Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

DocuDipity: un ambiente per esplorare tendenze nella scrittura di articoli scientifici

Relatore:
Dott.
ANGELO DI IORIO

Presentata da:
SIMONE BONFANTE

II sessione
2015-2016

Indice

Capitolo 1

Introduzione

In questa tesi verrà mostrato un sistema innovativo, finalizzato a supportare l'esplorazione e l'analisi di collezioni di articoli scientifici. Il nome stesso, DocuDipity, deriva dall'unione di "Document" e "Serendipity", cioè fare scoperte del tutto inattese, ma rilevanti, su documenti scientifici.
Seguendo i metodi tradizionali i lettori esaminano i documenti in molti modi diversi, a seconda dello scopo che si vuole raggiungere, e il tempo da spendere per questi documenti. In alcuni casi è sufficiente una lettura superficiale dell'articolo mentre in altri è necessaria una lettura più approfondita. Diversi lettori riesaminano un articolo più e più volte, concentrandosi su alcuni aspetti piuttosto che su altri, con diversi livelli di dettaglio, secondo la loro competenza e la consapevolezza.
Per esempio, mettiamo che un lettore voglia scoprire dove sono allocati in un documento i riferimenti bibliografici; ciò lo porta a dare una lettura approfondita a tutto l'articolo cercando ogni riferimento fino all'ultima pagina, senza mai averne una visione generale.
Grazie a DocuDipity è possibile individuarne i punti precisi in tutto il documento a vista d'occhio, senza l'obbligo di sfogliare pagina per

pagina, sezione per sezione, e prestare attenzione nel cercarli. Prendiamo un altro esempio più impegnativo. Immaginiamo che un lettore voglia analizzare un documento e capire se è stato elaborato da più autori diversi, e magari con stili di scrittura differenti. Oppure cercare in articoli multiautore le parti scritte da autori nativi britannici e non. Per uno studio del genere il lettore è costretto a leggere il documento per intero soffermandosi per studiare attentamente sezione per sezione.
DocuDipity fa lui stesso questa analisi distinguendo l'inglese americano dall'ortografia peculiare di quello britannico. Il lettore scopre a vista d'occhio quali sezioni sono state scritte da uno piuttosto che da un altro. Questo rende alternativa la visione e la lettura di un documento scientifico rispetto ai metodi tradizionali, mettendo in evidenza contenuti nascosti visibili solo grazie a uno studio approfondito dell'articolo stesso.
Questa visione viene data una una visualizzazione particolare, diversa da quelle esistenti: una visualizzazione in forma radiale chiamata "SunBurst"[1] progettata e disegnata per rappresentare la struttura gerarchica di un documento e mostrare i risultati delle analisi effettuate dal lettore.
Le analisi sul documento vengono fatte dal lettore stesso grazie ad un ambiente creato appositamente per la personalizzazione e l'editing di queste ricerche, chiamate "regole". Sorge spontaneo chiedersi come è possibile che studi e analisi su un documento, spesso complessi, possano trasformarsi dal linguaggio naturale a quello codificato senza perdere il significato. Ciò è possible con poche righe di codice JavaScript e CSS.
Vale la pena notare che i documenti presi in input da DocuDipity sono in formato XML[20], "un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo, ma non viene richiesta

alcuna informazione di base riguardo al formato con cui sono stati scritti".
Il sistema si basa sulla teoria dei pattern strutturali[8], il quale algoritmo "riconosce strutture testuali (paragrafi, sezioni, ecc) e fornisce meccanismi astratti; più in generale descrive documenti indipendenti dalla particolare semantica di specifici schemi di markup, strumenti e fogli di stile di presentazione." Permette di identificare il ruolo strutturale di ogni elemento in un insieme di articoli scientifici omogenei memorizzati come file XML. Infatti DocuDipity è in grado di estrarre informazioni relative alla struttura del set di documenti forniti come input, e usarlo sia per visualizzare i documenti stessi sia fornire un quadro di un'analisi per ispezionare il loro contenuto facilmente.
In DocuDipity la visualizzazione "SunBurst", accennata precedentemente, è strettamente legata ad una vista ipertestuale dell'articolo. Ciò consente ai lettori di avere una panoramica generale sia sulla struttura sia sulle informazioni del documento, avendo in primo piano l'oggetto di interesse primario: l'articolo stesso.
I documenti di DocuDipity e le "regole" accennate prima vengono raccolte e visualizzare in appositi ambienti. Il lettore può scorrere regole e documenti, visualizzarne in metadati come per esempio il titolo di una regola, la sua descrizione, l'autore ecc.., e scegliere una regola o un documento da applicare alla visualizzazione SunBurst per fare i tipi di analisi desiderati.
DocuDipity è un sistema che non nasce da zero, ma completa e arricchisce un sistema già esistente[7]. La staticità e i troppi limiti del sistema iniziale mi hanno portato a migliorarlo, tenendo sempre ben fisso il compito principale: fornire una visualizzazione alternativa dei documenti chia sia interattiva e dinamica. L'interattività e la dinamicità sono date dal coinvolgimento con l'utente. Sarà proprio lui l'esploratore e l'analizzatore dei documenti, e anche l'artefice delle sue regole. DocuDipity offre anche un pannello per la gestione di regole e documenti, dove l'utente può fare qualsiasi tipo di operazione con

le regole codificate da lui e da altri utenti come, ad esempio, copiarle, editarle, modificarle o eliminarle. In questo pannello di gestione l'utente ha anche la possibilità di caricare un set di documenti omogenei, cioè tutti dello stesso gruppo: più documenti sono dello stesso gruppo se hanno lo stesso vocabolario.

La tesi è strutturata come segue: nel capitolo 2 vengono citate e descritte alcune opere connesse con tecniche di visualizzazione simili. Nel terzo si descrive dettagliatamente il sistema, scelte progettuali e tool principali. Tutta la fase di implementazione è esposta nel capitolo 4. Il 5 tratta i punti deboli, punti di forza e infine le conclusioni.

Capitolo 2

Contesto scientifico

2.1 Ambiente: analisi della struttura gerarchica dei documenti

Uno dei punti chiave di DocuDipity è quello di rappresentare la struttura logica di un documento. Questa può essere ridotta al problema ben noto della visualizzazione delle gerarchie.
E' difficile voler rappresentare grandi quantità di dati in schermi sempre più piccoli, quindi un primo elemento da considerare è l'uso efficiente dello spazio.
Per questo motivo si preferiscono visualizzazioni gerarchiche implicite rispetto a visualizzazioni esplicite (tecniche che mostrano in modo esplicito le relazioni tra i nodi con archi e linee di collegamento) come per esempio in [14].
Questo articolo presenta una visualizzazione basata su calcoli matematici e confronti degli spazi utilizzati da varie rappresentazioni grafiche 2D strutturate ad albero. E' introdotto uno strumento che quantifica la distribuzione di tutti i nodi in una rappresentazione ad albero. Calcola la loro area media così da assegnare lo spazio destinato ad ogni

nodo.

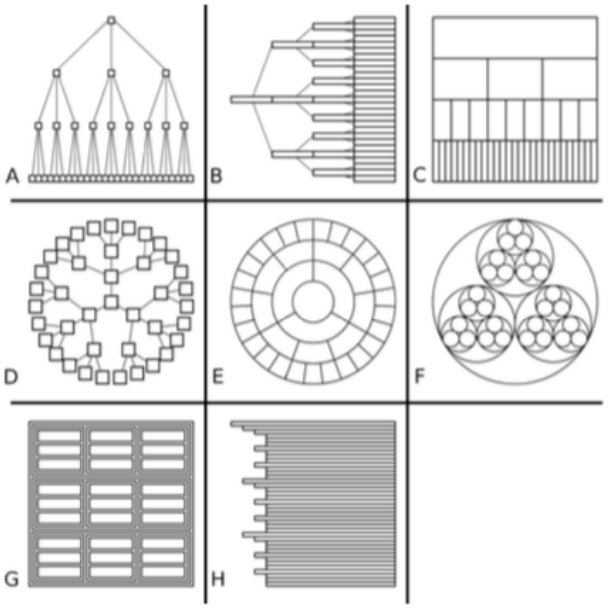


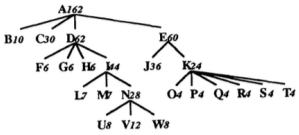
Figura 2.1: Diversi tipi fondamentali di rappresentazioni ad albero. A: classica (a strati). B: una variante A, nodi migliori sottoforma di etichette lunghe. C: ghiacciolo. D: radiale. E: cerchi concentrici. . F: cerchi annidati. G: treemap. H: contorno a frastaglia

Vediamo ora alcuni esempi e le tecniche di visualizzazione utilizzate.

2.2 Treemap

Treemap [18], una tecnica "slice-and-dice" che riempie lo spazio sulla base di una pianta rettangolare. Ogni nodo di un albero è rappre-

sentato da un rettangolo il quale contiene ulteriori rettangoli annidati fra loro rappresentando così un'estensione della struttura, usando al meglio lo spazio di visualizzazione disponibile.
Questo uso efficiente dello spazio permette la visualizzazione di gerarchie molto grandi e facilita la presentazione di informazioni semantiche.
In figura 2.2 e 2.3 vediamo come una struttura ad albero viene rappresentata attraverso Treemap.



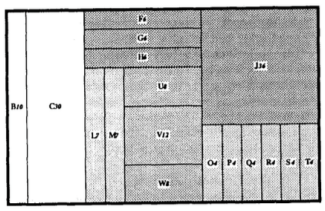
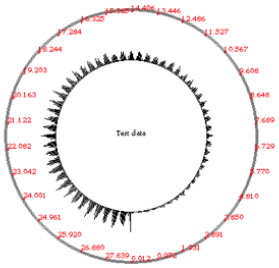


Figura 2.2: struttura ad albero. Figura 2.3: rappresentazione Treemap di dell'albero 2.1

Il colore e la superficie di ciascun elemento corrispondono attributi dell'elemento stesso. Possiamo, per esempio, codificare il tipo di elemento sotto esame o il numero di caratteri contenuti.
Sono state proposte diversi variazioni e miglioramenti delle tecniche di "treemap" iniziali. Vediamone alcune.

Triangular Aggregated Treemap

Triangular Aggregated Treemap[4] è un metodo per la gestione di grandi quantità di dati attraverso il metodo di aggregazione. Questo metodo è accompagnato una da una visualizzazione interattiva che migliorerà l'efficacia del processo di rappresentazione della struttura gerarchica. Utilizza diverse tecniche di visualizzazione come il *SolarPlot* (un istogramma circolare Fig 2.4) o un diagramma rettilineo (Fig 2.5).



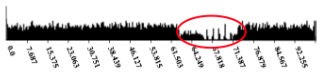


Figura 2.4: istogramma circolare. Figura 2.5: istogramma rettangolare con 2 liv. di aggregazione diversi

Quantum Treemap, Cascaded Treemap e Ordered Treemap

Quantum Treemap[3], è una variazione dei vari algoritmi Treemap, i quali garantiscono che ogni rettangolo generato avrà una larghezza e un'altezza multipli della dimensione dell'oggetto di input. Quantum Treemap sono progettati per la rappresentazione di immagini (come "PhotoMesa") o altri oggetti di dimensioni indivisibili ("Bubblemaps"). "Photomesa" è un browser di immagini che usa un algoritmo treemap per la rappresentazione di grandi quantità di immagini e metadati.
Cascaded Treemap [13] invece presenta una rappresentazione della gerarchia attraverso rettangoli a cascata e non tradizionali rettangoli annidati fra loro.
Ordered Treemap [19] è un'altra valida alternativa, la quale garantisce il mantenimento dell'ordine dei dati che cambiano continuamente.

La tecnica utilizzata in DocuDipity è un'altra alternativa a Treemap per la rappresentazione di strutture ad albero, *SunBurst*.
E' una rappresentazione in forma radiale, con al centro l'elemento radice, e la ramificazione sui raggi. SunBurst ha inoltre una rappresentazione più esplicita rispetto alla tecnica Treemap.

Sono state sviluppate più di 200 tipi di visualizzazioni diverse per la rappresentazione di strutture ad albero.
Ecco due esempi tra i più importanti. *Cone Tree*[16] rappresenta la struttura gerarchica in una visualizzazione 3D (fig 2.6), per ottenere il massimo dalle dimensione dello schermo, mostrando l'intera struttura anche attraverso animazioni interattive.

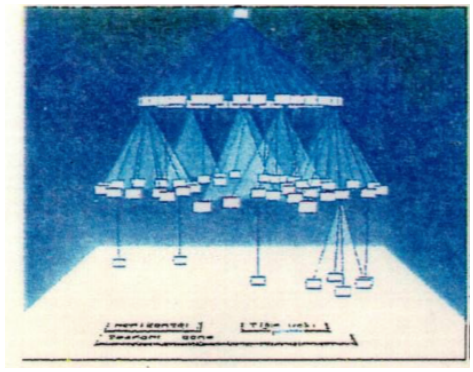


Figura 2.6: rappresentazione "cone tree" di una struttura gerarchica in 3D

Hyperbolic Tree[12] è una tecnica per la visualizzazione e la manipolazione di grandi gerarchie. Simile a *Cone Tree* ma in due dimensioni. Il nodo radice è al centro e gli altri estendo radialmente su un piano iperbolico (fig 2.7).

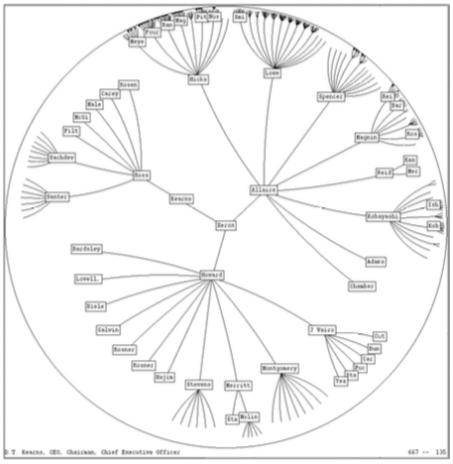


Figura 2.7: rappresentazione iperbolica di un organigramma.

2.3 Limiti

Abbiamo visto alcune delle tecniche principali per la rappresentazione della struttura gerarchica ad albero. Tra le più importanti è spiccata a tecnica "treemap" che, lo ricordiamo, mappa ogni nodo dell'albero su una figura rettangolare, annidando i nodi più interni.
Supponiamo di dover rappresentare una struttura complessa, con tanti nodi ramificati. La visualizzazione degrada, diventa complicato rappresentare troppi nodi e mantenere la chiarezza della struttura, perdendo, o quasi, i nodi più piccoli.

Abbiamo visto diverse variazioni e miglioramenti, come "treemap 3D", "Cone Tree" ecc. Con tecniche di visualizzazione 3D viene risolto il problema della rappresentazione della struttura, indipendentemente dal numero di nodi. Grazie ad animazioni interattive, l'albero di visualizzazione viene ruotato, tenendo sempre al centro l'elemento analizzato. Per far ciò però è necessario un costoso supporto di animazione in

3D, dal momento che l'albero ha le articolazioni che possono essere ruotate. E poi supponiamo di dover rappresentare una struttura articolato, con una grande quantità di nodi. Peccherebbe sulla scalabilità, in quanto troppo difficile da manipolare.

DocuDipity si presenta come un'alternativa a queste visualizzazioni. Rappresenta la struttura gerarchica in forma radiale, tenendo al centro l'elemento radice e le ramificazioni sui "raggi" del *SunBurst*. Può manipolare anche grandi quantità di dati, mantenendo la rappresentazione sempre chiara e ordinata.
E' un sistema che permette inoltre la customizzazione di regole grazie alle quali è possibile fare analisi approfondite sul documento.
Vedremo le sue funzionalità nei capitoli successivi.

Capitolo 3

Docudipity: un ambiente per esplorare tendenze nella scrittura di articoli scientifici

Nel capitolo precedente abbiamo analizzato alcuni sistemi che operano nel campo dello studio e dell'analisi della struttura logica dei documenti. Si sono evidenziati diversi limiti in ognuno di essi, ed è nata l'idea di creare un sistema innovativo rispetto a quelli già esistenti. In questo paragrafo spiegherò come DocuDipity sia stato pensato e progettato, analizzando ogni scelta progettuale.

3.1 Funzionalità principali

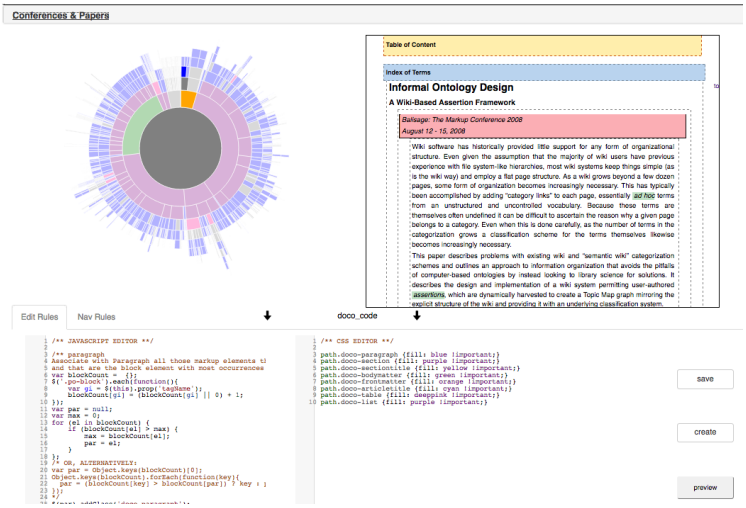


Figura 3.1: una panoramica dell'interfaccia di DocuDipity

3.1.1 Visualizzazione alternativa dei documenti

L'idea è stata quella di creare un sistema innovativo che offrisse una visione alternativa dei documenti; un sistema web-based, col compito di supportare l'elaborazione e l'analisi di articoli scientifici; un sistema capace di scoprirne ed evidenziarne caratteristiche senza leggerne il contenuto.

Il sistema è organizzato in due macro sezione: "home" e "management".
La prima è mostrata in fig. 3.1 e si compone di tre parti fondamentali: sul lato destro, il contenuto del documento viene visualizzato come un ipertesto; sul lato sinistro, una vista "zoomable" basata sulla tecnica SunBurst fornisce una panoramica dell'intera struttura del documento; nella parte inferiore, l'utente può scrivere JavaScript e CSS per

definire modelli personalizzati e valutandoli in modo interattivo.

Molti sistemi adottano visualizzazioni statiche, con il problema e il limite di non riuscire a rappresentare la struttura di documenti, sempre più complessa e nidificata, in spazi sempre uguali e ristretti come, ad esempio, lo schermo. Alcuni di questi rappresentavano elementi come blocchi e inlines con figure geometriche. Applicandogli documenti più pesanti la visualizzazione degrada in maniera esponenziale e diventa poco chiara perdendo di efficacia.

SunBurst

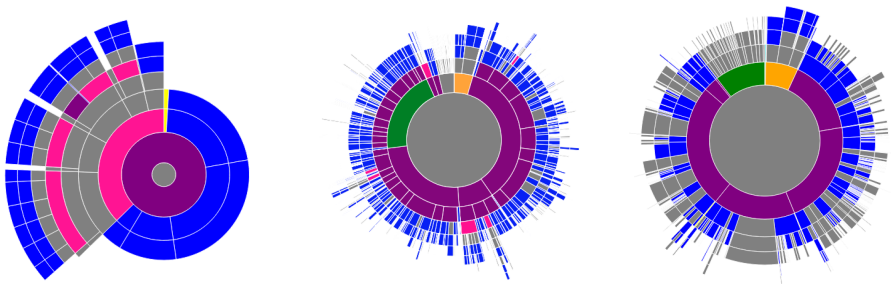


Figura 3.2: Tre rappresentazioni strutturali diverse di tre documenti diversi. Sezioni più grandi corrispondono ad paragrafi con un maggior numero di caratteri

Come si può notare dalla fig. 3.2, DocuDipity adotta una visualizzazione in forma radiale, chiamata visualizzazione SunBurst. In uno spazio limitato riesce a fornire una rappresentazione completa e intuitiva di documenti, anche di grandi dimensioni.
In SunBurst gli elementi della struttura gerarchica del documento sono disposti radialmente, con l'elemento radice al centro e gli altri che si allontanano man mano sempre di più. Gli elementi sono ordinati poiché vengono disegnati a partire da mezzanotte in senso orario.

Inoltre, la dimensione dell'elemento dipende dal numero di caratteri contenuto. Ciò comporta un graduale degrado man mano che ci si allontana dall'elemento radice. Questo problema viene risolto dall'interattività del sistema. Sono state aggiunte le funzioni di "zoom in" e "zoom out": quando l'utente fa clic su un elemento, SunBurst viene ridisegnato mostrando solo la sotto struttura dell'elemento selezionato e ponendo quest'ultimo come radice al centro (zoom in). Per salire di un gradino nella gerarchia l'utente può cliccare sulla nuova radice, cioè sull'elemento selezionato in partenza. (zoom out).
Abbiamo detto che il SunBurst oltre a mostrare la struttura di un documento permette anche di mostrare i risultati delle analisi fatte dagli autori, in poche parole applica le regole codificate dall'utente. Mostriamo qualche esempio accennato prima per rendere più chiaro il suo funzionamento.

Immaginiamo che il lettore voglia cercare tutte le referenze bibliografiche. Grazie a DocuDipity non sarà necessaria una ricerca approfondita sui documenti presi in esame, ma verranno mostrate tutte nella visualizzazione SunBurst (fig 3.3).

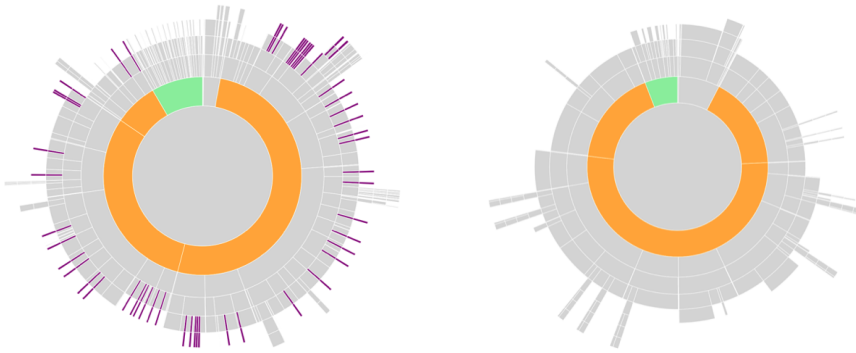


Figura 3.3: un articolo con riferimenti bibliografici, evidenziati in viola scuro, sparsi ovunque e un articolo dove non è stato menzionato

neanche un riferimento bibliografico

Immaginiamo altrimenti che il lettore voglia individuare le parti scritte da autori diversi, magari evidenziando quali sono state scritte da un autore nativo britannico, orgoglioso del suo inglese peculiare, e uno con uno stile americano acquisito da riviste, blog, film Hoollywoodiani e dalla stardardizzazione della lingua inglese nella letteratura scientifica. DocuDipity distingue linglese americano dall'ortografia britannica inglese, come ad esempio le terminazioni -ize -ise: si può immediatamente scoprire quali sezioni sono state scritte da un autore piuttosto che da un altro, come mostrato nella fig. 3.4.

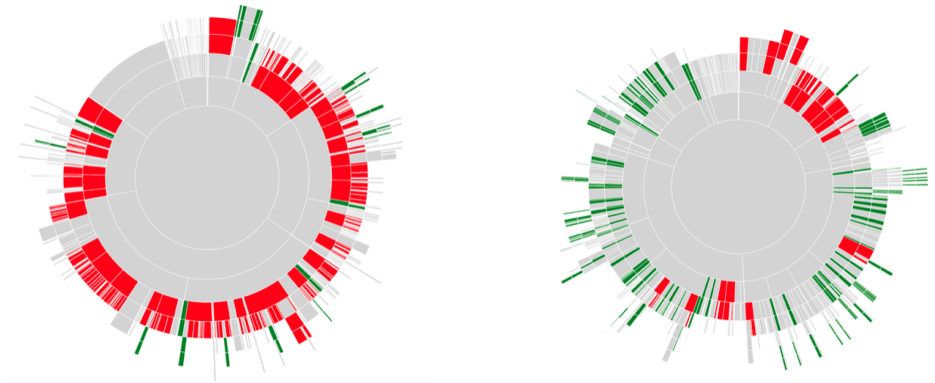


Figura 3.4: articolo con riferimenti bibliografici sparsi e un articolo dove non è stato menzionato neanche un riferimento bibliografico

Vista ipertestuale

Grazie all'algoritmo descritto in precedenza sullo schema strutturale dei pattern, DocuDipity identifica i modelli XML e sfrutta le informazioni prese da quest'ultimo per riprodurre una rappresentazione testuale simile al documento originale.

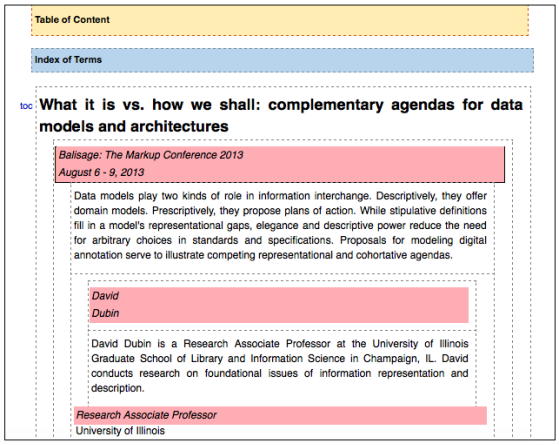


Figura 3.5: vista ipertestuale di un documento

DocuDipty converte i documenti XML in HTML composti da contenitori generici, blocchi e inlines associati a regole di codice CSS e JavaScript. L'organizzazione gerarchica di questi contenitori viene evidenziata attraverso bordi tratteggiati, in modo da rendere visibile al lettore la struttura del documento. Notare inoltre che all'utente non è mostrato nessun tag. Vengono solo messe in evidenza le strutture logiche del documento come contenitori, blocchi, inlines, frammenti di testo ecc..
Come si può notare dalla fig. 3.5 questa sezione è composta da due ulteriori ambienti per supportare e guidare l'utente alla lettura dell'articolo. Il prima è una tabella dei contenuti, dove viene raccolta in ordine gerarchico la struttura del documento. Il secondo è un indice di termini, raccolti in ordine alfabetico.
Come descritto prima il visualizzatore SunBurst, da all'utente una visione completa sulla struttura gerarchica del documento. Il sistema infatti permette di mantenere coordinati SunBurst e la vista ipertestuale. Passando il cursore del mouse su un frammento di testo nel visualizzatore ipertestuale (fig 3.1 elem. a destra) l'elemento corrispondente nel SunBurst e tutti i suoi antenati verranno evidenziati.

Questo permette di avere sempre una visione chiara sulla struttura del documento. Viceversa focalizzandosi su un elemento nel SunBurst il visualizzatore ipertestuale scrolla fino ad arrivare al frammento di testo corrispondente.

Editor JavaScript e CSS

Abbiamo visto, grazie agli esempi mostrati in precedenza, come DocuDipity, oltre ad analizzare la struttura dei documenti da la possibilità all'utente di scoprire caratteristiche nascoste evidenziandole. Fornisce quindi un mezzo per effettuare sofisticate analisi sul documento. Per fare ciò, gli utenti necessitano di un linguaggio per tradurre le ipotesi, spesso complesse, in condizioni, e un metodo per verificarne la validità. Invece di creare nuovi linguaggi e strumenti da zero, è stato deciso di utilizzare tecnologie web ben note come JavaScript e CSS.
Per questo abbiamo creato una sezione di editing composta di due aeree testuali; grazie alla libreria CodeMirror [5] l'utente può utilizzare JavaScript (in particolare JQuery3 [11]) in una per selezionare elementi e assegnar loro classi CSS. Nell'altra l'utente può definire regole CSS per specificare uno stile per le classi assegnate dal codice JavaScript.

Ecco un esempio di regola DocuDipity, il quale mostra gli elementi che ricorrono più volte nei documenti. Questo può essere convertito nel seguente codice JavaScript, che può essere valutato all'interno DocuDipity:

```
//DoCO-Paragraphs:
var blockCount = {};
$(".po-Block").each(function(){
  var gi = $(this).prop("tagName");
  blockCount[gi] = (blockCount[gi] || 0) + 1; });
var par = null;
var max = 0;
```

```
for (el in blockCount) {
  if (blockCount[el] > max) {
    max = blockCount[el];
    par = el;
  }
};
$(par).addClass("doco-Paragraph");
//DoCO-Section:
var toFilter = $(".po-HeadedContainer");
var sections = toFilter.filter(function() {
return (
  ($(this).children(".doco-Paragraph").length > 0)
    &&
  ($(this).parent('[class^="article"]').length != 0
    ) );
});
sections.each(function() {
  $(this).addClass("doco-Section");
});

//DoCO-FrontMatter
var toFilter = $(".po-Container , .po-HContainer , .po-
  Table, .po-Record");
var frontmatter;
var minSections = Number.MAX_VALUE;
toFilter.each(function() {
  if (
  (!$(this).is(docElement)) &&
  (!$(this).hasClass('doco-BodyMatter ')) &&
    ($(this).parent('.doco-Section ').length == 0 ) &&
    ($(this).parent('.doco-BodyMatter ').length == 0 )
      &&
```

```
    $(this).find('.doco-Section').length <
      minSections
  ) {
  frontmatter = $(this);
  minSections = $(this).find('.doco-Section').length;
}
});
frontmatter.addClass('doco-FrontMatter');
```

DocuDipity utilizza l'output dell'analizzatore pattern-based per assegnare una particolare classe CSS a ciascun elemento del documento. Ciascuna delle classi assegnate sta per uno dei modelli strutturali menzionati. In particolare, tutte queste classi utilizzano il prefisso "po-" seguito dal nome della classe. Ad esempio, il para elemento è stato assegnato alla classe "po-Block", la sezione elemento alla classe "po-HeadedContainer". I frammenti di codice introdotti sopra aggiungono nuove classi CSS a strutture Doco a partire dalle classi che identificano i modelli. Le regole di presentazione che dovrebbero essere usate per rappresentare queste nuove classi doco CSS, potrebbero essere create come segue:

```
.doco-FrontMatter { color: orange; }
.doco-Bibliography { color: black; }
.doco-Section { color: purple; }
```

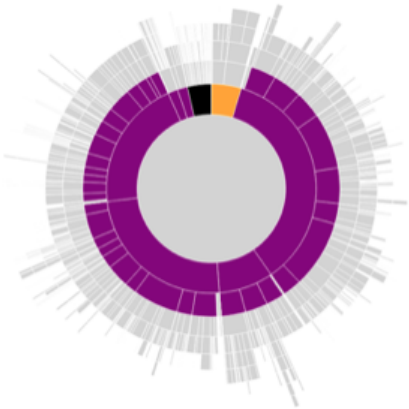


Figura 3.6: le sezioni sono in viola, il "front matter" davanti in arancione e la bibliografia in nero

Il Risultato è mostrato in fig. 3.6.
Questa mostra l'output dopo la valutazione del JavaScript e regole CSS. DoCO sections (in viola), front matter (arancione) e bibliography (in nero). L'organizzazione gerarchica della sezione è immediatamente evidente, così come la presenza del front matter e della bibliography e la loro posizione nel documento rispettivamente prima e dopo il corpo.

Vediamo ora un esempio già visto: come evidenziare le referenze bibliografiche in un documento.

```
/** BIBLIOGRAPHIC REFERENCES **/

var toFilter = $('xref');
var biblios = $('bibliography>bibliomixed');
toFilter.each(function() {
  var idRef = $(this).attr('linkend');
  var biblioFiltered = biblios.filter(function() {
```

```
        return $(this).attr('xml:id') == idRef;
    });
    console.log(biblioFiltered.length);
    if (biblioFiltered.length == 1)
      $(this).addClass('doco-xref');
});
toFilter = $('article>section');
toFilter.each(function() {
    $(this).addClass('doco-section-class');
});

toFilter = $('article>bibliography');
toFilter.each(function() {
    $(this).addClass('doco-bibliography-class');
});
```

Una volta selezionate le classi le assegnamo sempre agli elementi del documento. In particolare, si cerca tutti gli elementi ***xref*** filtrano fuori riferimento interno di tabelle e figure, riconosciuto dall'attributo **@linked**.
Dopo di che vengono customizzate le classi trovate con diversi colori per riconoscerli a vista d'occhio nel SunBurst:

```
path {fill: #f9f9f9 !important;}
path.doco-xref {fill: purple !important;}
path.doco-section-class {fill: yellow !important;}
path.doco-bibliography-class {fill: lightgreen !important;}
```

Il risultato è mostrato in Figura 3.3

Questo modulo è composto anche da una sezione di operazioni, subito di fianco alle aree testuali (fig 3.7). Per l'esattezza questi task sono *save*, *create* e *preview*. La funzione *save* opera in due maniere

differenti: se la regola presa in esame è dell'utente loggato allora salverà i progressi e le modifiche fatte nel codice. Se invece appartiene ad un altro autore la copierà, rinominandola, ed ereditando il codice JS e CSS. Con la funzione *create* l'utente crea una nuova regola ancora senza codice JavaScript e CSS, dandole un titolo, una descrizione e uno status che può essere privato o pubblico. Cliccando invece sul pulsante *preview*, sia JavaScript e CSS sono dinamicamente valutati in base al documento corrente, e il risultato di questo calcolo viene applicato direttamente sia alla vista Sunburst sia all'ipertesto. Questa facilità di utilizzo sostiene il processo di analisi, favorendo le indagini dell'utente avente a disposizione un mezzo per testare rapidamente le ipotesi e valutare la loro validità al volo.

Figura 3.7 :editor con bottoni save, create, preview

E' opportuno fare la distinzione delle regole complete e finite da quelle incomplete. Ad ogni regola viene attribuito uno "status" che può essere pubblico o privato, con la differenza che le regole pubbliche saranno visibili ed ereditatili da tutti gli utenti, mentre quelle private, in quanto tali, solo dall'utente creatore. L'utente stesso può cambiare lo status in qualsiasi momento da pubblico a privato e viceversa.

Il sistema ha 2 versioni diverse di navigazione: per utenti loggati, e semplici lettori non registrati. Per i primi è possibile usufruire di

tutte le operazioni elencate, mentre i secondi sono semplici spettatori dell'ambiente; possono scorrere documenti e regole, senza poterle creare copiare, modificare e salvare.

Alcuni degli obbiettivi per la nuova versione di DocuDipity erano:

- La creazione di regole (codificate sempre in JavaScript e CSS) da parte dell'utente, con i relativi metadati.
- La gestione di regole con operazioni di salvataggio, copia, modifica, eliminazione, caricamento.
- Il caricamento di un set di documenti omogenei.

Queste operazioni comportavano una creazione dinamica di dati da gestire e mantenere. La creazione di regole appartenenti ad un utente necessita di un modulo per la registrazione e il riconoscimento dell'utente stesso. Era necessario mantenere le credenziali di login in un database. Così facendo un utente registrato e riconosciuto ha la possibilità di creare il suo set di regole e di poterle salvare nel DB con i relativi metadati come l'autore, una descrizione, ecc.. Il sistema dalla prima alla seconda versione acquisisce dinamicità e modularità, offrendo così ad ogni utente la possibilità di poter navigare e svolgere le attività offerte in totale autonomia.

Inoltre la prima versione di DocuDipity si presentava con parti sconnesse graficamente, poca gestione degli spazi e una versione ottimizzata solo su schermi Full HD.
Uno degli altri obbiettivi era renderlo più navigabile e intuitivo possibile tenendo sempre come elementi principali il SunBurst e l'ipertesto al centro.
Per ridurre gli spazi ho creato sezioni collassabili (quella di visualizzazione dei documenti fig 3.9), ho ridimensionato la vista ipertestuale e la

visualizzazione SunBurst e ho inoltre ricostruito il layout e rendendolo responsive su qualsiasi tipo di schermo fino a una certa dimensione prefissata grazie al framework Bootstrap [9] e a nuove regole CSS.

Metadati di regole e documenti

Abbiamo visto finora un ambiente di editing e customizzazione delle regole codificate in JavaScript e CSS. Uno dei punti prefissati nella progettazione della nuova versione era un ambiente nuovo per la raccolta di regole e documenti, con i relativi metadati in modo che ogni autore potesse navigare in totale autonomia avendo il proprio set di regole e caricando i propri documenti.

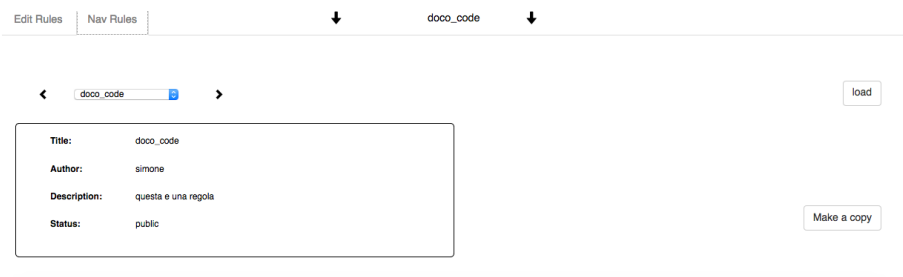


Figura 3.8: sezione di metadati delle regole

Si può notare in fig. 3.8 una sezione organizzata in 2 ambienti interscambiabili attraverso due tab. "Edit Rules" e "Nav Rules". Questo ambiente dedicato ai metadati delle regole lo troviamo nel tab "Nav Rules". Qui l'utente potrà scorrere tutte le regole presenti nel sistema e raccogliere informazioni quali il titolo, l'autore, la descrizione. Ogni regola ha inoltre uno status, che può essere pubblico o privato. L'autore di una regola può decidere in qualsiasi momento di cambiare lo status di una regola, ma questo verrà mostrato dopo nel pannello di

gestione. Quelle private non saranno visualizzate se non dal suo creatore. Una volta scelta una regola questa può essere caricata o copiata. Al caricamento il sistema applica al SunBurst la regola descritta, valutando il codice CSS e JS, e lo ridisegna, sempre secondo la struttura logica del documento preso in analisi insieme alla regola appena caricata. Inoltre viene fatto il passaggio in automatico alla sezione di editing con il codice JS e CSS relativo alla regola considerata, quindi passa in automatico da "Nav Rules" a "Edit Rules".
L'utente può creare una copia della regola esaminata, ereditando il codice CSS e JS, e rinominandola a suo piacimento grazie a *make a copy*. Vale la pena sottolineare che l'operazione di copia viene eseguita solo se l'utente è riconosciuto. In caso contrario il sistema chiederà a quest'ultimo di effettuare il login aprendo l'apposita finestra.

Come per le regole, è stato creato un ambiente anche per la raccolta dei documenti.

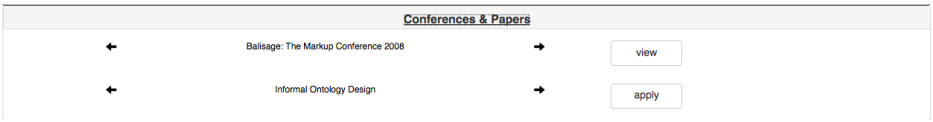


Figura 3.9: sezione di metadati dei documenti

"Conferences and Title" è una sezione collassabile la quale raccoglie tutti i documenti caricati nel sistema (fig 3.9). E' divisa in due sotto sezioni. La prima dove vengono suddivisi tutti i documenti per gruppo, cioè secondo il vocabolario con cui sono stati scritti.
Facendo clic su *view* DocuDipity carica nella seconda tutti gli articoli relativi al gruppo selezionato. Facendo clic su *apply* invece DocuDipity carica il documento scelto ridisegnando nuovamente il SunBurst e cambiando il contenuto ipertestuale secondo la struttura del documento applicato.

Gestione di regole e documenti

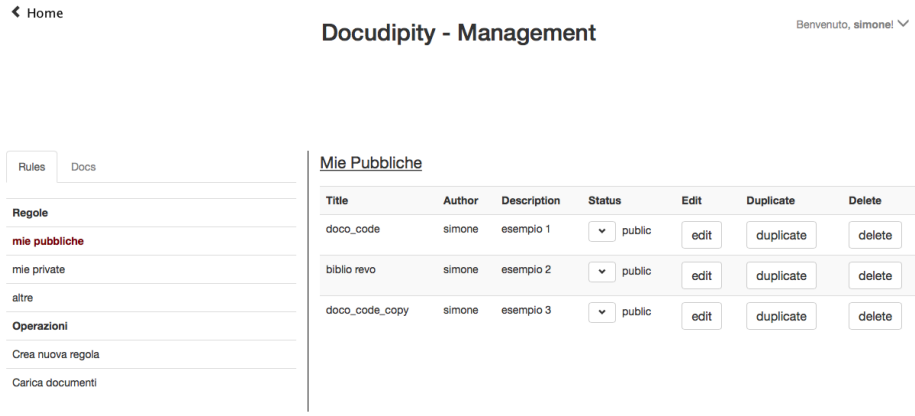


Figura 3.10: pagina di gestione di regole e documenti. Sono selezione le regole pubbliche dell'utente loggato

E' una sezione dedicata alla gestione di regole e documenti. Date le tante funzionalità ho deciso di organizzarla e progettarla in un'altra pagina.
Come nella pagina principale c'è sempre un modulo per la registrazione e il login, senza il quale non è possibile fare alcuna operazione. Sulla sinistra due tab racchiudono le due categorie prese in gestione: regole e documenti fig 3.10.

Regole

Andando su Rules appariranno diverse sezioni. Concentriamoci su "mie pubbliche", "mie private" ed "altre". Cliccando su mie pubbliche al centro apparirà una tabella dove vi saranno tutte le regole pubbliche dell'utente loggato. Saranno visibili tutti i metadati di ogni regola e disponibili diverse operazioni:

- Cambiare status della regola da pubblica a privata.

- Editare la regola, quindi applicarla al SunBurst e alle aree di testo nella sezione di editing nella pagina principale.
- Duplicare la regola, quindi farne una copia, ereditandone il codice.
- Eliminare la regola.

A "mie private" è attribuita una tabella medesima con le stesse operazioni, con la differenza che vengono mostrate le regole private dell'utente, e che lo status viene cambiato da privato a pubblico. Nella sezione "altre" le uniche feature disponibili sono quelle di editing e di duplicating. Dato che le regole illustrate non appartengono all'utente lettore, non gli sarà possibile farle il cambio di status o eliminarle. Chiaramente saranno visibili solo le regole pubbliche, e non quelle private.

Oltre all'elenco delle regole disponibili è possibile anche creare una nuova regola in questa pagina, per poi editarla nella pagina principale.
Oltre all'elenco delle regole disponibili è possibile anche creare una nuova regola in questa pagina, per poi editarla nella pagina principale.

Documenti
Ogni documento caricato nel sistema appartiene a un gruppo, cioè è stato scritto secondo un certo vocabolario. DocuDipity raggruppa i documenti per gruppo.
Nella sezione dei documenti DocuDipity fornisce al lettore tutti i gruppi dei documenti caricati, e per ogni gruppo una tabella con tutti gli articoli aderenti. E' possibile caricare ogni articolo della tabella nella pagina principale e applicarlo al SunBurst e all'ipertesto, valutando

così su di esso le regole.

Figura 3.11 :pagina di gestione di regole e documenti. Sezione di caricamento dei documenti

Oltre all'elenco dei gruppi e dei vari articoli è possibile anche caricare un set di documenti seguendo determinate regole illustrate nella sezione di caricamento fig(3.11): il file da caricare dovrà essere in formato ZIP e di una dimensione massima di 10MB. Il file dovrà contenere solo documenti XML i quali verranno mandati in pasto all'algoritmo, citato in precedenza, e poi caricati nel sistema, pronti per essere letti dal visualizzatore ipertestuale e dal SunBurst. E' importante che tutti i documenti caricati siano appartenenti allo stesso gruppo. Cioè siano stati scritti con lo stesso vocabolario. Grazie ad un algoritmo creato, controllo per ogni documento la radice andando ad estrapolare il namespace e la versione dell'articolo:

```
<article xmlns="HTTP://docbook.org/ns/docbook"
version="5.0-subset Balisage-1.2">
```

In realtà non è così semplice questo tipo di analisi, ma non essendo questa la funzionalità principale non mi sono focalizzato su questo aspetto. Sarà sicuramente un punto da prendere in considerazione per futuri miglioramenti di DocuDipity.

Se il file caricato non rispettasse queste caratteristiche oppure fosse già stato caricato da un altro utente non verrà accettato.

API RESTful

Docudipity è stato creato come un Web Service RESTful, per il collegamento di risorse e la comunicazione senza stato adottando lo schema URI. Il sistema identifica le risorse mappate nell'URI, cioè documento da caricare e regola da applicare

```
index_logged .php?doc=doctitle&rule=ruletitle;
```

dove doctitle è il documento da caricare e ruletitle la regola da carica. Una volta presi questi valori il sistema si incarica di caricare nel SunBurst e nell'ipertesto il documento selezionato e la regola, con il relativo codice CSS e JS nelle aree di testo nella sezione "Edit Rules". Vengono aggiornati anche i metadati relativi al documento e alla regola. Per esempio, immaginiamo che l'URI sia

```
index_logged.php?doc=Informal Ontology Design&–
rule=biblioref_code
```

e che quindi il lettore voglia caricare l'articolo *Informal Ontology Design*e la regola *biblioref-code* (la quale individua nel documento le referenze bibliografiche e le evidenzia colorandole di viola). Il sistema si occuperà di caricare il documento e di applicargli la regola mostrando così la visualizzazione desiderata (fig 3.12)

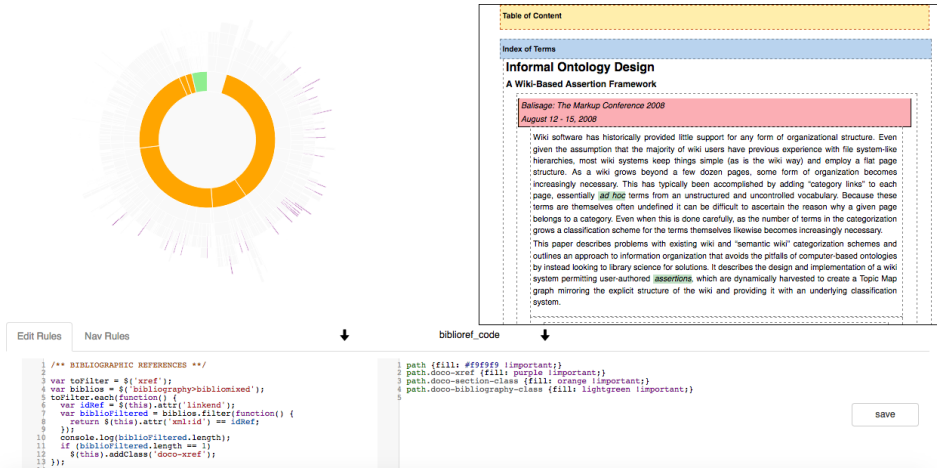


Figura 3.12: nel SunBurst e nell'ipertesto è stato caricato il documento selezionato, e nell'editor il codice della regola

3.2 Wireframe: "Mockup Balsamiq"

Un Wireframe è un grafico, molto semplificato, che serve per rappresentare gli spazi e le funzionalità che una pagina web dovrà avere. Per progetti, anche di piccole dimensioni, spesso è utile realizzare un wireframe durante la progettazione, perché ci aiuta a:

- chiarirci le idee sui contenuti della pagina.
- verificare che abbiamo fatto posto a tutto ciò che serve.
- comunicare il progetto ad altri (grafici e front-end developer)
- fare tentativi per ottimizzare il posizionamento dei contenuti

In principio i web designer realizzavano i wireframe con carta e matita, cosa che possiamo pensare di continuare a fare, ma avere a disposizione

uno strumento visuale, come *Balsamiq Mockups*[2], significa avere una serie di vantaggi non indifferenti, i due più importanti sono:

- Riutilizzo di porzioni di pagina già disegnati (con carta e matita sarebbe sempre un fastidioso collage), questo abilita a lavorare per "design pattern", modelli già pronti da adattare al contesto.

- Modifiche e adattamenti veloci, che ci permettono di comparare due soluzioni rapidamente e spostare elementi in modo efficiente.

Nella fase di progettazione ho dovuto fare un lavoro di mockup scegliendo proprio *Balsmiq* come software. Sono riuscito a ordinare le idee su un piano di lavoro simulato, e ad organizzare una prima versione "cartacea" dell'interfaccia di DocuDity.
Sono state create più versioni di DocuDipity durate la fase di progettazione, dove sono state apportate di volta in volta diverse modifiche. In fig 3.13 troviamo la prima versione progettata di DocuDipity.

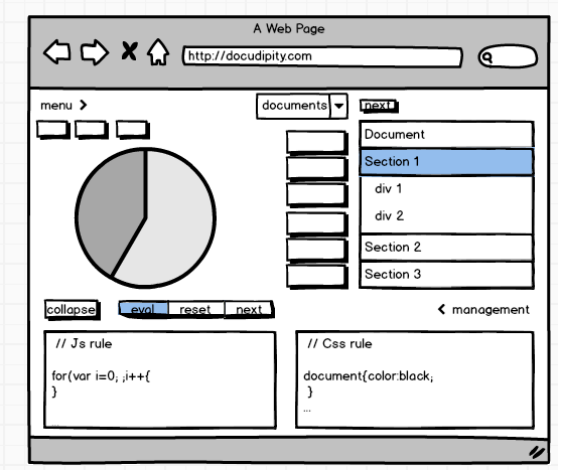


Figura 3.13: prima versione di DoduDipity progettata con Balsamiq

Le parti centrali erano SunBurst, Ipertesto ed editor di testo per il coding delle regole. Non c'erano le sezioni dedicate ai metadati e allo scorrimento di regole e documenti e non era in progetto un modulo per la registrazione degli utenti.

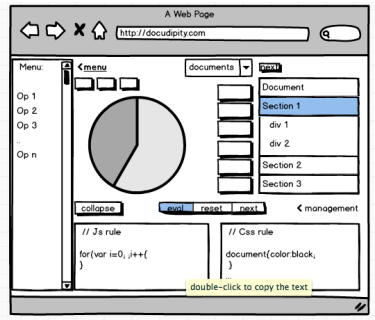


Figura 3.14: seconda versione di DocuDipity progettata con Balsamiq

Nella versioni successive (fig 3.14) avevo progettato un menu collassabile a sinistra il quale conteneva tutte le operazioni ora descritte nella pagine di gestione di regole e documenti, fino ad arrivare ad una versione simile all'originale, compresa di sezioni collassabili, operazioni su regole e documenti, modulo per la registrazione e pannello di gestione fig (3.15).

Figura 3.15: versione ultimata di DocuDipity

Capitolo 4

Dettagli sull'implementazione di DocuDipity

Abbiamo visto nel capitolo precedente come è strutturato DocuDipity. Vediamo ora i punti salienti dell'implementazione

4.1 Le tecnologie

DocuDipity è sviluppato in diversi linguaggi :

- HTML per la struttura della pagina
- CSS per lo stile e il layout
- JavaScript per tutto ciò che riguarda il front end
- PHP per le funzionalità di back end.

Sostanzialmente le parti principali sono quelle sviluppate in JS e in php. la prima parte consiste in una serie di operazioni asincrone come richieste HTTP e chiamate $ajax$/post per la comunicazione con script php lato server. Inoltre sfrutto due librerie JavaScript :

- d3 (data-driven documents) [6] per lo sviluppo del SunBurst.
- CodeMirror per lo sviluppo degli editor JS e CSS.

e un framwork per lo sviluppo del layout della pagina.

4.2 Librerie

Per lo sviluppo di DocuDipity mi sono fornito di due librerie: d3 (data-driven documents) per la realizzazione del SunBusrt e CodeMirror, per creare un ambiente di editing riservato al JavaScript e al CSS.

4.2.1 d3: data-driven documents

D3 (data-driven Documenti o d3.JS) è una libreria JavaScript per la visualizzazione dei dati utilizzando gli standard web. D3 aiuta a visualizzare i dati utilizzando SVG, Canvas e HTML. D3 combina potenti tecniche di visualizzazione e interazione con un approccio data-driven per manipolazione del DOM, offrendo tutte le funzionalità del browser moderni e la libertà di progettare interfacce visive interattive.

E' possibile utilizzare D3 per generare una tabella HTML da una serie di numeri. In alternativa, utilizzare gli stessi dati per creare un grafico a barre in formato SVG interattiva con transizioni morbide e l'interazione.

Come descritto in [6] "D3 non è un quadro monolitico che cerca di

fornire ogni caratteristica immaginabile. Invece, D3 risolve il nocciolo del problema: efficiente manipolazione di documenti in base ai dati. Questo evita la rappresentazione di proprietà e offre una straordinaria flessibilità, esponendo tutte le funzionalità di standard web come HTML, SVG e CSS. Con un overhead minimo, D3 è estremamente veloce, sostenendo grandi set di dati e comportamenti dinamici di interazione e animazione. Lo stile funzionale di D3 permette il riutilizzo del codice attraverso una variegata collezione di componenti e plugin."

D3 ci permette di ottenere la visualizzazione SunBurst, simile a una visualizzazione treemap, ma con la differenza che utilizza un layout radiale. Il nodo radice dell'albero è al centro, con le foglie sulla circonferenza. Abbiamo usato d3.JS per rappresentare la struttura gerarchica dei documenti attraverso il SunBurst.
Vediamo ora pezzi di codice, del modulo *draw.JS*, grazie ai quali è stato creato il SunBurst.
Una volta selezionato un documento richiamo la funzione *updateDocument* passandogli come argomento il documento selezionato. *updateDocument*, oltre a caricare nella vista di ipertesto il nuovo documento, richiama la funzione *drawVisualisation* che disegnerà il SunBurst applicandogli il documento passatogli come parametro.

```
var arc = d3.svg.arc()
    .startAngle(function(d) { return Math.max(0, Math.min(2 * Math.PI, x(d.x))); })
    .endAngle(function(d) { return Math.max(0, Math.min(2 * Math.PI, x(d.x + d.dx))); })
    .innerRadius(function(d) { return Math.max(0, y(d.y)); })
    .outerRadius(function(d) { return Math.max(0, y(d.y + d.dy)); });

var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height)
  .append("g")
    .attr("transform", "translate(" + width / 2 + "," + (height / 2) + ")");
```

Come illustrato prima il SunBurst dispone di una funzione di zoom in e di una funzione di zoom out. Per zoomare in avanti basta cliccare su uno dei raggi del subrust, e questo verrà messo come elemento radice, mostrando solo la sotto struttura dell'elemento selezionato. se si clicca al centro zooma all'indietro di un gradino.

```
function click(d) {
  svg.transition()
      .duration(750)
      .tween("scale", function() {
        var xd = d3.interpolate(x.domain(), [d.x, d.x + d.dx]),
            yd = d3.interpolate(y.domain(), [d.y, 1]),
            yr = d3.interpolate(y.range(), [d.y ? 20 : 0, radius]);
        return function(t) { x.domain(xd(t)); y.domain(yd(t)).range(yr(t)); };
      })
    .selectAll("path")
      .attrTween("d", function(d) { return function() { return arc(d); }; });
}
```

4.2.2 CodeMirror

CodeMirror è un editor di testo versatile implementato in JavaScript per il browser. E' specializzato per la modifica del codice, e viene fornito con una serie di modalità di linguaggi che implementano funzionalità di editing più avanzate.

Una ricca API di programmazione e di un sistema di tematizzazio-

ne CSS sono disponibili per la personalizzazione di CodeMirror e per estendere l'applicazione con nuove funzionalità.

CodeMirror è un componente Editor che può essere integrato nelle pagine web. La libreria di base prevede solo la componente editor, nessun pulsante di accompagnamento o altre funzionalità IDE. Essa fornisce una ricca API in cima alla quale tale funzionalità può essere implementata semplicemente.

CodeMirror opera con modalità specifiche del linguaggio.Queste modalità sono programmi JavaScript che aiutano a colorare (e a volte tratteggiare) il testo scritto in un determinato linguaggio. La distribuzione viene fornita con un numero di modalità (vedi la directory / modalità), e non è difficile scriverne di nuovi per altri linguaggi.

In DocuDipity CodeMirror crea due aree di testo, una per il coding di JavaScript e l'altra per il coding di CSS.

```
var js_textarea = document.getElementById("js_editor");
var js_editor = CodeMirror.fromTextArea(js_textarea, {
    mode: "javascript",
    lineNumbers: true,
    theme: "default",
    readOnly: false
});
js_editor.setSize(width, height);
js_editor.setValue(js_code);
```

Figura 4.1: funzioni di libreria CodeMirror per inizializzare l'area di testo per il coding di JavaScript

1. Mode indica il linguaggio che si vuole utilizzare nell'area di testo.

2. Numeri delle linee come guida.

3. Tema quello di default e con "readOnly: false" specifico che non è un editor di sola lettura, ma ovviamente anche di scrittura.

4. Infine *.setSize()* setta l'altezza e la grandezza dell'editor mentre *.setValue()* setta il valore da caricare nell'area di testo. In poche parole il codice JavaScript da inserire nell'editor creato.

E' esattamente uguale per l'editor CSS.
Le variabili sono *css-textarea* e *css-editor* e cambia naturalmente la mode, che viete settata con "CSS".
Ricordo che gli editor servono per creare regole codificandole in JavaScript e CSS.

4.3 Implementazione

L'implementazione di DocuDipity si divide in client side e server side. Nel lato client utilizzo CSS (e Bootstrap) per l'elaborazione del layout e tutto ciò che riguarda JavaScript (jQuery) per la realizzazione dei vari script, incluse chiamate Ajax e richieste HTTP.

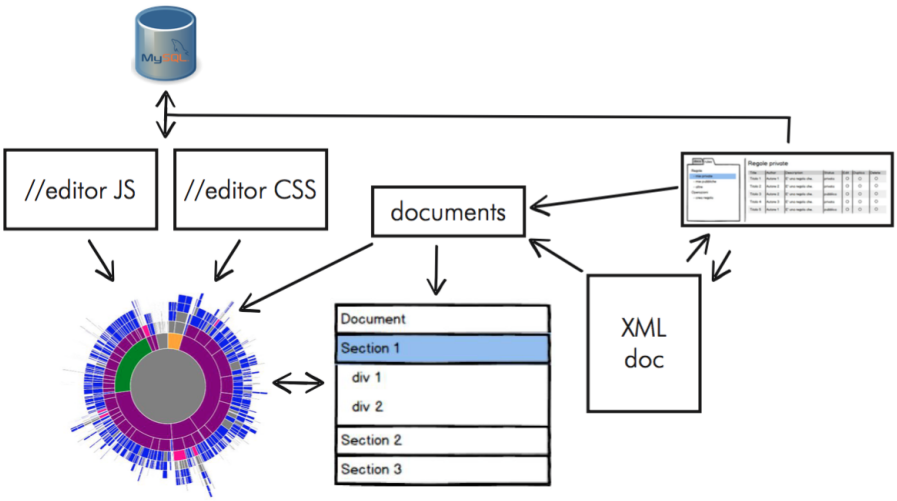


Figura 4.2: schema strutturale di DocuDipity

4.3.1 Client side: JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione web lato client per la creazione, in siti web e applicazioni web, di 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, caricamento della pagina ecc...).

Ho sviluppato diverse funzioni per l'elaborazione e la manipolazione di documenti e regole usufruendo anche di richieste HTTP e chiamate asincrone.
Vedremo ora nel dettaglio come.

Documenti e richieste HTTP

I documenti analizzati da DocuDipity vengono caricati nel sistema attraverso un tool specifico, che illustrerò nel prossimo paragrafo. In questo mostro come vengono acquisiti i documenti, ragguppati, e applicati al SunBurst e alla vista ipertestuale.

Uno script php, che verrà mostrato anch'esso nel paragrafo successivo, scansiona una lista dei file presenti nella directory xml, passati poi al client. Questa lista, sotto forma di array, corrisponde alla lista dei documenti caricati nel sistema.
Per ogni elemento dell'array viene eseguita una richiesta HTTP al server per ottenerne il contenuto.

```
function load_docs(){
    for(var i=2; i<docs.length; i++){
        getXml(docs[i],i);
    }
}
function getXml(data, i){
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (xhttp.readyState == 4 && xhttp.status == 200) {
            getTilte(xhttp, data);
        }
    };
    xhttp.open("GET", "data/xml/"+data+"", true);
    xhttp.send();

    function getTilte(xml, data) {
        data=("data/json/"+data);
        data=data.replace('.xml', '.json');
        var xmlDoc = xml.responseXML;
        var x = xmlDoc.getElementsByTagName('title')[0];
        var h = xmlDoc.getElementsByTagName('conftitle')[0];
        var t = h.childNodes[0];
        var y = x.childNodes[0];
        loadtitle(y.nodeValue, t.nodeValue, i, data);
    }
}
```

Figura 4.3: richiesta HTTP: cerco in ogni docuemtno XML il tag "title" e "conftitle"

La funzione *load-docs()* scorre tutta la lista di documenti e per ognu-

no di essi viene richiamata la funzione *getXml()* la quale genera una richiesta HTTP facendosi restituire il contenuto del documento passatogli come argomento; richiama poi la funzione *getTitle()* la quale valuta la risposta, analizza il documento xml ottenuto e ne cerca titolo e gruppo.

Viene fatta una raccolta di tutti i gruppi dei documenti presenti, e ad ogni gruppo viene associato un set di articoli che ne fanno parte con i rispettivi titoli.

Essendo un sistema RESTful il primo controllo che si fa è che il documento specificato nell'URI esista.

index_logged.php?doc=doctitle&rule=ruletitle&pass=0;

Questo significa che al caricamento della pagina viene fatto un controllo. Sempre grazie a uno script php viene preso il valore nella variabile doc passandolo al JavaScript il quale cerca il documento selezionato. Una volta trovato, richiama la funzione update*Text()* e *drawVisualization()*, funzioni predefinite per ridisegnare il SunBurst e ristrutturare la vista ipertestuale con il documento corrente.
La lista dei gruppi e i relativi documenti vengono appese nella sezione collassabile "Conferences Papers". Cambiando gruppo ricarico i documenti appartenenti a quest'ultimo nella sezione sottostante; applicando un documento invece vengono richiamate sempre le funzioni *drawvisualisation()* e *updateText()* per il SunBurst e l'ipertesto, e inoltre aggiorno l'URI con il nuovo documento caricato.

```
var newHREF="index_logged.php?doc="+all_doc[0][i]+"&rule="+vettore_regole[idx]["title"]+"&pass=0";
history.pushState('', 'New Page Title', newHREF);
```

Figura 4.4: cambio dinamico dell'URI

Regole e chiamate asincrone

Le regole, contrariamente ai documenti, vengono salvate nel database; un script php esegue una query sql permettendogli di ottenere tutte le regole presenti, con i relativi metadati. Queste vengono salvate in un vettore e passate al lato client.
Un array associativo contiene tutti i dati riguardanti le regole: il codice CSS e JavaScript, il titolo l'autore e gli altri metadati, e lo status.

```
<script type="text/JavaScript">var vettore_regole =
<?php echo JSon_encode($vettore_regole); ?></script>
```

Nella sezione "Nav Rules" viene creata una raccolta delle regole filtrando quelle pubbliche di ogni utente e pubbliche e private dell'utente loggato.

```
for(var i=0; i<vettore_regole.length; i++){
    if ((vettore_regole[i]["status"]==1)||((vettore_regole[i]["status"]==0)&&(vettore_regole[i]["author"]==user))){
        $("#seltit").append('<option idx="'+vettore_regole[i]["id"]+'" id="op_'+i+'" value="'+vettore_regole[i]["title"]+'">'
    }
}
```

Figura 4.5: vengono caricati i metadati delle regole: tutte le regole pubbliche, e private ma solo dell'autore loggato

Per ogni regola viene caricata una tabella riassuntiva con i relativi metadati: titolo, autore, descrizione e status.

Al caricamento della pagina, oltre al controllo per il documento, viene cercata anche la regola specificata nella variabile rule nell'URI. Una volta trovata viene applicata al SunBurst richiamando la funzione *evaluateJS()*.

```
$("#CSS_included").remove();
var CSS_text = CSS_editor.getValue();
$('head').append('<style type="text/CSS" id="CSS_included"
>' + CSS_text + '</style>');
var doco_text = JS_editor.getValue();
eval("function draw_doco() { try{"+ doco_text + "}
catch (e) {..}
```

Viene preso dagli editor JS e CSS il codice della regola esaminata. Viene incluso il CSS con il tag *<style>* e valutato il codice JavaScript, rappresentato come una stringa. Con la struttura *try and catch* si riesce a catturare errori nel JS di tipo: *TypeError*, *RangeError*, *SyntaxError*, *EvalError* e *ReferenceError*.
Le classi create dal coding della regola vengono assegnate attraverso la funzione *addClassToPath()*.

Ogni regola può essere editata. Una volta scelta la regola il sistema carica nei rispettivi editor il codice JavaScript e CSS, così da poterla editare, testare o salvare. Al momento del caricamento viene sempre richiamata la funzione *evaluateJS()* ; se la regola è valida e sintatticamente corretta verrà applicata al SunBurst, altrimenti verrà comunicato il tipo di errore.

Sono inoltre possibili diverse funzionalità per l'utente, come salvare, creare o copiare una regola.
Chiaramente ognuna di esse ha bisogno dell'appoggio del database, e saranno degli script php a caricare i dati nel db. In JavaScript vengono gestite le chiamate asincrone a questi script. Prendiamo ad esempio la funzione *save()*;

```
$("#save").submit(function(event) {
  /* stop form from submitting normally */
  event.preventDefault();

  /* get the action attribute from the <form action=""> element */
  var $form = $( this ),
      url = $form.attr( 'action' );

  /* Send the data using post with element id name and name2*/
  var posting = $.post( url, { title: $('#savetitle').val(), description: $('#savedesc').val(), .. });

  /* Alerts the results */
  posting.done(function( data ) {
    window.location.href = data;
  });
});
```

Figura 4.6: funzione per il salvataggio dei progressi di una regola

event.preventDefault() non permette l'attivazione dell'azione predefinita dell'evento; vengono passate così al file.php, nella variabile url, tutte le variabili necessarie da caricare nel database restando poi in attesa di una risposta.

Questo metodo viene usato per tutte le funzioni, tranne che per il caricamento dei documenti. In quel caso si richiama il file.php attraverso una chiamata Ajax nel modo seguente.

```
$('#uploadSubmitBtn').on('click', function() {
      var file_data = $('#uploadBrowseBtn').prop('files')[0];
      var form_data = new FormData();
      form_data.append('file', file_data);
      $.ajax({
          url: 'php/caricadoc.php',
          dataType: 'json',
          cache: false,
          contentType: false,
          processData: false,
          data: form_data,
          type: 'post',
          success: function(data){
```

Figura 4.7: chiamata Ajax per il caricamento di documenti

Viene catturato il file selezionato e passato a caricadoc.php per tutte le analisi necessarie, illustrate nel paragrafo successivo.

Una volta ricevuta la risposta questa viene analizzata, verificando che non sia un codice di errore.
Se i documenti caricati rispettano le regole stabilite viene effettuata una richiesta HTTP dei documenti presi in esame e caricati temporaneamente.

```
for(i=0; i<data.length; i++){
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            count_contr_gr=count_contr_gr+1;
           myFunction(this, count_contr_gr);
        }
    };
    xhttp.open("GET", "data/xml/"+data[i]+"", true);
    xhttp.send();
}
function myFunction(xml, idx) {
    var x, xmlDoc, a;
    xmlDoc = xml.responseXML;
    txt = "";
    x = xmlDoc.getElementsByTagName('article');
    if(idx==1){
        version = x[0].getAttribute('version');
        xmlns = x[0].getAttribute('xmlns');
        console.log("ecco: "+version);
    }
    else{
        if((version!=x[0].getAttribute('version'))||(xmlns!=x[0].getAttribute('xmlns'))){
            controllo_gruppo=0;
        }
    }
    //count_group(version,xmlns, idx);
}
```

Figura 4.8: altra richiesta HTTP: prendo in ogni documento i tag "article", "version", "xmlns" per individuarne il gruppo

Diversamente dall'altra si deve controllare che ogni documento che si vuole caricare faccia parte dello stesso gruppo. Si controlla la versione (*version*) e il namespace (*xmlns*) e se solo uno risulta differente, con un'altra chiamata *ajax* si richiama uno script php il quale elimina tutti i documenti esaminati. In caso contrario si comunica lo stato di accertamento e mantiene i documenti caricati.

4.3.2 Server side: PHP

PHP è un linguaggio per lo scripting server-side, ovvero un linguaggio che risiede in un server in remoto e che in fase di esecuzione interpreta le informazioni ricevute da un client grazie al Web server, le elabora e restituisce un risultato al client che ha formulato la richiesta.

Il PHP in DocuDipity è fondamentale per la comunicazione e lo scambio di dati con il database, attraverso query SQL, e per la manipolazione di dati nel server, in questo caso i documenti.
E' bene notare che in quasi tutti gli script richiedo una connessione al database. Se si volesse installare DocuDipity su un altro server andrebbero cambiate le credenziali per la connessione, servendosi di un db diverso, modificando ogni file. Sarebbe un lavoro di ricerca troppo lungo e il sistema sarebbe poco flessibile.
Per rendere più modulare l'installazione di DocuDipity in qualsiasi server ho creato un file *config-db.php* dove sono settati i valori per connettersi al database di riferimento. Così facendo l'utente dovrà configurare solo questo file e non ogni file.php che presenta una connessione al database.

```php
<?php
    // set yours credentials
    session_start();
    $_SESSION["host"]="localhost"; //hostname
    $_SESSION["username"]="***"; // your user name
    $_SESSION["password"]="***"; // your password
    $_SESSION["db_name"]="docudipity"; // db name
?>
```

Figura 4.9: file di configurazione del DB.

I due file principali sono *index.php* e *gestione-regole.php*. Oltre alla struttura HTML della pagina viene eseguito anche uno script php il

quale richiede al database tutte le regole presenti pubbliche e private con i relativi metadati. Queste regole verranno salvate in un vettore e passate al lato client. Inoltre, sempre in questo script, vengono scanditi tutti i file XML, cioè i documenti, e li salvo in una variabile, senza leggerne il contenuto. Come abbiamo visto verranno aperti e analizzati attraverso una richiesta HTTP e suddivisi per gruppi:

```
$dir    = 'data/xml';
$files1 = scandir($dir);
```

Essendo DocuDipity una struttura RESTful, prendo dall'URI le variabili che identificano il documento e la regola correnti per poi, come spiegato prima, lasciarli in elaborazione al JavaScript.

```
$url=$_SERVER['REQUEST_URI'];
$parts = parse_url($url);
parse_str($parts['query'], $query);
$curr_document=$query["doc"];
$curr_rule=$query["rule"];
```

Viene eseguito uno script php diverso per ogni operazione richiesta dal client: registrarsi, effettuare il login, salvare i progressi di una regola, crearla una, copiarla, caricarla dal pannello di gestione ed eliminarla. Ognuno di questi script ha compiti diversi, ma sono tutti strutturati nella stessa maniera:

- Per prima cosa si esegue una connessione al database ottenendo le credenziali dal file *config-db.php*.
- In base all'operazione, si esegue la query per inserire o richiedere dati al database.
- Infine si restituiscono i dati ottenuti al client per l'elaborazione.

Ecco un esempio di script in php :

```php
$conn = new mysqli($host, $username, $password, $db_name);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$stmt2= "SELECT id FROM rules";
$result = $conn->query($stmt2);

if ($result->num_rows > 0) {
    // output data of each row
    $j=0;
    while($row = $result->fetch_assoc()) {
        $j++;
    }
} else {
    echo "0 results";
}
$id=$j;

$stmt = $conn->prepare("INSERT INTO rules (js, css, title, id, status, description, author) VALUES (?, ?, ?, ?, ?, ?, ?)");
if ( false===$stmt ) {
  //header('refresh: 3; url = ../index.php');
  echo "Failed to prepare the query";
}
$stmt->bind_param('sssddss', $js, $css, $title, $id, $status, $description, $author);
if (!$stmt->execute()) {
    //header("refresh: 2; url=../index.php");
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}
//header('refresh: 2; url = ../index.php');
echo "index_logged.php?doc=".$doc."&rule=".$title."&pass=0";

$conn->close();
```

Figura 4.10: esempio file php per il salvataggio di una regola

Questo esegue una copia di una regola già esistente. Si possono notare tutti i punti salienti spiegati al passo precedente. La connessione al database, l'esecuzione della query, e la risposta.

Caricamento documenti

DocuDipity è un sistema che lavora contemporaneamente su regole e documenti. Abbiamo visto nel paragrafo precedente come un utente possa creare e fare operazioni sulle regole. Ora vedremo come il sistema permette all'utente di caricare un set di documenti.

Analizziamo passo passo come avviene il caricamento:
Il client, tramite una chiamata ajax, richiama il file *caricadoc.php* passandogli il file caricato dall'utente e resta in attesa di una risposta.
caricadoc.php analizza il file ricevuto grazie alla variabile -FILE: Un array associativo di elementi caricati allo script corrente tramite il metodo HTTP POST. Array composto da "name" contenente il nome,

"type" che ne specifica il tipo, "tmp-name" che mostra il nome temporaneo del file nel server, "error" che da un codice di errore e "size" che ne specifica la grandezza.
Per prima cosa controllo la grandezza del file non superi 10MB,

```
if ($_FILES["file"]["size"] > 10000000) {
    echo JSon_encode("Sorry, your file is too large.");
    $uploadOk = 0;
}
```

e che, specificando il percorso dove il file verrà caricato, sia di tipo .zip.

```
$target_dir = "../data/intermezzo/";
$target_file = $target_dir . basename($_FILES["file"]["name"]);
$FileZip = pathinfo($target_file,PATHINFO_EXTENSION);
if($FileZip != "zip" ) {
    echo JSon_encode("Sorry, only ZIP files are allowed.");
    $uploadOk = 0;
}
```

Se solo una delle due condizioni non è rispettata la variabile di controllo "uploadOk" viene settata a 0 e si interrompe l'eseuzione. In caso contrario il file viene spostato nella directory specificata.

```
move_uploaded_file($_FILES["file"]["tmp_name"], $target_file)
```

Il file zip viene aperto e la funzione controlla che ogni elemento sia di tipo .xml. Se così non fosse tutti i file vengono eliminati e il sistema restituisce un un segnale di errore all'ajax in attesa. Una volta passato questo controllo verifica che non ci sia già uno dei documenti nella directory che l'utente vuole caricare.

```
$countfile=0;
for ($i=0; $i<count($files); $i++){
  if(file_exists("../data/xml/".$files[$i])){
    $file2[$countfile]="this file: '".$files[$i]."'
    already exists" ;
    $countfile++;
    $controllo2=0;
  }
}
```

Se solo uno dei file esiste già vengono eliminati tutti e viene restituito un codice di errore specificando quali file (*file2[]*) sono già presenti nel sistema.

Superato anche questo test i file caricati vengono spostati dalla directory di appoggio alla directory xml dove sono contenuti tutti i documenti restituendoli al client.

```
$name_dir = basename($_FILES["file"]["name"]);
...
...
...
if($controllo2==1){
  for ($i=0; $i<count($files); $i++){
    $old="../data/intermezzo/".$name_dir."/".$files[$i];
    $new="../data/xml/".$files[$i];
    rename($old , $new);
  }
  echo JSon_encode($files);
}
```

Come abbiamo visto prima, ora il client riceve i file caricati e attraverso una richiesta HTTP controlla che siano tutti dello stesso gruppo. In caso positivo il sistema manda un alert all'utente di avvenuto caricamento. In caso negativo per ognuno di essi il sistema richiama il file *delete-xml.php* e li elimina dalla directory.

```
<?php
        $file_xml=$_POST["file"];
        unlink("../data/xml/".$file_xml);
        echo JSon_encode($file_xml);
?>
```

SQL in PHP

Uno dei moduli principali di un'applicazione moderna è, generalmente, quello che riguarda la gestione dei dati, che può essere basato sull'utilizzo di varie tecnologie. Si può pensare, per piccoli progetti, di memorizzare i dati in file di testo, più o meno strutturati; ma per i software professionali e di grandi dimensioni come DocuDipity, è quasi sempre impossibile prescindere dall'utilizzo di un database.
Per interagire con tale classe di database uso SQL(acronimo che sta per Structured Query Language) [15].

Come accennato prima in quasi ogni script PHP utilizzo SQL per la comunicazione con il DB.
Vengono eseguite diversi tipi di query che ora mostrerò.

Per ottenere tutte le regole con i relativi metadati si esegue una SELECT, chiedendo al database di prendere i dati specificati nella tabella rules: Dopo li assegno all'array *vettore-regole[]* che verrà poi passato al client.

Un altro tipo di query invece per il salvataggio di una regola. Concettualmente bisogna eseguire un "UPDATE" sulla regola esistente dell'utente con i nuovi dati modificati (JS e CSS) specificando, tramite "title" e "author" su quale riga fare l'update.

```
$query= "UPDATE rules
        SET JS = ?, CSS = ?
        WHERE title = ? and author = ?";

$stmt->bind_param('ssss', $JS, $CSS, $title, $author);
$stmt->execute();
```

Per la copia o la creazione di una regola è necessario un "INSERT INTO" con le variabili passate tramite bind param.
I "bind parameter" servono per filtrare automaticamente le stringhe da passare alla query (eseguono già il filtraggio dei caratteri dannosi), ma non solo. Essi servono anche come selettori per processare al meglio e costruire la propria query.

```
$stmt = $conn->prepare("INSERT INTO rules (js, css, title, id, status, description, author) VALUES (?, ?, ?, ?, ?, ?, ?)");
if ( false===$stmt ) {
  //header('refresh: 3; url = ../index.php');
  echo "Failed to prepare the query";
}
$stmt->bind_param('sssddss', $js, $css, $title, $id, $status, $description, $author);
if (!$stmt->execute()) {
    //header("refresh: 2; url=../index.php");
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}
//header('refresh: 2; url = ../index.php');
echo "index_logged.php?doc=".$doc."&rule=".$title."&pass=0";
```

Figura 4.11: query SQL "INSERT INTO" con la funzione bind-param per il passaggio di variabili

Per eliminare una regola dal pannello di gestione si richiama un file.php il quale prima esegue una "SELECT" per trovare l'id (ricordiamo essere univoco) della regola con il titolo passato come parametro, e poi esegue

una "DELETE" di tutta la riga specificandola grazie all'id appena trovato:

```
$sql = "DELETE FROM rules WHERE id='".$id."' ";
```

Una volta eseguita questa operazione, per non avere "id" in modo non ordinato nella tabella rules, vengono riassegnati eseguendo un ulteriore Update riordinandoli uno per uno:

```
for($i; $i<$idd; $i++){
        $query = "UPDATE rules
                SET id = ?
                 WHERE id= ?";
        $stmt = $conn->prepare($query)
                $stmt->bind_param('dd', $id1, $id)
                $stmt->execute()
        $id1++;
        $id++;
}
```

Capitolo 5

Discussione e conclusioni

Abbiamo analizzato diversi sistemi che operano nell'ambito dello studio e dell'analisi della struttura gerarchica dei documenti evidenziandone i punti deboli e i limiti che presentavano. DocuDipity è un sistema che li migliora, trasformandoli da punti deboli a punti di forza. Per esempio risolve il problema della rappresentazione strutturale di un documento: rappresenta strutture complesse e ramificate in uno spazio ristretto attraverso la visualizzazione SunBurst, sfruttandolo anche per analisi specifiche e anche complesse mostrate ed evidenziate in maniera chiara e innovativa.

5.1 Limiti

DocuDipity tuttavia presenta dei limiti:
Il visualizzatore ipertestuale prende in input qualsiasi documento XML e fornisce una singola visualizzazione basata sul vocabolario del documento scritto. Se da un lato, un tale approccio consente di aprire ed esplorare qualsiasi documento, dall'altro la visualizzazione non è completamente precisa.

Il processo di trasformare ipotesi in condizioni, spesso anche complesse, utilizzando le tecnologie appena citate è sicuramente un passo avanti, ma restringe la cerchia di utenti. Infatti DocuDipity è sistema inutilizzabile per chiunque non sia esperto di JavaScript e CSS.
Il modulo per il caricamento dei documenti è sicuramente limitato. Il sistema è in grado di gestire solo documenti dello stesso tipo (si rimanda al capitolo 4 per maggiori informazioni). Se in un set di documenti da caricare solo uno appartiene ad un altro gruppo, il sistema rifiuta e non ne carica nessuno.
Abbiamo detto che DocuDipity è un sistema repsonsive, che si adatta a vari tipi di schermo fino a una dimensione prefissata. Da "mobile" però DocuDipity non è ottimizzato, anche perché è progettualmente difficile visualizzare in uno schermo di piccolissime dimensioni la vista SunBurst e il pannello di lettura insieme, e l'editor CSS e JS allineati, rendendo tutto intuitivo.

5.1.1 Come migliorare ed evolvere DocuDipity

Ecco vari punti in cui propongo dei miglioramenti:

Scrivere una regola non è complicato ma potrebbe essere ulteriormente semplificato. Finora, l'editing di regole è utilizzabile da una cerchia ristretta di utenti, coloro che hanno le basi della programmazione JavaScript e CSS.
Tuttavia, si potrebbe immaginare uno strato intermedio che prende in input le regole di visualizzazione scritte in un linguaggio semplificato e le converte in codice JavaScript e CSS in modo trasparente come, per esempio, una semplice espressione XPath.
Oppure un modulo dove gli utenti inesperti sono guidati alla scrittura di regole attraverso linee guida, o ancora più semplicemente, ma anche più utopisticamente, riuscire a decifrare parole del linguaggio naturale ed applicarlo all'algoritmo.

DocuDipity è totalmente indipendente da una struttura particolare, e possono essere utilizzati altri strumenti, anche al posto di quelli inclusi nella implementazione corrente come EXT-JS [17], un framework JavaScript utile a realizzare GUI complesse per web application, ma non solo. Con *Ext.JS* si possono creare pagine con layout avanzati anche se non si possiedono particolari competenze CSS; si possono agganciare funzionalità AJAX di aggiornamento per le nostre interfacce; creare moduli wizard o textarea con testo enriched (formattato come lo formatterebbe un programma di videoscrittura); agganciare le nostre web application a ulteriori strumenti (MooTools, JQuery, Prototype, Google Gears e molti altri) in modo da avere rapidamente software decisamente vicini alle applicazioni desktop con cui lavoriamo quotidianamente.
Un'altra opzione è quella di integrare un visualizzatore dipendente dalla linguaggio che aiuta i lettori a scoprire il contenuto del documento. Nel caso di articoli scientifici, per esempio, si può integrare il visualizzatore RASH [10]. RASH è un sottoinsieme di HTML che comprende solo una trentina di elementi, organizzati in un modo completamente pattern-based e che è stato progettato per agli articoli scientifici. Ci sono diverse alternative valide a RASH e l'idea di usare HTML per la scrittura, la presentazione e la revisione di documenti sta rapidamente guadagnando rilevanza
L'obbiettivo, indipendentemente dalle tecniche che si adottano, è quello di estrapolare elementi nascosti da documenti e rappresentarne la struttura. Grazie a DocuDipity possiamo fare diverse analisi sulle abitudini di gruppi di autori.
Come mostrato prima DocuDipity è un sistema con una cerchia di utenti ristretta. Infatti vorremmo anche rendere DocuDipity facile da usare per altri studiosi, senza esperienza né competenze in informatica come, per esempio, gli studiosi di scienze umane, per scoprire cose di loro competenza.

Rispetto alla versione iniziale DocuDipity ha fatto diversi evoluzioni. Uno degli obiettivi iniziali era quello di creare un ambiente di lettura condivisa in cui gli utenti possono sperimentare nuove regole, scriverle in modo incrementale e riutilizzarle. Con la realizzazione di questo si sono aperte prospettive molto interessanti: gli esperimenti iniziali di un utente potrebbero essere seguiti da altri, e anche intuizioni parziali potrebbero dare input e sollecitare nuove idee. L'attenzione, ancora una volta, è il serendipity: visioni alternative sul medesimo documento possono fare gli utenti a scoprire caratteri imprevisti e percorsi.

L'applicazione di DocuDipity è solo una delle possibili direzioni da esplorare. Facendo scoperte significative, si possono espandere ad altri documenti o addirittura ad altri domini.

Bibliografia

[1] Ragaad AlTarawneh and Shah Rukh Humayoun. Visualizing software structures through enhanced interactive sunburst layout. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '16, pages 288–289, New York, NY, USA, 2016. ACM.

[2] Balsamiq. Balsamiq. https://balsamiq.com.

[3] Benjamin B. Bederson. Photomesa: A zoomable image browser using quantum treemaps and bubblemaps. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, UIST '01, pages 71–80, New York, NY, USA, 2001. ACM.

[4] M. C. Chuah. Dynamic aggregation with circular visual designs. In *Information Visualization, 1998. Proceedings. IEEE Symposium on*, pages 35–43, 151, Oct 1998.

[5] CodeMirror. Codemirror. https://codemirror.com.

[6] d3. d3: Data-driven documents. https://d3js.org.

[7] Angelo Di Iorio, Silvio Peroni, Francesco Poggi, Fabio Vitali, and Paolo Ciancarini. Docudipity: disclosing habits in scholar-

ly writing. In *DocuDipity: disclosing habits in scholarly writing*, 2016.

[8] Angelo Di Iorio, Silvio Peroni, Francesco Poggi, Fabio Vitali, and David Shotton. Recognising document components in xml-based academic articles. In *Proceedings of the 2013 ACM Symposium on Document Engineering*, DocEng '13, pages 181–184, New York, NY, USA, 2013. ACM.

[9] getbootstrap. Bootstrap. https://getbootstrap.com.

[10] Angelo Di Iorio, Andrea Giovanni Nuzzolese, Francesco Osborne, Silvio Peroni, Francesco Poggi, Michael Smith, Fabio Vitali, and Jun Zhao. Poster of the ISWC 2015 demo paper "The RASH Framework: enabling HTML+RDF submissions in scholarly venues". 10 2015.

[11] jQuery. jquery. https://jquery.com.

[12] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[13] Hao Lü and James Fogarty. Cascaded treemaps: Examining the visibility and stability of structure in treemaps. In *Proceedings of Graphics Interface 2008*, GI '08, pages 259–266, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.

[14] Michael J McGuffin and Jean-Marc Robert. Quantifying the space-efficiency of 2d graphical representations of trees. *Information Visualization*, 9(2):115–140, June 2010.

[15] MySql. Sql. https://dev.mysql.com/doc/.

[16] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 189–194, New York, NY, USA, 1991. ACM.

[17] sencha. Ext-js. https://www.sencha.com/products/extjs.

[18] Ben Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, January 1992.

[19] Ben Shneiderman and Martin Wattenberg. Ordered treemap layouts. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, INFOVIS '01, pages 73–, Washington, DC, USA, 2001. IEEE Computer Society.

[20] W3C. Xml. https://www.w3.org/standards/xml/.