizzazione di due o più flussi testuali tramite markup interno di questi

ultimi. In questo senso si era iniziata a sviluppare una funzionalità di questo tipo ma non ha dato risultati soddisfacenti e si è deciso di escluderla dalle funzionalità di Sider.

# Conclusioni

Con questo lavoro si è voluta andare a coprire la carenza di supporti specifici per il testo a fronte nel mondo della generazione automatica del layout. In particolare i vantaggi ottenuti tramite l'utilizzo di Sider rispetto agli approcci tradizionali basati su template o tramite formattazione manuale sono:

- Drastico snellimento del codice richiesto per ottenere un layout specializzato, del testo a fronte in questo caso.Il *trade-off* tra specializzazione/qualità del layout e automatizzazione viene minimizzato, dove le caratteristiche chiave caratterizzanti il layout possono ancora essere dichiarate direttamente dall'utente ma in maniera molto meno verbosa preservando tutti i maggiori vantaggi di una composizione di layout automatica.
- Riduzione della "distanza" tra il documento di input e quello di output, in opposto a quelli basati su template che richiedono di cambiare strutturalmente il documento; in maniera tale da cambiare le impostazioni di impaginazione solo dove richiesto.
- Sono richieste molte meno conoscenze pre-requisite al designer.
- Velocità di aggiornamento/personalizzazione molto ridotta.

## Sviluppi futuri

Come prima cosa sarebbe necessario risolvere i problemi di compatibilità descritti nel capitolo 3.3 tra le diverse tipologie di browser, aggiungendo del codice ad hoc nella sezione di output di Sider, per la generazione dei diversi tipi di markup html/css supportati(non è stato possibile generare tutto il markup di output con *Bootstrap* poiché correntemente non supporta l'allineamento verticale degli elementi nelle celle e la compressione degli elementi in uno stesso "div" contenitore; funzionalità che sono state implementate tramite markup classico html/css).

In seconda si potrebbero aggiungere funzionalità addizionali per l'allineamento di flussi testuali tramite ancore interne, simili a quelle proposte da XSL-FO 2.0[6]. Permettendo così all'utente di specificare direttamente i punti esatti di sincronizzazione del testo.

Si potrebbe introdurre inoltre una diversificazione delle modalità di input dei parametri di allineamento, di debugging e dell'albero dei contenuti da allineare, come per esempio permettere all'utente di specificarli tramite interfaccia grafica o un file di configurazione esterno oppure permettere il parsing dell'albero *DOM* dei contenuti da documenti salvati in remoto o in locale, con possibilità di specificare l'elemento radice su cui eseguire l'allineamento.

Un altra possibile miglioria potrebbe essere la riorganizzazione del codice in classi in modo tale da favorirne la riusabilità (Per la stesura di Sider si è preferito usare l'approccio per funzioni dato che il Javascript non presenta supporti espliciti per la gestione delle classi e l'utilizzo di queste avrebbe complicato ulteriormente lo sviluppo).

Infine Sider si potrebbe integrare come modulo aggiuntivo di un *framework* più a largo spettro, arricchendolo con funzionalità apposite per la gestione del testo a fronte, che attualmente nessuno presenta.

# Bibliografia


[1] W3C. CSS Paged Media Module Level 3. W3C Working Draft, 2013.

[2] W3C. CSS Print Profile. W3C Working Group Note, 2013.

[3] Barzelay. Document Mechanics: Basic Tools and Techniques for Variable Content Publishing, 2011.

[4] N. Hurst, W. Li, K. Marriott. Review of automatic document formatting. DocEng '09 Proceedings of the 9th ACM symposium on Document engineering, 2009.

[5] W3C. Extensible Stylesheet Language (XSL) Version 1.1. W3C Recommendation, 2006.

[6] W3C. Design Notes for Extensible Stylesheet Language (XSL) 2.0. W3C Working Draft, 2010.

[7] F. Giannetti. XSL-FO 2.0: automated publishing for graphic documents.DocEng '09 Proceedings of the 9th ACM symposium on Document engineering,2009.

[8] A. Di Iorio, L. Furini, F. Vitali, J. Lumley, T. Wiley. Higher-level layout through topological abstraction. DocEng '08 Proceedings of the eighth ACM symposium on Document engineering, 2008.

[9] M. Nebeling, F. Matulic, L. Streit, M.C. Norrie. Adaptive layout template for effective web content presentation in large-screen contexts.

DocEng '11 Proceedings of the 11th ACM symposium on Document engineering, 2011.

[10] Twitter, `http://getbootstrap.com/css/`, 2015.

[11] R. Piccoli, J. B. Oliveira. Balancing font sizes for flexibility in automated document layout. DocEng '13 Proceedings of the 2013 ACM symposium on Document engineering, 2013.

[12] J. B. Oliveira. Two algorithms for automatic page layout and possible applications. Multimedia Tools and Applications archive Volume 43 Issue 3, 2009.

[13] E. Schrier, M. Dontcheva, C. Jacobs, G. Wade, D. Salesin.Adaptive layout for dynamically aggregated documents. IUI '08 Proceedings of the 13th international conference on Intelligent user interfaces, 2008.

[14] Lumley J., Gimson R. and Rees O. Extensible Layout in Functional Documents. In Digital Publishing, Proc. of SPIEIS & T Electronic Imaging, Vol 6076. 2006.

[15] H. Brown, Desktop publishing, In Encyclopedia Encyclopedia of Computer Science 4th Pages 532-535, 2003.