Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Gestione di eventi utente in un editor di documenti legali basato su Akoma Ntoso

Relatore:
Chiar.mo Prof.
Vitali Fabio

Presentata da:
Poggioli Mattia

Correlatore:
Chiar.ma Prof.
Palmirani Monica

III Sessione
Anno Accademico 2022/2023

"Pensare è facile,

agire è difficile,

e mettere i propri pensieri in pratica

è la cosa più difficile del mondo"
- Goethe.

Abstract

Questa tesi analizza l'editor avanzato per la redazione di documenti legali basati sullo standard Akoma Ntoso SIMPLEX, illustrando i risultati ottenuti in seguito al lavoro di progetto effettuato con l'obiettivo di valutare e sviluppare miglioramenti sull'utilità, efficacia ed efficienza delle operazioni fornite dall'applicativo. L'utilizzo dello standard XML Akoma Ntoso in questo contesto è importante in quanto riesce a fornire un metodo di rappresentazione strutturale per documenti legislativi, parlamentari e giudiziari di ogni tipo e giurisdizione. Questa particolare tipologia di documenti richiede particolare attenzione, richiedendo delle operazioni delicate per la loro corretta modifica. Risulta quindi importante sviluppare e valutare operazioni sempre più affidabili, attività oggetto della seguente dissertazione. L'attività di progetto svolta si è focalizzata sulla gestione dei principali eventi utente attraverso un approccio metodologico e l'implementazione di meccanismi avanzati, sia per nuove funzionalità che per il controllo della correttezza del documento. Grazie alla collaborazione in fase di sviluppo e test di un gruppo di esperti del settore, l'attività ha portato a risultati significativamente positivi per quanto riguarda l'utilità, l'efficacia e l'efficienza delle operazioni interne all'applicativo.

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'era digitale odierna è caratterizzata da una rapida e costante evoluzione in cui tutte le tematiche e le problematiche della società vivono grandi cambiamenti. Questo accade generalmente all'interno di in ogni ambito: nel mondo lavorativo, in cui anche i mestieri più tradizionali vengono rinnovati e nuove occupazioni nascono con l'avvento della tecnologia; nella società, che vive cambiamenti radicali nel nome di una migliore qualità della vita o di comodità sempre nuove; nel mondo dell'amministrazione, in cui l'innovazione diventa necessaria per far fronte alla popolazione in crescita e alle sfide che ne derivano.

In questo contesto l'interazione tra gli utenti e le applicazioni sofisticate ha assunto un ruolo centrale, specialmente se si prendono in esame strumenti avanzati come editor per la produzione e manipolazione di documenti legali. Si tratta di documenti importanti per la società e con strutture complicate, ma che vanno rispettate per assicurare una comunicazione diretta, funzionale e completa.

Questo argomento è ancora più importante se si prendono in considerazione controversie legali anche internazionali, con legislazioni differenti o comunicazioni tra governi. Risulta quindi chiara l'importanza di una costante ricerca di soluzioni effi-

caci e utili per migliorare e agevolare l'utilizzo di tali strumenti: questa tesi propone l'analisi di uno di questi software, delle problematiche associate e di alcuni sviluppi effettuati.

Un elemento cruciale in questo contesto è la gestione degli eventi utente, che comprende tutte le azioni effettuabili dall'utente attraverso operazioni da tastiera o pulsanti visualizzati a schermo. Questo aspetto riveste particolare importanza nella redazione di documenti legali, dove la precisione e l'efficienza sono di primaria importanza.
La corretta ed efficiente gestione degli eventi utente diventa quindi un elemento chiave per ottimizzarne l'esperienza e favorire operazioni veloci, dirette ed efficaci. Basti pensare a quanto risulta comodo riuscire ad aggiungere nuove voci ad un elenco numerato semplicemente premendo il tasto "invio", lasciando quindi al software il compito di mantenere un comportamento coerente. In generale oramai questa operazione può sembrare un esempio di poca importanza, in quanto tutti i principali editor di testo permettono di avere un tale comportamento, ma questo ci deve far capire che anche le operazioni apparentemente più semplici devono essere gestite utilizzando soluzioni robuste ed efficaci, in particolar modo quando si lavora con strutture complesse che devono rispettare proprietà ben definite.

La presente tesi si propone di analizzare ed approfondire le operazioni essenziali e particolarmente efficaci/funzionali nell'utilizzo di editor avanzati per la redazione di documenti legali, giuridici e parlamentari conformi ad uno specifico standard: lo standard *Akoma Ntoso*. Questo standard ormai ampiamente diffuso e utilizzato è parte integrante del tema della tesi, ed è integrato all'interno di un applicativo software appositamente sviluppato chiamato SIMPLEX (Simple Lex) di cui ho potuto sviluppare alcune nuove funzionalità nell'ottica di proporre soluzioni mirate e strategie di gestione documentale ottimizzate. Lo sviluppo ha anche riguardato

l'implementazione di features di QOL (Quality Of Life) tramite l'integrazione di funzionalità avanzate del linguaggio utilizzato e dello stack di tecnologie scelto, in costante evoluzione e integrazione. Risulta particolarmente importante conoscere la storia di questo sviluppo, in quanto permette di capire meglio le necessità ed il percorso che ha portato alla nascita di questo standard e allo sviluppo del software SIMPLEX, applicazione web su cui si basa il lavoro di questa tesi.

La nascita di questa nuova applicazione ha come principale obiettivo quello di modernizzare l'attuale software di manipolazione di documenti *Akoma Ntoso* LIME. LIME è un editor di markup open-source, parametrico e indipendente da qualsiasi linguaggio XML (eXtendible Markup Language).
L'idea alla base di LIME è che è possibile astrarre un linguaggio XML assegnando ad ognuno dei suoi componenti base uno dei pattern utilizzati dal linguaggio *Akoma Ntoso*. Questa astrazione può essere descritta con parametri definiti all'interno di appositi file di configurazione. In questo modo in LIME è possibile utilizzare altri linguaggi XML semplicemente scrivendo un insieme di file JSON che ne descrivono una implementazione adeguata. Questo significa che ogni eventuale personalizzazione non necessita di competenze specifiche in nessun linguaggio di programmazione, ma richiede soltanto la scrittura di un nuovo file JSON contenente le regole necessarie e le convenzioni da adottare per il nuovo formato che si intende introdurre. [Cer13]

Per quanto sia ancora un editor estremamente avanzato e ricco di funzionalità, LIME necessità una rivisitazione del proprio codice sorgente. Le principali cause che rendono difficile il processo di aggiunta di nuove funzionalità complesse sono le tecnologie adottate per il suo sviluppo, tecnologie che hanno portato ad una implementazione ricca di dipendenze tra componenti per la gestione della logica dell'applicazione che inoltre rendono difficile il passaggio ad un differente framework. La curva di apprendimento dell'utilizzo delle tecnologie adottate e l'ampia codebase

aggiungono un'ulteriore difficoltà nella comprensione del codice, incrementando il tempo necessario agli sviluppatori per l'introduzione di nuove funzionalità.

Con l'aumentare dell'importanza degli editor di documenti legali nel panorama digitale, emerge la necessità di esaminare con attenzione le complessità connesse alla gestione degli eventi utente. La gestione accurata di tali eventi costituisce una sfida importante, poiché rappresenta il punto principale per garantire l'accuratezza, la velocità di redazione e l'integrità dei documenti legali. Una delle ulteriori sfide in questo campo è data dalla varietà delle legislazioni presenti e delle tipologie di documento che necessitano di essere redatte. Per questo motivo si ha bisogno di trovare uno standard in grado di gestire differenti casi di utilizzo senza la necessità di realizzare software personalizzati per ogni specifica circostanza o contesto. Esistono diversi standard XML applicati alla legge: molti di questi sono pensati soltanto per la pubblicazione sul Web; altri permettono di ottenere una rappresentazione del documento giuridico in forma strutturata ma sono realizzati per lavorare con un singola tipologia di documento o per un singolo paese e quindi difficili da utilizzare con giurisdizioni diverse.[PV12]

Akoma Ntoso propone quindi uno standard XML per la rappresentazione di documenti legislativi, giudiziari e amministrativi in forma strutturata, con l'importante particolarità di non avere vincoli sulla tipologia e il paese dei documenti che è possibile rappresentare, permettendone l'utilizzo per qualsiasi documento dell'ambito legale. L'adozione di questo standard introduce chiarezza semantica nei testi normativi, agevolando la comprensione e aprendo le porte ad elaborazioni automatiche. Tuttavia, la gestione degli eventi utente in un editor che lavora con documenti *Akoma Ntoso* introduce sfide peculiari, richiedendo un'analisi approfondita per garantire un'implementazione corretta, efficace ma anche flessibile vista la natura altamente personalizzabile dello standard. Questa dissertazione nasce dalla necessità di svi-

luppare meccanismi avanzati per una corretta gestione degli eventi utente in SIMPLEX, un'applicazione di editing di documenti legali basata sullo standard *Akoma Ntoso*. L'indagine approfondita in questo ambito mira a migliorare l'esperienza utente fornendo apposite funzionalità, con l'obiettivo di migliorare in modo sostanziale l'efficienza e la precisione degli editor utilizzati nel contesto digitale dei documenti legislativi.

Gli obiettivi di questa tesi partono quindi con il fornire un'analisi approfondita sull'importanza degli editor di documenti legali nell'attuale contesto digitale, con un focus sulla rilevanza di una gestione efficiente ed efficace. Il ruolo di *Akoma Ntoso* è centrale e verrà analizzato adeguatamente come argomento chiave in questo contesto in quanto standard per la rappresentazione di testi legislativi.

Si tratta di un formato tanto integrato e importante per l'ambito in esame pensato per aiutare la redazione di documenti, ed è per questo motivo che è importante studiarlo per comprendere ed estrapolare modi di utilizzo o caratteristiche che possano aiutare durante l'implementazione delle operazioni atte alla redazione dei documenti. In particolare, l'analisi si concentra sulle sfide specifiche riguardanti la gestione degli eventi utente nel contesto degli editor che utilizzano lo standard *Akoma Ntoso*. Le convenzioni e strutture necessarie ad un documento legislativo sono notoriamente complesse, articolate e impegnative da realizzare e mantenere, pertanto sottolineano l'utilità di porre una attenzione speciale agli aspetti interattivi, rendendo importante lavorare a funzionalità che si integrino alle operazioni svolte dagli utenti arricchendole, velocizzandole e facilitandole. Sarà prestata particolare attenzione agli sviluppi in questa direzione studiati ed implementati durante l'attività di progetto.

In sintesi, la tesi vuole esplorare le dinamiche legate agli editor di documenti legali, con un focus sullo standard *Akoma Ntoso* implementato nel software SIMPLEX e sulle sfide connesse alla gestione degli eventi utente, attraverso l'esperienza e i risultati ottenuti durante le fasi di test da parte di utenti esperti.

Capitolo 2

Akoma Ntoso

2.1 Panoramica su Akoma Ntoso

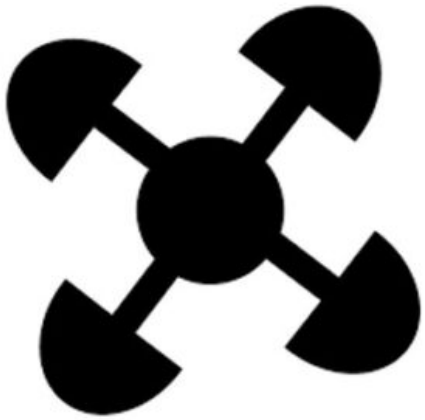


Figura 2.1: Logo di Akoma Ntoso

Akoma Ntoso (in italiano "*cuori uniti*") è uno standard XML per la rappresentazione di documenti giudiziari, parlamentari e legislativi.
Vede la sua nascita tra il 2004 e il 2005 grazie ad un progetto del dipartimento dell'economia e degli affari sociali delle Nazioni Unite, che aveva come scopo quello

di supportare il parlamento Africano per migliorare la gestione delle loro procedure legali, favorendo una democrazia efficiente con l'ausilio di strumenti digitali.
L'acronimo ha come significato *Architecture for Knowledge-Oriented Management of African Normative Texts using Open Standards and Ontologies*, e la sua origine deriva dal popolo akan[1].

Il simbolo nella figura 2.1 oltre ad essere il logo dello standard assume il significato di comprensione ed accordo per la popolazione akan. [VZ07]

2.2 Utilizzo nel contesto legislativo

Il primo sviluppo di un sistema digitale per la gestione delle questioni legislative risale agli anni '60, quando vennero introdotti i primi database di documenti legali. Tra la fine degli anni '90 e l'inizio degli anni 2000 vi è stato un aumento dell'impatto sulla gestione e sull'integrazione dei processi di produzione e applicazione della legge. Approfittando delle opportunità offerte da Internet, il campo d'azione viene esteso alle comunicazioni tra le organizzazioni legali e il loro pubblico, diventato un aspetto importante e significativo come risorsa utile per l'*e-government*.[2]

Gli obiettivi dell'e-government sono quello di migliorare l'informazione e la fornitura di servizi, incoraggiare la partecipazione e rendere il governo più responsabile, trasparente ed efficace.
Già dai primi anni del 2000 Internet è diventato la principale fonte di conoscenza giuridica per i cittadini, diventando rapidamente anche la principale fonte di infor-

[1] Gli akan sono un gruppo etnico dell'Africa Occidentale formato da diverse popolazioni.

[2] Il termine e-government implica l'adozione di strumenti digitali e soluzioni tecnologiche per modernizzare e trasformare i processi governativi tradizionali.
(`http://qualitapa.gov.it/sitoarcheologico/relazioni-con-i-cittadini/open-government/e-government/index.html`)

mazione per gli avvocati, poiché le fonti giuridiche precedentemente esistenti stavano migrando su Internet. Si vide, inoltre, la nascita di nuove fonti di informazione giuridica mirate a migliorare la ricezione di informazioni.
Nell'era di Internet un sistema di informazione parlamentare deve avere una doppia faccia: una prima che fa riferimento al lato interno del parlamento e ha come scopo quello di supportare tutte le attività parlamentari, la seconda che si relaziona con l'esterno e fornisce ai cittadini informazioni sulle attività e sui risultati del parlamento, nonché opportunità di partecipazione alle attività parlamentari. [BFP+08]

2.3 Definizione e caratteristiche

Akoma Ntoso permette di redigere e visionare documenti composti da appositi elementi di markup che forniscono informazioni aggiuntive sulla categorizzazione del contenuto, sotto forma di etichette esplicative. Questi blocchi permettono una visualizzazione del contenuto del documento in forma strutturata e compartimentata seguendo convenzioni strutturali. Queste convenzioni possono anche essere diverse tra stati, visto che con gli stessi strumenti a disposizione è possibile creare elementi molto diversi tra loro, ma sempre compatibili con questo standard.

Ogni tipo di elemento è formato da un "*name*" e un "*pattern*".
Il "name" rappresenta il tipo di elemento e svolge il ruolo di etichetta descrittiva, indicando la natura del contenuto selezionato.
Il "pattern", d'altra parte, definisce le proprietà di una classe e la sua grammatica, fornendo un modello di contenuto e specificando il comportamento e la gerarchia della classe rispetto ad altre classi . Le tipologie di pattern maggiormente trattate nello sviluppo delle azioni definite successivamente sono *container, hcontainer, block* ed *inline*. La prima rappresenta una categoria di elementi unici nella struttura del documento, la seconda specifica un elemento generalmente formato da un numero

ed un titolo contenente o del testo proprio o una serie di figli, la terza è pensata per contenere le parti testuali del documento; mentre la quarta riguarda un elemento preposto per l'arricchimento del contenuto testuale. Quest'ultimo pattern viene solitamente utilizzato per contrassegnare frasi o parole di una certa importanza all'interno del documento.[VZ07]

Si tiene a precisare che questi non sono gli unici tipi di pattern gestiti dallo standard, ma sono quelli più usati nel contesto specifico preso da me in esame.

In generale si possono avere diverse tipologie di utilizzi delle etichette che possono essere associate ai seguenti elementi testuali:

- **Intero documento**: l'etichetta può essere associata all'intero documento, fornendo una descrizione della tipologia del testo normativo;
- **Parti principali:** come Prefazione, Conclusioni, ecc. Queste rappresentano elementi strutturali fondamentali del documento e permettono di fornire una struttura gerarchica;
- **Parti secondarie:** come Articoli, Paragrafi, ecc. Queste rappresentano elementi di base del documento inseribili all'interno dei contenitori principali, anch'esse organizzate secondo una gerarchia;
- **Singole frasi o parole**: ad esempio Riferimenti, Titoli, ecc.

Con *Akoma Ntoso* è possibile gestire una serie di metadati in grado di descrivere ed identificare ogni versione di un testo normativo, permettendo all'utente di visionarne le precedenti e comprendere così al meglio l'evoluzione nel tempo di un determinato documento.
Tutto ciò è reso possibile dagli URI, ovvero identificatori univoci che permettono il collegamento tra documenti che possono essere interni o esterni alla nazione.[VZ07] [VP24]

I metadati non sono altro che informazioni aggiuntive che meglio descrivono, definiscono e classificano il contenuto dei documenti legali, generati automaticamente da un'analisi del testo o definite manualmente da un intervento umano.[BCP+10]

L'utilizzo di *Akoma Ntoso* come standard permette agli sviluppatori di sfruttare le sue caratteristiche particolari per la gestione in modo efficiente delle operazioni eseguibili dall'utente in fase di editing del documento. La gestione degli eventi utente è essenziale per garantire un'esperienza fluida e intuitiva a tutti gli utenti che hanno il bisogno di lavorare con documenti legislativi di qualsiasi complessità.

Uno degli aspetti principali della gestione degli eventi utenti in questo contesto è la necessità di avere delle operazioni controllate, che sono in grado di garantire l'integrità strutturale del documento in seguito alle modifiche effettuate.

Di seguito un esempio di un documento scritto nel formato *Akoma Ntoso*:

Figura 2.2: Contenuto XML di un documento Akoma Ntoso

Capitolo 3

Architettura dell'editor di documenti legali Simplex

3.1 Struttura generale dell'editor

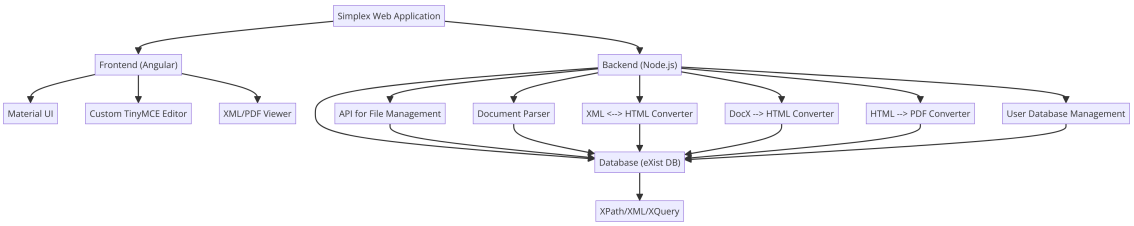


Figura 3.1: Architettura di SIMPLEX

Nella figura soprastante, tratta dalla tesi magistrale di Andrea D'Arpa[D'A24], possiamo osservare l'architettura generale di SIMPLEX, che consiste in un'applicazione web utile alla creazione e alla modifica di testi legislativi, giudiziari e parlamentari basati sullo standard *Akoma Ntoso*. La sua gestione è affidata, come ogni applicazione web, ad una parte di *Frontend* ed a una di *Backend* e le tecnologie utilizzate per la realizzazione di queste sono rispettivamente Angular e NodeJs.

Si parte dall'analisi delle tecnologie e delle responsabilità della parte *Frontend* dove viene utilizzato Angular, ovvero il framework JavaScript open source di Google. La sua particolarità consiste nel riuscire a fornire strumenti per realizzare applicazioni web in modo semplice e veloce, eseguibili su qualsiasi piattaforma. Questo framework è predisposto per lo sviluppo di software adeguatamente suddivisi in blocchi autonomi detti componenti. Ogni componente rappresenta, in primo luogo, una sezione dell'interfaccia definita da un proprio template grafico e una logica sottostante. Questi blocchi possono poi essere composti per realizzare l'intero applicativo. In seconda analisi, i componenti possono però anche essere unicamente logici, fornendo un ulteriore livello di astrazione del codice e una separazione che, unita alla progettazione a componenti, facilita la manutenzione, la gestione e l'implementazione di nuove funzionalità.

L'interfaccia utente dell'applicazione è realizzata utilizzando la libreria Material UI che fornisce un'implementazione di componenti Angular che seguono le indicazioni di UX(User eXperience) fornite da Google, pensate per avere una parte grafica che si comporti imitando oggetti reali, con la possibilità di adattarsi dinamicamente alla situazione con effetti grafici di uguale ispirazione, morbidi, piacevoli e soprattutto intuitivi.

Il principale aspetto gestito dal frontend è l'integrazione della libreria *tinymce* per la realizzazione dell'editor avanzato in grado di modificare e la visualizzare la struttura Akoma Ntoso di un documento. *Tinymce* è un editor di documenti HTML WYSIWYG(What You See Is What You Get), ovvero che permette la modifica del contenuto senza cambiare direttamente il codice che lo genera, ma utilizzando un'interfaccia in cui è possibile visualizzare l'attuale stato del documento, rendendo così le operazioni di modifica paragonabili a quelle di un editor di testo classico: questo perché gestisce automaticamente i cambiamenti alla struttura del codice HTML che

lo genera. Questa sua caratteristica aggiunta alla sua natura altamente personalizzabile, lo rende l'editor ideale per lo sviluppo di SIMPLEX, in quanto ci permette di fornire un'interfaccia semplice ed intuitiva per la modifica di documenti dell'ambito legislativo che fanno uso dello standard *Akoma Ntoso* per la loro rappresentazione, ma anche di integrare in modo semplice tutte le funzionalità atte alla manipolazione della struttura associata al documento. La sua funzione all'interno dell'applicazione è la gestione della rappresentazione e delle modifiche effettuate al documento nel formato HTML, il quale viene poi utilizzato per la generazione del documento *Akoma Ntoso* equivalente. Per questo motivo ogni operazione di inserimento o modifica degli elementi all'interno della struttura ha bisogno di una serie di regole in grado di determinare quali sono le operazioni valide e capire se il contenuto HTML attuale segue o meno le indicazioni dello standard utilizzato.

Si analizzano ora le tecnologie e le responsabilità della parte *Backend* dove viene utilizzato NodeJs, un ambiente di runtime che permette l'esecuzione di codice JavaScript lato server, pensato principalmente per la realizzazione di applicazioni web asincrone e basate su eventi. Il suo utilizzo permette di avere un'applicazione scalabile ed efficiente, in quanto ogni operazione viene gestita in modo non bloccante. Grazie alla grande community di professionisti che hanno scelto di adottare questa tecnologia, è possibile fare affidamento su una ricca documentazione ed ottenere facilmente supporto e accesso a svariati moduli e librerie messe a disposizione tramite NPM (Node Package Manager), un gestore di pacchetti che permette la loro integrazione in modo semplice e controllato. NPM inoltre ha la caratteristica di poter gestire autonomamente le dipendenze dei vari pacchetti installati.

Il *Backend* si occupa di gestire l'interazione dell'applicazione con il database contenente i documenti in formato XML, memorizzati utilizzando una nomenclatura in grado di distinguere e identificare le principali caratteristiche di ogni documen-

to. Vengono anche esposte delle API, in grado di fornire le principali operazioni atte alla gestione e consultazione dei file, occupandosi inoltre di tutte le operazioni di comunicazione con il database. Queste API permettono anche di eseguire tutte le operazioni necessarie all'autenticazione e gestione degli utenti. L'altra mansione principale affidata al *Backend* è la conversione del contenuto del documento mediante l'utilizzo di appositi parser, che permettono di ottenerne diversi formati per i vari scopi dell'applicazione:

- una in HTML utile alla visualizzazione e manipolazione da parte del frontend;
- una in XML utile per la visualizzazione del documento con i tag dello standard *Akoma Ntoso*;
- una in PDF utile alla esportazione e alla condivisione del documento.

Si notano diverse similitudini tra SIMPLEX e LIME, in particolar modo nell'interfaccia utente. La scelta di mantenere le due applicazioni graficamente simili è derivata dal non voler creare situazioni di confusione in seguito alla sostituzione del precedente applicativo e facilitare l'utilizzo della stessa da parte degli utenti abituati alla precedente configurazione. Per questo motivo anche le funzionalità offerte da SIMPLEX sono principalmente basate su quelle offerte da LIME, con l'aggiunta di una serie di novità in grado di agevolare e guidare l'utente nelle operazioni di marcatura e modifica del documento. Per quanto riguarda le differenti tra i due applicativi, invece, troviamo in SIMPLEX una differente rappresentazione della struttura di marcatura, che rispetto alla precedente va a posizionare il contenuto marcato all'interno di un box colorato, identificato da un'etichetta che ne rappresenta la tipologia. Questa soluzione oltre a cercare di rendere la visualizzazione del documento marcato più simile possibile a quella del documento finale, permette all'utente di impostare la modalità di visualizzazione delle etichette, rendendo possibile visualizzarle o meno la struttura di marcatura: in questo modo si è facilmente

in grado di ottenere la visualizzazione di stampa del documento senza ricorrere all'utilizzo di conversioni. Non si hanno soltanto differenze strettamente legate alla parte grafica, ma anche a quella logica, infatti SIMPLEX punta ad aggiornare e migliorare le funzionalità attualmente presenti, proponendo soluzioni per facilitare le operazioni di manipolazione della struttura di marcatura. Tra queste troviamo ad esempio la possibilità di poter cambiare il tipo di elemento senza ricorrere alla rimozione dell'attuale marcatura per poi successivamente effettuare un'operazione di marcatura con l'elemento desiderato oppure la possibilità di cambiare il livello di gerarchia di un elemento sfruttando le operazioni di *Upgrade* e *Downgrade* che verranno viste nel dettaglio nei capitoli successivi. Queste novità permettono agli utenti di risparmiare tempo durante le operazioni di marcatura.

3.2 Componenti principali e interazioni

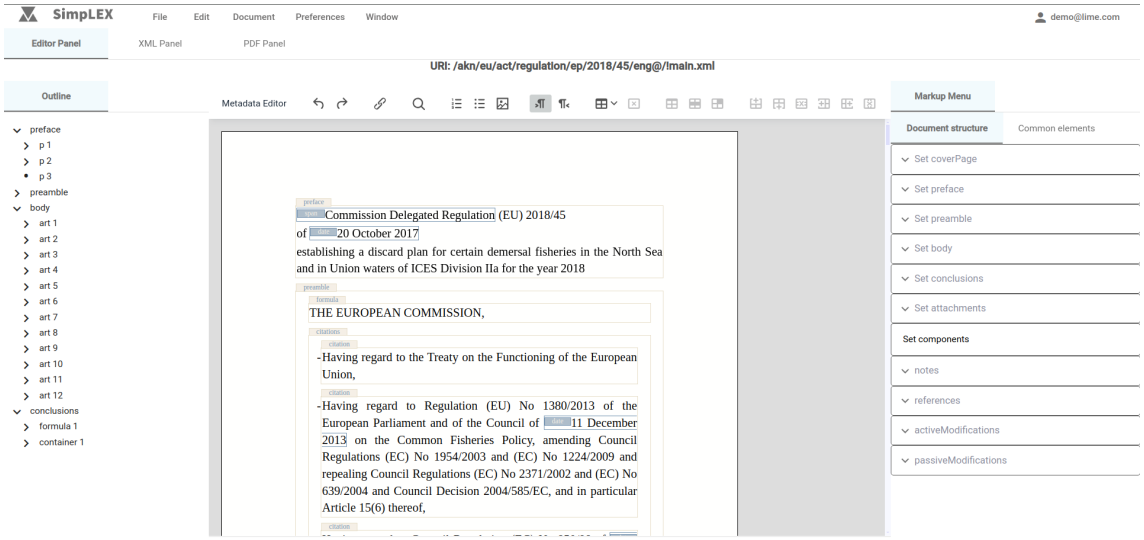


Figura 3.2: Schermata principale dell'applicazione SIMPLEX

L'applicazione come descritto in precedenza è formata da componenti Angular, ognuno dei quali responsabile della gestione di una specifica funzionalità dell'editor.

Questi componenti collaborano con una serie di Services[1] Angular e funzioni TypeScript, che contribuiscono al corretto funzionamento delle operazioni. Il cuore dell'applicazione è rappresentato dal "tinymce-integrator", un componente che ha il compito di supervisionare e gestire il comportamento dell'editor. Esso agisce infatti come un middleware, fungendo da ponte di collegamento tra la parte grafica e i vari servizi dell'applicazione. Possiamo paragonare il tinymce-integrator ad un controller del paradigma di programmazione Model-View-Controller (MVC): gestendo le azioni effettuate dall'utente, effettua operazioni raggruppando correttamente i dati necessari (Model), comunicandoli ad adeguati servizi per poi aggiornare l'interfaccia utente (View). Come si osserva nella Figura 3.2, l'applicazione presenta al centro della pagina l'editor tinymce che mostra il contenuto del documento con l'attuale rappresentazione della struttura di marcatura. Intorno all'editor si trovano i componenti principali dell'applicazione, che mettono a disposizione dell'utente strumenti di visualizzazione e modifica della struttura del documento. Di seguito è riportata una panoramica dei principali componenti dell'applicazione e le loro relative azioni, che sono a disposizione dell'utente per l'utilizzo dell'editor e della manipolazione della struttura del documento.

[1]Un services è un termine generale che si può riferire ad un qualsiasi valore, metodo o funzionalità di cui l'applicazione ha necessità: tipicamente si tratta di una classe con uno scopo contenuto e ben definito.`https://angular.io/guide/architecture-services`

3.2.1 Outline

L'outline permette all'utente di visualizzare la struttura del documento aperto, sotto forma di albero di elementi *Akoma Ntoso*. Questa rappresentazione mostra il nome di ciascun elemento insieme ai suoi discendenti nella gerarchia. L'outline non fornisce soltanto una panoramica della struttura del documento, ma agisce anche come uno strumento di navigazione rapida al suo interno. Ogni voce presente funge infatti da link ipertestuale con il contenuto dell'elemento, che gli utenti possono sfruttare per navigare rapidamente le varie parti della struttura del documento.

Figura 3.3: Outline di un documento

3.2.2 Markup menù

Il menù di markup mette a disposizione dell'utente un set di pulsanti progettati per il processo di marcatura del documento utilizzando gli elementi presenti nello schema di *Akoma Ntoso*. Questi pulsanti sono organizzati in una struttura gerarchica che fornisce una visualizzazione a matrioska, dove ogni livello contiene al suo interno la lista degli elementi validi come figli di quello precedente. Il menù in esame è formato da due sotto categorie: quella contenente tutti gli elementi disponibili per la giurisdizione e la tipologia del documento in uso; e quella degli elementi generici utilizzabili in qualsiasi situazione, soprattutto nel caso in cui non si abbiano ancora a disposizione elementi specifici che soddisfino le richieste.

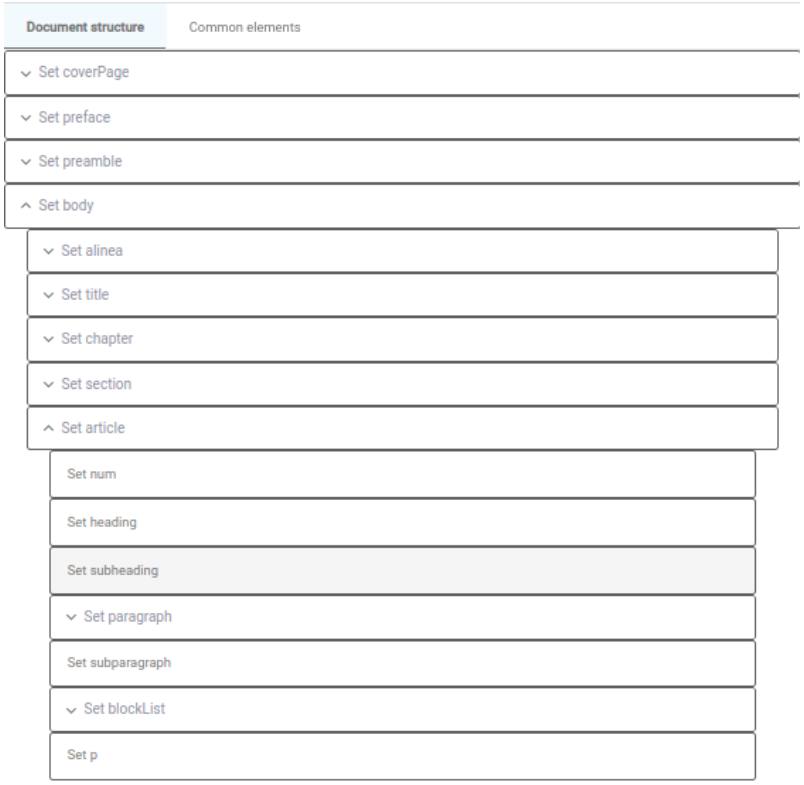


Figura 3.4: Menù di markup

3.2.3 Metadata editor

Il menù dei metadati permette agli utenti di gestire le informazioni associate al documento. Utilizzando questo menù l'utente è in grado di effettuare diverse operazioni che gli permettono di monitorare lo stato attuale dei metadati presenti. Per esempio l'utente può ottenere una panoramica dei metadati collegati al documento divisi in diverse categorie, come la lista dei riferimenti presenti nel testo, le informazioni utili all'identificazione del documento stesso, le modifiche, ecc. che permettono all'utente di modificare o aggiungere informazioni collegate al documento. La presenza di queste informazioni non permette soltanto di identificare o integrare parti significative, ma anche di tenere traccia dei cambiamenti temporali o di stato, cioè tutti quei cambiamenti che riguardano le modifiche effettuate al contenuto testuale o alla relativa struttura, e quelli relativi allo stato delle procedure amministrative.

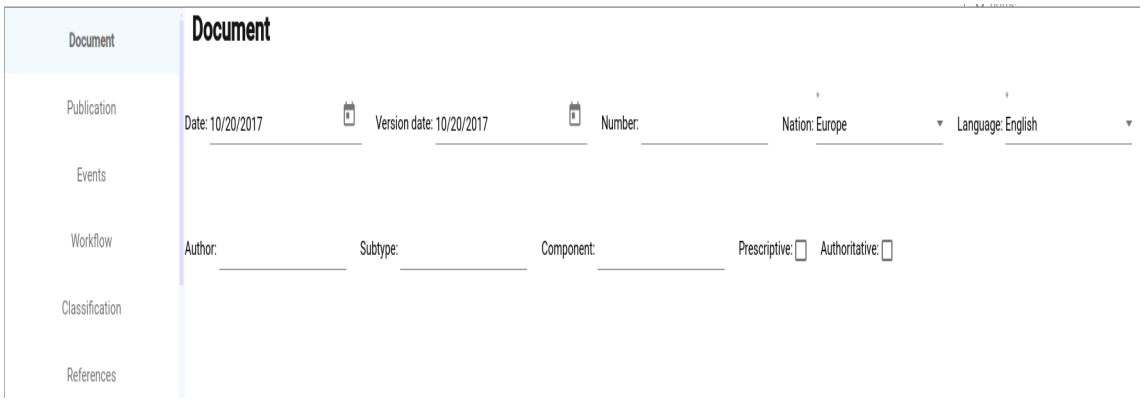


Figura 3.5: Editor dei metadati

3.2.4 Context menù

Il menù contestuale svolge un ruolo di grande importanza nel facilitare le operazioni di modifica della struttura del documento. Questo menù offre all'utente una serie di comode operazioni per la modifica o l'inserimento di nuovi elementi strutturali.

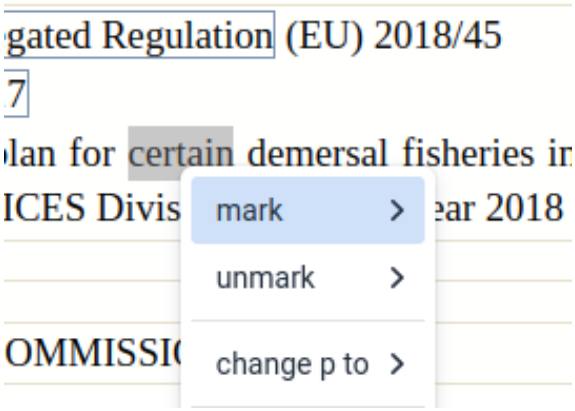


Figura 3.6: Menù contestuale

Si descrivono ora le operazioni attualmente gestite all'interno del menù contestuale.

Mark (Marcatura). Fornisce una comoda scorciatoia per l'applicazione della marcatura. L'utente può selezionare il contenuto che desidera marcare ed in seguito, cliccando il tasto destro del mouse, il menù contestuale mostrerà una lista di possibili elementi validi all'inserimento all'interno del componente in cui la selezione si trova. Questa funzionalità semplifica notevolmente il processo di marcatura di bassa complessità che può quindi essere eseguita in poche semplici operazioni;

Unmark (Annulla Marcatura). Essenziale per rimuovere la marcatura da un elemento. Nonostante l'applicazione di regole puntigliose nella marcatura gli errori possono comunque verificarsi, e l'operazione di "Unmark" offre una soluzione pratica per correggere eventuali inserimenti impropri di etichette;

Change (Cambia). Consente all'utente di modificare il tipo di struttura di un elemento, sostituendolo con un altro elemento che abbia lo stesso pattern e che sia valido all'interno del componente in cui mi trovo. Questo processo è particolarmente utile per adattare la struttura del documento senza compromettere la validità. L'operazione di "Change" offre quindi una soluzione pratica per risolvere tutti quei casi in cui la struttura inserita non sia quella desiderata, permettendo di effettuare la modifica senza dover ricorrere all'annullamento della marcatura ed ad una successiva marcatura con l'elemento corretto.

Capitolo 4

Gestione degli eventi utente

4.1 Definizione di eventi utente nel contesto dell'editor di documenti legali

Analizziamo gli eventi utente nel contesto di editor di documenti dell'ambito legislativo basati sullo standard *Akoma Ntoso*.
Gli eventi che devono essere gestiti in modo accurato comprendono una serie di azioni ed operazioni che gli utenti possono eseguire durante la manipolazione del contenuto o della struttura del documento. La gestione di tali eventi in modo accurato ed efficace è fondamentale, in quanto l'obiettivo di un'ottima gestione è quello di ridurre il più possibile la probabilità di generare documenti non conformi con le specifiche dello standard in esame.

Questi eventi possono essere suddivisi in due categorie principali: operazioni di base ed operazioni strettamente legate a SIMPLEX, ovvero tutte le operazioni caratteristiche e specifiche per l'editor in questione che comunemente non troviamo in editor di testo tradizionali.

Eccone una lista:

Operazioni di base:

- Creazione di un documento
- Apertura di un documento esistente
- Salvataggio di un documento
- Esportazione di un documento

Operazioni legate a SIMPLEX:

- Inserimento di elementi strutturali
- Modifica di elementi strutturali
- Eliminazione di elementi strutturali
- Filtraggio dell'input da tastiera
- Azioni al click di tasti speciali (Enter, Tab, Backspace, ecc..)
- Navigazione del documento tramite outline
- Aggiornamento dell'outline
- Numerazione automatica degli elementi
- Controlli sulla validità della struttura del documento
- Operazioni di Upgrade e Downgrade

4.2 Analisi dei tipi di eventi e delle loro implicazioni

Analizziamo ora il comportamento delle operazioni precedentemente elencate.

Creazione di un documento

La creazione di un documento richiede all'utente, tramite la visualizzazione di un dialog apposito, l'inserimento del tipo di documento, la giurisdizione da utilizzare e la lingua in cui esso verrà scritto. Una volta ottenuti i dati l'editor si occuperà di generare la struttura dei metadati associati e di visualizzare a schermo la struttura base del documento fornita da un apposito template generato tramite le regole fornite.

Apertura di un documento

L'apertura di un documento può avvenire in due modalità differenti.
Per i documenti già nel formato *Akoma Ntoso*, l'editor effettuerà la lettura dei metadati presenti in esso e, mediante l'utilizzo di appositi convertitori, il contenuto del documento verrà convertito da XML a HTML per rendere possibile la visualizzazione e la manipolazione dall'editor *tinymce*.

Per tutti gli altri formati sarà necessario effettuare una prima conversione in XML del documento, generando la struttura base dei metadati. Si procede poi con i passaggi accennati poco fa per aprire il documento.

In entrambi i casi, i metadati permettono a SIMPLEX di filtrare dai file di configurazione le informazioni essenziali per la generazione del menù di marcatura ideale per il documento corrente.

Inserimento di elementi strutturali

L'inserimento degli elementi strutturali utili alla marcatura del documento, come accennato in precedenza, viene eseguito dal menù di marcatura oppure dall'operazione *Mark* del menù contestuale. Quello che avviene in seguito alla richiesta di inserimento di un elemento, come mostrato anche nella soluzione proposta da Marek Patryk Dziedzic, è la seguente: l'editor ottiene il nome e il pattern dell'elemento strutturale da inserire, viene lanciato l'evento di *AddElement* e il *tinymce-integrator* affidandosi al componente *AknRuleChecker*, determina la validità dell'operazione e la relativa azione da effettuare per una corretta gestione[Dzi22].

Modifica di elementi strutturali

La modifica degli elementi strutturali generalmente è gestita dall'operazione di *Change*, che si occuperà di effettuare il cambio di elemento associato senza alterarne il contenuto.

Input da tastiera

L'editor necessità di filtrare l'input da tastiera permettendo l'utilizzo di una serie di regole configurabili, che vincolano le operazioni valide effettuabili da un utente in base al contesto in cui si trova il cursore di inserimento.

Tasti speciali

I tasti speciali come ad esempio l'Enter, Shift-Enter e Tab svolgono ruoli specifici nella navigazione e manipolazione della struttura del documento, consentendo operazioni di split del contenuto, navigazione e creazione di nuovi elementi.

Navigazione del documento tramite outline

Come visto in precedenza, l'outline fornisce una rappresentazione strutturata del documento sotto forma di albero gerarchico, agevolando la navigazione e permet-

tendo la visualizzazione del contenuto degli elementi selezionati attraverso l'utilizzo di ID univoci.

Aggiornamento dell'outline

Ogni modifica strutturale al documento scatena un evento di aggiornamento dell'outline, mantenendolo sincronizzato con la struttura attuale del documento.

Numerazione automatica

La numerazione automatica degli elementi, dove necessario, avviene in modo coerente in seguito all'inserimento di nuovi elementi, consentendo di mantenere un ordinamento corretto.

Controlli sulla validità della struttura del documento

Ogni qualvolta che viene effettuata un'operazione che va a modificare la struttura del documento, come ad esempio un inserimento di un nuovo elemento strutturale, bisogna fare in modo che la validità della struttura venga rispettata. Possiamo quindi utilizzare le regole definite nei file di configurazione per controllare che la gerarchia sotto analisi sia coerente.

Operazioni di Upgrade e Downgrade di un elemento

Le operazioni di *Upgrade* e *Downgrade* nascono con l'utilità di fornire supporto al drag and drop dell'outline, che permette di trascinare con il puntatore gli elementi cambiando la loro posizione o gerarchia all'interno della stessa. Il funzionamento delle operazioni di *Upgrade* e *Downgrade* fornisce un'astrazione atta alla gestione dell'evento sopracitato.

4.3 Metodologie per la gestione degli eventi utente

Gli eventi sopra elencati vengono intercettati dall'applicazione utilizzando due diversi metodi:

- Le operazioni comuni vengono gestite da *tinymce*, come ad esempio l'inserimento da tastiera, o il click del mouse, che sono catturabili mediante l'utilizzo degli event listener di JavaScript. Questa gestione permette inoltre di applicare funzionamenti personalizzati alle operazioni di base;
- Per tutte le operazioni strettamente legate a SIMPLEX invece abbiamo bisogno di definire degli eventi personalizzati che verranno chiamati ed intercettati da *tinymce* nel momento che l'applicazione ne fa richiesta.

Le operazioni che vanno ad apportare modifiche alla struttura o al contenuto del documento necessitano un'approvazione da parte delle regole definite dall'applicazione e dallo standard. Ogni operazione infatti può incorrere in una situazione di approvazione o di rifiuto. Nel secondo caso abbiamo bisogno di un metodo di segnalazione degli errori per l'utente, indicando che l'operazione effettuata è stata rifiutata e specificando il motivo dell'annullamento nel dettaglio, ove possibile.
Questo meccanismo di prevenzione sulle operazioni effettuabili garantisce la correttezza della struttura del documento *Akoma Ntoso* per le operazioni di manipolazione del contenuto. Il controllo della validità della struttura del documento, invece, viene eseguito successivamente alla risoluzione dell'operazione, rendendo la gestione del controllo più semplice e l'utilizzo dell'applicazione più libero, mantenendo comunque la filosofia di notificare l'utente nel caso di situazioni invalide.

Capitolo 5

Gestione degli eventi in Simplex

Analizziamo ora le scelte implementative effettuate per la gestione ed il miglioramento dei principali eventi per l'utilizzo dell'applicazione.

5.1 Rivisitazione del markup menù

Utilizzando la struttura dei file di configurazione ideata da Luca Cervone per la creazione del menù di markup in LIME[Cer13], siamo in grado di configurare e generare menù adeguati per ogni tipologia di documento e giurisdizione. La semplicità dello schema proposto permette agli enti di comprendere ed apportare modifiche al contenuto in modo semplice ed intuitivo. La soluzione precedentemente adottata per la creazione del menù di marcatura prevedeva la presenza di un singolo file di configurazione, contenente i dettagli di tutti i possibili pulsanti raggruppati in base al pattern dell'elemento associato. Essi venivano attivati o disattivati in base all'esito di un controllo sul posizionamento del cursore in quel momento.
Questa gestione richiede l'utilizzo di una struttura JSON complessa, non facilmen-

te personalizzabile ed adattabile alle varie tipologie di documento e alle diverse giurisdizioni trattate.

Osserviamo di seguito una panoramica dei cambiamenti e dei benefici introdotti dalla proposta sopra citata per la creazione e la gestione del menù di marcatura.

Di seguito viene proposto un esempio introduttivo sulla creazione del menù, partendo dal file di configurazione che segue lo schema proposto da Luca Cervone.

```
{
    "rootElements": ["preface", "preamble", "body", "conclusions", "attachments", "components", "notes", "references"],
    "elements": {
        "preface": {
            "children": ["p", "toc", "docType", "docNumber", "docDate", "docTitle", "docAuthority", "docketNumber"]
        },
        "preamble": {
            "children": ["p", "toc", "formula", "recitals", "citations", "list"]
        },
        "body": {
            "children": ["alinea", "title", "chapter", "section", "article", "clause", "subclause", "list", "point", "paragraph", "mod"]
        }
    }
}
```

Figura 5.1: Esempio di configurazione menù di markup

Il file di configurazione serve a definire la struttura del menù desiderata, specificando gli elementi presenti ed eventuali personalizzazioni dei vari sottomenù che in linea generale sono ottenuti leggendo la configurazione di default.

La struttura del menù viene generata in seguito alla creazione o all'apertura di un documento, dal quale ricaviamo tipo e giurisdizione attraverso la lettura dei metadati, se presenti. In alternativa verranno richiesti tramite un apposito menù all'utente. Queste informazioni vengono utilizzate dall'applicazione per ottenere il file di configurazione adatto alla coppia di valori ricevuti, che procederà ad effettuare una conversione del contenuto fornito in un oggetto JavaScript utilizzato da Angular per la visualizzazione a schermo. Il risultato di questa operazione è un insieme di elementi definiti dalla struttura visualizzabile nella figura sottostante.

```
interface verticalButtonBar {
  type: "verticalButtonBar";
  label: string;
  pattern: AknPattern;
  elementName: string;
  action: string;
  items: (button | verticalButtonBar)[];
}
```

```
interface button {
  type: "button";
  label: string;
  pattern: AknPattern;
  elementName: string;
  action: string | Function;
}
```

Figura 5.2: Interfacce dei pulsanti

Abbiamo a disposizione i seguenti due tipi di pulsanti:

- *verticalButtonBar*, che rappresentano generalmente elementi che svolgono la funzione di contenitori di altri elementi all'interno della struttura del documento;
- *button*, i quali vengono solitamente utilizzati per l'inserimento di elementi validi come figli dei contenitori presenti.

Questa possibile rappresentazione permette all'applicazione di gestire adeguatamente l'evento di click eseguendo l'azione associata al pulsante.

Nella maggior parte dei casi l'operazione assegnata ad ogni pulsante coincide con l'evento di *AddElement* a cui passiamo come argomenti il nome e il pattern associati.

5.2 Rinnovamento dell'utilizzo e gestione delle regole

I file di configurazione delle regole sono stati ampliati ed, inoltre, è stata rivisitata la struttura delle regole per permettere una rappresentazione più leggibile e chiara. Le regole contenute nei file di configurazione vengono utilizzate dall'applicazione

durante la fase di verifica della validità delle operazioni e nella gestione della correttezza della gerarchia del documento. Come riportato nella tesi di Marek Patryk Dziedzic, abbiamo a disposizione quattro file di regole pensati per la gestione delle operazioni dell'editor.[Dzi22]
Si elenca e si descrive ora l'utilizzo delle sopracitate tipologie di regole.

Regole di bella forma

Utilizzate per controllare che la selezione effettuata per un operazione di marcatura o di rimpiazzamento del contenuto testuale abbia inizio e fine all'interno dello stesso elemento. Lo scopo di queste regole è di impedire la creazione di elementi sprovvisti del loro tag di apertura e/o chiusura, o in generale di impedire annidamenti non validi. Un possibile esempio di una selezione non valida è la seguente:

Il <date>6 agosto 1991</date>nasce il World Wide Web

La porzione di testo evidenziata rappresenta una possibile selezione che per i motivi elencati poc'anzi, risulta essere non valida in quanto essa ha inizio all'interno del tag p e termine all'interno di date.

Regole dei pattern

Utilizzate per effettuare controlli relativi al pattern associato agli elementi della struttura del documento. Sfruttando queste regole è possibile determinare se un elemento segue le regole di gerarchia dei pattern fornite dallo standard, bloccando tutte le operazioni di rimpiazzamento che renderebbero invalida la struttura del documento, oppure segnalando eventuali errori della stessa.

Regole di Akoma Ntoso

Queste regole servono a gestire situazioni particolari specifiche di Akoma Ntoso, impostando vincoli su quali elementi può contenere un determinato elemento dal

punto di vista del funzionamento base dello standard. Queste regole sono quelle proprie del linguaggio, e non le personali implementazioni di template custom che invece verranno raggruppate nella prossima categoria di regole.

Regole locali

Queste regole permettono di aggiungere vincoli personalizzati, utili alla gestione di situazioni presenti soltanto in determinate giurisdizioni, dando quindi spazio a tutte le possibili implementazioni personalizzate di Akoma Ntoso che ne caratterizzano elasticità, versatilità, efficienza ed efficacia nei singoli speciali ambiti di utilizzo.

Con la nuova implementazione proposta, i quattro gruppi di regole vengono utilizzati seguendo il seguente ordine: bella forma, pattern, Akoma Ntoso ed infine locali, affidando l'esito dell'operazione alla risposta del gruppo avente maggiore priorità, in questo caso quello locale. Può infatti succedere che in certi casi le regole di livello superiore rifiutino l'operazione, mentre i livelli inferiori forniscono una possibile azione da eseguire per la sua esecuzione. Questo è un ottimo indicatore della elasticità di Akoma Ntoso: i documenti personalizzati, caratterizzati da regole speciali proprie della singola legislazione e decise in contesti e circostanze del tutto ignare allo standard, possono essere implementate anche in contraddizione con le regole di base della struttura del formato (solo se l'implementazione della regola offre un modo di risolvere il problema che, per conto suo, il programma non saprebbe come gestire e quindi rifiuterebbe se non si trattasse di una regola personalizzata).

Di seguito viene riportato un esempio dello schema utilizzato per riscrivere le regole dell'editor.

Figura 5.3: Esempio di regola di sostituzione

Osservando l'esempio possiamo notare come la regola sia composta dai seguenti attributi:

- **id**: utile per identificare la regola e renderla facilmente localizzabile;
- **selection:** indica il tipo di selezione effettuata dall'utente;
- **replacement:** indica il tipo di elemento da inserire o sostituire nel contenuto della selezione;
- **action**: indica l'azione che deve essere eseguita per il completamento dell'operazione richiesta;
- **_doc:** permette di inserire una descrizione della regola.

Questa nuova rappresentazione permette di avere una maggiore chiarezza sugli elementi coinvolti e sull'effettivo comportamento della regola, semplificando la fase di debug durante lo sviluppo e la fase di testing dell'utilizzo.

Ogni file inoltre può contenere al proprio interno un ulteriore set di regole utili alla gestione della correttezza della struttura del documento dove, indicando l'elemento interessato, possiamo definire i valori di pattern o name validi come figli.

Ad esempio, nel file di configurazione delle regole di pattern troviamo la seguente definizione:

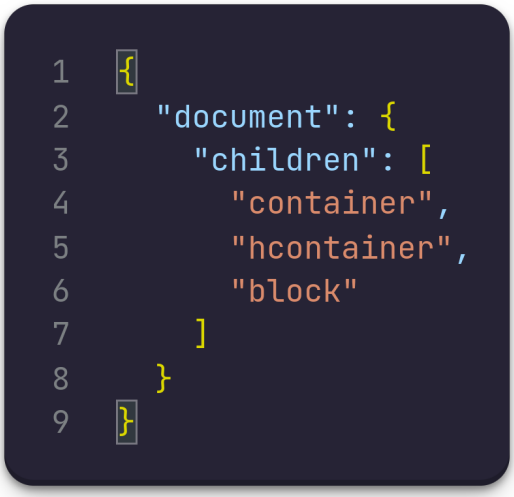


Figura 5.4: Esempio di regola di gerarchia dei pattern

La struttura visualizzata in Figura 5.4 imposta dei vincoli sui pattern validi ad essere inseriti come figli di *document*. L'esempio proposto fornisce una possibile configurazione.

Per tenere meglio traccia del funzionamento delle regole è stato introdotto un sistema pensato per semplificare le fasi di debug e di testing da parte degli sviluppatori. Questo strumento nasce dalla necessità di avere un meccanismo in grado di fornire

un report chiaro sull'effettiva applicazione delle regole per le operazioni gestite dall'editor. Per ogni operazione teniamo traccia della relativa selezione e del contenuto da sostituire mostrando la sequenza di esecuzione delle regole adatte al contesto in esame, mostrando infine l'esito delle valutazioni effettuate.

Esamineremo come l'operazione di inserimento testuale venga gestita da SIMPLEX, utilizzando le regole specificate nei differenti file di configurazione.

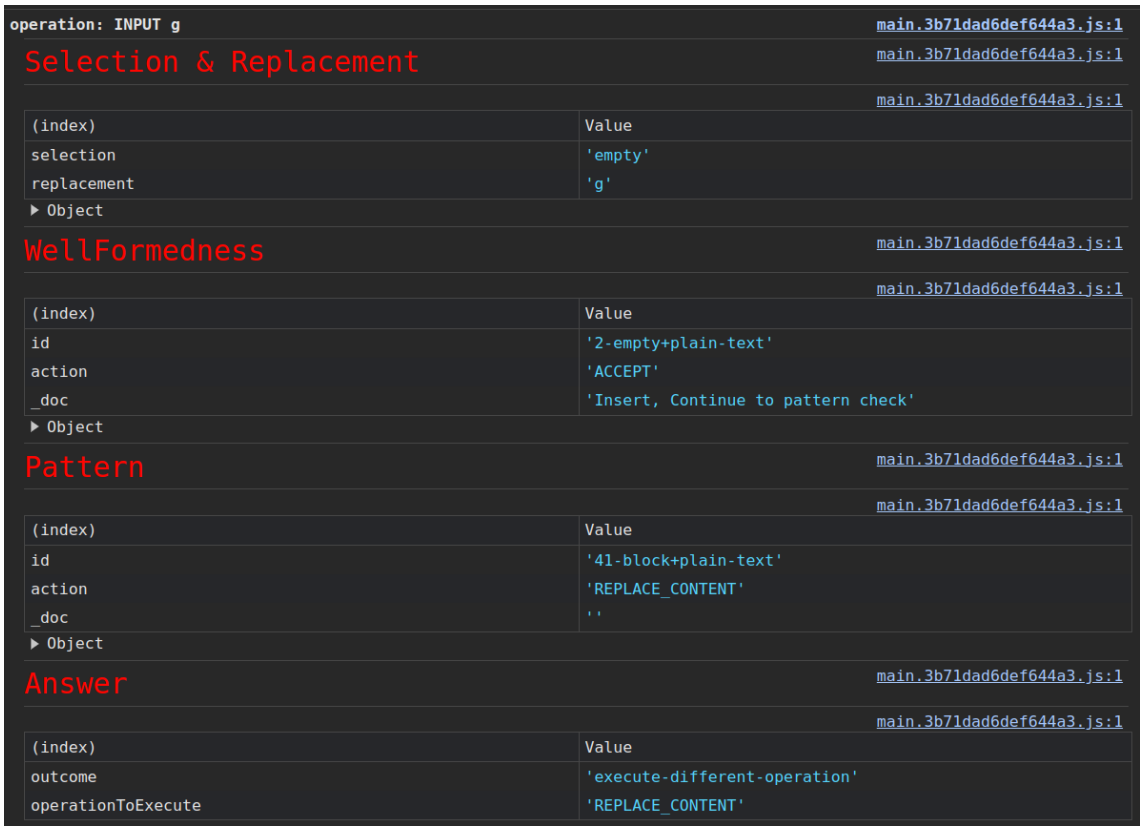


Figura 5.5: Sequenza di utilizzo delle regole

Esaminiamo in particolare la gestione dell'inserimento del carattere "g" all'interno del contenuto di un articolo, focalizzandoci sulla sequenza di risoluzione delle regole.

Bella Forma: Le regole di bella forma verificano che la selezione effettuata dall'utente non sia sbilanciata, ovvero che inizi e finisca all'interno dello stesso elemento.

In questo specifico caso notiamo la presenza di una selezione vuota, che sta ad indicare che l'utente ha posizionato il cursore senza selezionare alcun testo.

Pattern: Le regole di pattern utilizzano il pattern dell'elemento che contiene la selezione effettuata per valutare e determinare l'azione corretta da eseguire. Nel contesto in esame, l'inserimento è eseguito all'interno di un elemento avente pattern block, utilizzato per lo specifico scopo di racchiudere il contenuto testuale degli elementi all'interno della struttura del documento. Per la gestione di questa situazione le regole di pattern indicano che l'operazione consigliata è il rimpiazzamento del contenuto, permettendo quindi il corretto inserimento del carattere "g".

Locali e di Akoma Ntoso: In questo specifico caso non vengono utilizzate regole del tipo "locale" o "Akoma Ntoso" poiché l'esempio trattato gestisce l'inserimento di un carattere, che non influisce sulla struttura del documento, ma soltanto sul suo contenuto testuale.

Questo meccanismo garantisce un controllo robusto sulle operazioni di inserimento o sostituzione di testo o elementi, che rispettano le regole strutturali e semantiche del documento in conformità allo standard Akoma Ntoso. Inoltre, gli sviluppatori ricevono un feedback dettagliato sulla sequenza di approvazione delle azioni e ciò consente loro di verificare che l'editor segua correttamente la sequenza prevista, assicurandosi inoltre che le regole vengano applicate in modo appropriato e fornendo un valido tool di analisi per lo sviluppo di nuove funzionalità e regole speciali. Nel caso in cui un'operazione venga rifiutata, gli utenti ricevono un feedback informativo che cerca di specificare nel dettaglio la motivazione di tale scelta. Questo approccio consente agli utenti di comprendere chiaramente il motivo per cui l'operazione non sia stata accettata, agevolando la correzione e la comprensione delle regole applicate. Fornendo dei feedback dettagliati agli utenti si promuove una migliore comprensione delle regole Akoma Ntoso, garantendo così la coerenza e l'integrità dei documenti

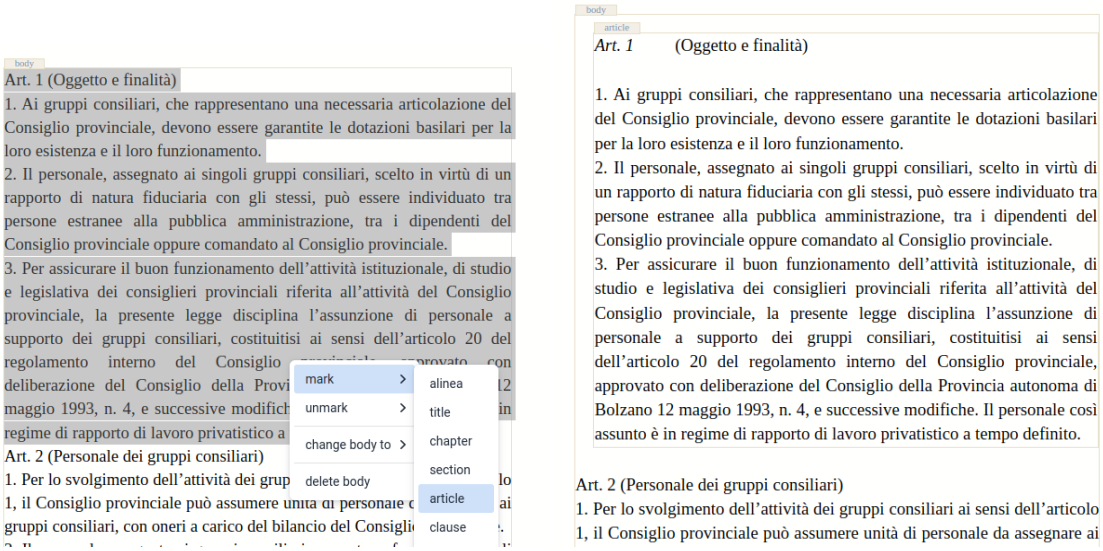
redatti. Gli utenti sono in un certo senso guidati nella creazione di documenti che rispettano le regole dello standard, contribuendo quindi a mantenere un elevato livello di qualità e conformità nei processi di editing.

5.3 Operazioni di markup

Come mostrato in precedenza le operazioni di marcatura messe a disposizione all'utente dal menù di markup e dall'operazione di Mark presente nel menù contestuale servono ad inserire elementi strutturali all'interno del documento. Le operazioni di marcatura variano di complessità di gestione in base al pattern dell'elemento da inserire e in base alla struttura e al contesto nel quale lo si vuole aggiungere. Abbiamo quindi la necessità di effettuare operazioni differenti per l'inserimento di un elemento avente pattern *inline* rispetto a quelli con pattern container o *hcontainer*. Nel caso degli elementi semplici come gli inline è sufficiente effettuare un'operazione di wrap del testo selezionato, racchiudendolo all'interno dell'elemento richiesto. Invece, nel caso in cui l'inserimento riguardi elementi con pattern container o *hcontainer* abbiamo bisogno di effettuare controlli sull'attuale struttura in cui l'elemento deve essere aggiunto, facendo in modo che il risultato di tale inserimento rispetti i requisiti dello standard Akoma Ntoso.

Abbiamo inoltre necessità di aggiungere funzionalità particolari a determinati elementi, in particolar modo a tutti quelli che mantengono un collegamento diretto con la struttura dei metadati come ad esempio i riferimenti e le mod. Questi particolari elementi vengono inseriti all'interno della struttura come tutti gli altri, ma ad essi vengono associati dei comportamenti aggiuntivi. I riferimenti vengono affiancati da un evento di click che permette all'utente di specificare le informazioni utili alla generazione del metadato associato mediante l'utilizzo di una finestra apposita. Invece per quanto riguarda le mod necessitiamo della presenza di un servizio che si occupi

di notificare il gestore dei metadati le informazioni relative alla modifica appena inserita, permettendo di aggiornare automaticamente i metadati associati ad essa. Visualizziamo di seguito un esempio di inserimento di un elemento di marcatura utilizzando l'operazione di mark presente nel menù contestuale.



(a) Selezione effettuata e marcatura scelta

(b) Risultato dell'operazione di marcatura

Figura 5.6: Esempio di marcatura del testo

5.4 Feedback utente

La gestione ed il controllo degli errori permette di avere un sistema pratico e personalizzabile che fornisce all'utente informazioni dettagliate che motivano l'annullamento di un'operazione richiesta o la presenza di incompatibilità all'interno della struttura del documento. Per controllare che la struttura del documento sia valida, viene analizzata quest'ultima partendo dalla radice e verificando che l'elemento corrente rispetti le regole di pattern e di Akoma Ntoso impostate nelle configurazioni. Nel caso in cui si verifichi un'incompatibilità, l'editor evidenzia in rosso il contenuto te-

stuale dell'elemento non valido, mostrando inoltre un popup informativo contenente il motivo per il quale l'elemento viene contrassegnato come tale.

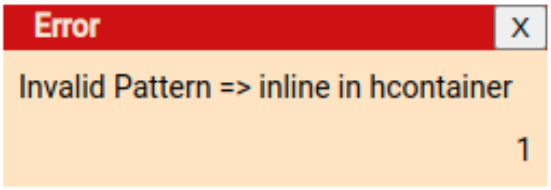


Figura 5.7: Esempio di messaggio di errore

Nella Figura 5.7 osserviamo un esempio di popup di errore. Nel caso in esame viene segnalata la presenza di un pattern non valido, cioè la presenza di un inline all'interno di un elemento hcontainer non permesso dalle regole di pattern dello standard.

5.5 Undo e redo personalizzati

Per annullare o ripristinare le operazioni di modifica della struttura del documento è necessaria una gestione personalizzata degli eventi di *undo* e *redo*. Sarà quindi importante adottare un comportamento personalizzato capace di annullare o ripetere le operazioni di marcatura effettuate, in quanto il funzionamento gestito dall'editor *tinymce* da solo non è in grado di tenere traccia di queste modifiche. Per ovviare a questo limite si definiscono dei comportamenti adatti alla gestione di queste ultime, che permettono all'utente di annullare oppure ripristinare le operazioni effettuate. La proposta implementata prevede di utilizzare una lista che mantiene in memoria lo stato attuale del documento e che viene aggiornata in seguito all'esecuzione di operazioni di inserimento o modifica di elementi. L'applicazione, tramite l'utilizzo di un indice, tiene traccia della struttura corrente del documento, incrementando o diminuendo l'indice stesso per ottenere e visualizzare la situazione precedente o successiva del contenuto in seguito ad una richiesta di un *undo* o *redo*, rispettivamente.

5.6 Upgrade e Downgrade di un elemento

Le operazioni di Upgrade e Downgrade forniscono all'applicazione metodi per la gestione della struttura del documento. Quello che introducono, infatti, è la possibilità di far salire o scendere di gerarchia elementi aventi pattern *hcontainer*. L'operazione di Upgrade permette ad un elemento di diventare fratello di quello che prima era suo padre, salendo quindi nella gerarchia e diventando dello stesso tipo del padre. L'operazione di Downgrade, invece, permette ad un elemento di scendere di gerarchia diventando figlio del precedente fratello. Queste operazioni saranno possibili solo sotto specifiche condizioni: per l'Upgrade bisogna verificare che l'elemento selezionato ed il relativo padre abbiano lo stesso pattern, mentre il Downgrade sarà possibile solo dal secondo fratello in poi e quando la struttura del fratello precedente ne permette l'inserimento.

Analizziamo ora un esempio dell'operazione di Upgrade visualizzabile nelle figure sottostanti, dove il paragrafo tre, inizialmente figlio di articolo uno, viene dapprima cambiato in articolo ed infine riposizionato nella gerarchia come fratello di articolo uno.

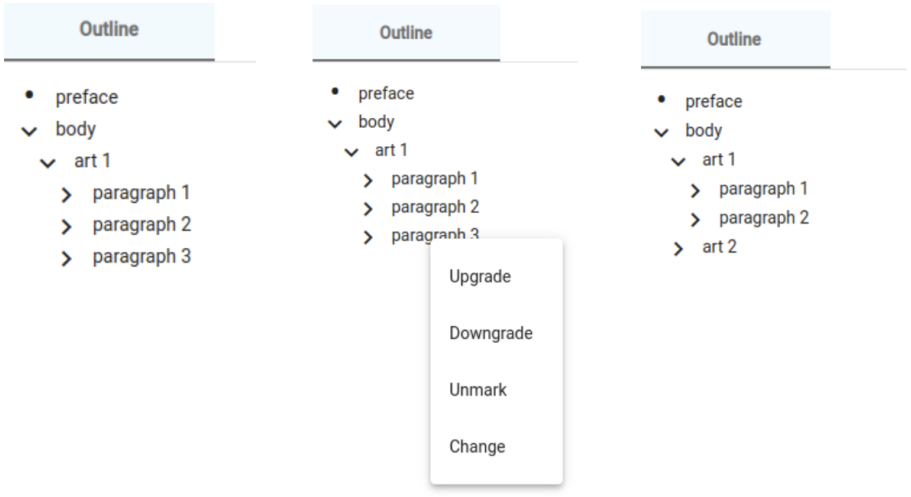


Figura 5.8: Esempio di operazione di Upgrade

5.7 Gestione dei tasti speciali

L'editor utilizza i tasti speciali come ad esempio "Enter", "Tab" oppure lo "Shift-Enter" per gestire agevolmente operazioni di navigazione o inserimento di elementi all'interno della struttura del documento. Per tale scopo sono definite una serie di regole per la gestione del comportamento di tali tasti, che assegnano ad ognuno un'operazione in base al contesto nel quale sono utilizzati.

Come caso di studio prendiamo l'aggiunta di un nuovo elemento: in precedenza, era necessario utilizzare il menù di marcatura per richiederne l'inserimento, selezionando tipo e caratteristiche desiderate. Con l'introduzione dei miglioramenti oggetto di questa tesi diventa invece possibile ottenere lo stesso risultato in maniera agevolata, sfruttando l'evento utente associato al tasto Enter. Come possiamo apprezzare nell'immagine sotto riportata, l'operazione crea un nuovo elemento avente le caratteristiche del precedente in modo continuativo, automatico e coerente.

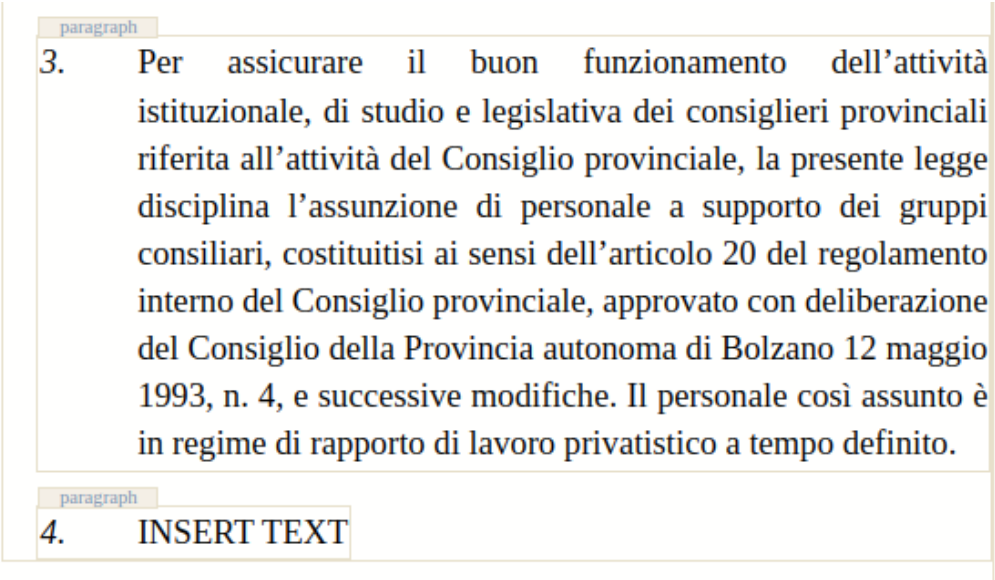


Figura 5.9: Risultato di un'operazione d'inserimento tramite Enter

Capitolo 6

Valutazioni e risultati

6.1 Test di validità

Al fine di valutare concretamente le operazioni aggiunte sono state condotte sessioni di test per verificare la loro correttezza ed efficacia, insieme ad un gruppo di esperti nello sviluppo e nell'utilizzo di editor che fanno uso dello standard Akoma Ntoso. Questo ha permesso la raccolta di dati utili al fine di valutare la correttezza, l'utilità e la facilità di utilizzo durante le fasi di marcatura di un documento. Innanzitutto siamo partiti testando la funzionalità di diverse operazioni ed innovazioni testando che funzionassero su di un documento già in formato Akoma Ntoso, predisposto e corretto, di cui era importante mantenere la validità. In seguito abbiamo esteso l'analisi e i relativi test a documenti testuali, non ancora nel formato Akoma Ntoso, in cui la struttura del documento andava creata marcando il contenuto puramente testuale, o addirittura creando un documento da zero.

Abbiamo quindi iniziato predisponendo test che verificassero la correttezza delle operazioni di modifica, inserimento, eliminazione e gestione degli elementi strutturali o del contenuto testuale del documento. Il test prevedeva l'utilizzo di un documenti

in formato Akoma Ntoso non banali, presi da un contesto professionale: nello specifico, abbiamo utilizzato come materiale alcuni documenti reali, aventi quindi una complessità elevata e con strutture complesse al loro interno. Abbiamo utilizzato diversi documenti per variare legislazioni, strutture, convenzioni e stili di redazione, in modo da variegare il soggetto.

Gli esperti dovevano effettuare operazioni di modifica, inserimento, eliminazione e gestione degli elementi strutturali o del contenuto testuale del documento.
Nel caso della modifica dovevano provare ad effettuare inserimenti da tastiera per modificare il contenuto o operazioni di "change" per verificare la praticità di questo nuovo strumento introdotto. In questo caso, va soprattutto analizzato il punto di vista dell'utilizzatore: ci si chiede quindi se si tratta di una operazione di cui si sente davvero il bisogno, e se riesce a dimostrarsi sufficientemente intuitiva da migliorare l'efficienza di questa modifica (confrontata con l'effettuare la stessa operazione con gli strumenti preesistenti non automatizzati).
Nel caso dell'inserimento dovevano provare ad utilizzare le operazioni associate al tasto "Enter" verificando la correttezza di due diverse funzionalità nell'inserimento di nuovi elementi, con o senza contenuto: se viene premuto con il cursore posizionato alla fine di un contenuto testuale di un elemento, l'operazione inserirà un nuovo elemento privo di contenuto, in coda a quello che contiene il cursore, con le stesse caratteristiche strutturali del precedente; se viene premuto con il cursore in un punto intermedio del contenuto testuale di un elemento, l'operazione inserirà un nuovo elemento, in coda a quello che contiene il cursore, dello stesso tipo e contenente il testo dell'elemento precedente successivo alla posizione del cursore. Per entrambe le operazioni, la struttura circostante va mantenuta coerente e corretta. Se per esempio vogliamo inserire un nuovo elemento all'interno di una lista numerata, il numero dell'elemento deve essere corretto e i numeri di tutti gli elementi successivi devono cambiare di conseguenza.

Nel caso dell'eliminazione dovevano provare ad effettuare operazioni di "smarcatura" degli elementi, verificando che il sistema riuscisse a gestire correttamente l'eliminazione della struttura sottostante senza errori.

Tutti questi test sono stati eseguiti con le medesime modalità, fornendo a un gruppo di esperti del settore la situazione di partenza e il task da eseguire, analizzandone la soluzione e chiedendo informazioni sulla intuitività logica e opinioni generali, giudizi pratici come ad esempio piccole variazioni della funzionalità non considerate precedentemente che aumenterebbero l'applicabilità dell'operazione, ed errori riscontrati. Questi test hanno permesso di raccogliere le opinioni del gruppo di esperti e di ripensare, correggere o raffinare le funzionalità introdotte e testate in base alle loro considerazioni.
In particolare, mancando una esperienza nell'utilizzo concreto del software, le funzionalità sviluppate ed introdotte effettuavano l'operazione corretta nei casi "standard" ma avevano problemi nei casi limite. Tutte queste segnalazioni di errori nell'applicazione di queste funzionalità in casi speciali in cui si sono comportate in maniera inaspettata sono state raccolte e analizzate.

In conclusione, i test si sono rivelati particolarmente utili per l'organizzazione della fase di sviluppo, permettendo una migliore comprensione dell'utilizzo delle funzionalità in fase di lavorazione fondato sulla esperienza sul campo degli esperti coinvolti. Sulla base delle loro segnalazioni e suggerimenti è stato possibile migliorare, correggere e rielaborare le operazioni sviluppate per arrivare a un risultato solido e affidabile, che rispetti i requisiti di intuitività e funzionamento prevedendo un buon numero di casi limite. L'esperienza si è rivelata particolarmente utile sia sul fronte dello sviluppo della singola funzionalità specifica che per una più generale comprensione dei processi di sviluppo e utilizzo del software. Un'implementazione corretta di queste operazioni ha anche permesso di valutarne positivamente l'impatto sull'ef-

ficacia e efficienza di utilizzo, apprezzando i miglioramenti effettuati alle operazioni di modifica. In un caso semplice i miglioramenti effettuati a queste operazioni prevedono più intuitività e poche operazioni necessarie in meno, ma in una struttura complessa queste piccole differenze crescono fino a rappresentare un miglioramento sostanziale.

Dopo questi test abbiamo provato a effettuare analoghi test in un caso di utilizzo differente. Nello specifico, partendo da un documento puramente testuale, senza una struttura in formato Akoma Ntoso da modificare, con l'obiettivo di riuscire a crearla da zero. Al gruppo di esperti è stato dato un documento puramente testuale su cui sperimentare tentando di creare una struttura di marcatura valida e completa. I test puntavano ad ottenere sia validazioni in merito alla correttezza della struttura creata con le operazioni messe a disposizione, che valutazioni sul funzionamento e sulla effettiva praticità della cosa. Le operazioni da provare erano analoghe a quelle precedentemente analizzate, ma era particolarmente importante (visto che il focus del test era la creazione di una struttura complessa a partire da un documento non strutturato) l'inserimento di almeno un elemento per ogni tipologia di pattern (in modo da verificare problemi relativi ad ogni singolo pattern, oltre che all'utilizzo congiunto di pattern diversi in strutture annidate) verificando l'integrità strutturale del documento risultante. Ogni esperto ha potuto testare queste operazioni e segnalare qualsiasi funzionamento anomalo, errato o desiderato che notava durante i test. Queste segnalazioni sono poi state usate per valutare l'utilità e il funzionamento del sistema e direzionare lo sviluppo tenendo presenti queste direttive. Anche in questo caso gli esperti hanno potuto identificare alcuni casi di comportamenti inaspettati, errati o inadatti e fornire possibili miglioramenti, casi d'uso non previsti, situazioni da dover supportare e tecniche di utilizzo degli strumenti differenti.

In conclusione, le segnalazioni riscontrate grazie ai test ci hanno permesso di distinguere ulteriori sviluppi dei componenti, specificando più precisamente alcune loro proprietà definite a partire dallo standard Akoma Ntoso ed aggiungendo eccezioni e template interni di base per altri componenti.
Questa implementazione di template interni è stata uno degli sviluppi più utili e significativi derivanti da questi test. Abbiamo potuto apprezzare come il processo di marcatura non sia univoco, ma personale, e il sistema deve essere in grado di supportare la creazione della struttura a partire da elementi diversi a seconda delle preferenze logiche dell'utente, e con criteri di priorità differenti. Pertanto, certi elementi devono poter esistere isolati durante il processo, o magari devono prevedere l'esistenza o meno di altri elementi all'interno e, per esempio, se prendiamo un elemento contenitore, è essenziale che sia in grado non solo di leggere il testo al suo interno e usarlo come contenuto, ma anche riconoscere che l'utente può avere già creato una struttura articolata e specifica per quell'elemento testuale e il container deve riuscire a prendere e contenere quell'elemento già creato, e non un nuovo componente base con solo il testo del documento dentro.

6.2 Sfide incontrate durante l'implementazione

Nel corso dell'attuazione delle recenti funzionalità, ho notato la manifestazione di difficoltà nell'operatività delle attuali funzioni già presenti all'interno dell'applicazione. Queste problematiche mi hanno portato ad effettuare correzioni e miglioramenti, facendo in modo che il funzionamento generale rispettasse le specifiche richieste.

L'architettura di SIMPLEX nasce con l'intenzione di essere altamente personalizzabile graficamente ma anche funzionalmente. Utilizzando i file di configurazione dell'applicazione, infatti, è possibile definire le specifiche grafiche dei componenti della stessa. Questo approccio però richiede di avere a disposizione fin da subito

la configurazione per tutte le tipologie di documenti che vengono trattati, che non sempre sono disponibili. Per ovviare a questa problematica si è deciso di rendere la creazione e la gestione del menù di markup indipendente, utilizzando l'implementazione fornita in precedenza, ottenendo così la possibilità di avere un comportamento dinamico ed una configurazione più chiara e facilmente personalizzabile.

Ho comunque cercato, dove possibile, di utilizzare metodi di gestione forniti da Angular, come ad esempio gli *Observer*[1]e i *Subject*[2]), che forniscono un potentissimo strumento asincrono per l'aggiornamento dei dati, utile ad esempio per notificare al layout che i dati da visualizzare sono cambiati.

6.3 Possibili miglioramenti e estensioni

L'applicazione continua a fare progressi e soddisfare sempre di più le esigenze degli utenti, ma ci sono ancora alcuni aspetti che vanno migliorati per rendere SIMPLEX un valido successore di LIME.

In base a quanto è stato implementato ci sono funzionalità che non vengono ancora sfruttate appieno dall'applicazione, in quanto richiedono l'utilizzo di servizi gestiti lato server ma che al momento non sono ancora stati completati.

Sarà necessario convertire i componenti attualmente gestiti mediante funzioni JavaScript in classi gestite da Angular, in modo da poter sfruttare appieno le funzionalità fornite dal framework, aumentando così la chiarezza nella gestione delle operazioni principali dell'applicazione.

[1]Un Observer è un oggetto che osserva un *observable* e risponde agli eventi emessi da quest'ultimo (`https://rxjs.dev/guide/observer`)

[2]Un Subject è sia un Observable che un Observer, ciò significa che può essere sottoscritto come un normale Observable, ma può anche emettere nuovi valori(`https://rxjs.dev/guide/subject`

Facciamo quindi luce su alcuni punti di forza aggiunti in seguito al cambiamento effettuato:

1. si ha una dipendenza minore da *tinymce* per la gestione di determinati comportamenti che attualmente risultano complicati da gestire;

2. possibilità di utilizzare le funzionalità dell'applicazione anche con editor diversi da *tinymce*;

3. ottimizzazione delle performance tramite la riduzione del numero di chiamate intermedie atte al funzionamento di determinate operazioni;

4. codice più chiaro che ci permette di effettuare cambiamenti diretti senza incorrere in modifiche a catena.

Capitolo 7

Conclusioni

La ricerca condotta ha proposto un'implementazione solida per la gestione delle principali azioni all'interno di un editor dedicato alla manipolazione di documenti conformi allo standard *Akoma Ntoso*. Come evidenziato nei capitoli precedenti, l'attenzione principale è stata rivolta a semplificare e guidare l'utente nelle operazioni di marcatura e modifica del documento.

Focalizziamoci sui principali aspetti che rendono SIMPLEX uno strumento efficace:

- **L'utilizzo del menù contestuale per eseguire operazioni rapide**
  L'introduzione di operazioni all'interno del menù contestuale ha notevolmente accelerato la manipolazione della struttura e del contenuto del documento. Questo approccio consente all'utente di eseguire rapidamente le principali azioni dell'editor, migliorando significativamente l'efficienza durante il processo di modifica.

- **Flessibilità e personalizzazione delle regole**
  I cambiamenti e i miglioramenti apportati alle funzionalità esistenti

hanno reso l'applicazione più flessibile dal punto di vista della personalizzazione delle regole e della struttura del menù di marcatura. Ciò permette alle organizzazioni di adattare l'editor alle proprie esigenze specifiche, garantendo una maggiore aderenza ai requisiti normativi.

- **Utilizzo di framework avanzati**

  Sfruttando le potenzialità che Angular mette a disposizione, è stato possibile ottenere comportamenti dinamici per i componenti principali dell'applicazione. Questa implementazione ha contributo ad un aumento delle prestazioni nella gestione degli eventi generati dall'utente e delle operazioni eseguite dall'editor per la corretta manipolazione della struttura del documento.

- **Controllo e gestione degli eventi**

  Il processo di approvazione delle azioni di modifica è stato migliorato grazie all'implementazione di controlli avanzati. Le azioni eseguite dall'utente sono sottoposte a regole specifiche e la struttura del documento è attentamente verificata prima dell'approvazione, o in seguito alla gestione di eventi legati all'inserimento di nuovi elementi strutturali.

- **Feedback per una maggiore chiarezza**

  L'introduzione di feedback, sia lato sviluppatore che utente, in seguito alle operazioni eseguite permette una comprensione più approfondita del flusso di risoluzione degli eventi e delle eventuali incompatibilità con le regole di editazione o con la struttura del documento. Questo contribuisce a una maggiore trasparenza e consapevolezza durante il processo di editing.

In conclusione, si può affermare che l'implementazione proposta ha notevolmente arricchito le funzionalità di SIMPLEX rendendolo più veloce, flessibile e aderente alle esigenze specifiche degli utenti. L'utilizzo di tecnologie avanzate ha contribuito all'ottimizzazione delle prestazioni complessive dell'applicazione, offrendo un ambiente di modifica avanzato e intuitivo per i documenti conformi allo standard *Akoma Ntoso*.

Il futuro di SIMPLEX prevede l'aggiunta di ulteriori funzionalità atte a semplificare le operazioni di marcatura e manipolazione della struttura del documento, ma anche per la gestione di nuovi elementi strutturali in modo adeguato, come ad esempio le tabelle.
Sarà però necessario riadattare alcune funzionalità presenti per permettere la gestione di operazioni più complesse.

Tra le principali novità che sono prossime ad una concretizzazione abbiamo l'aggiunta dell'evento di drag & drop all'interno dell'outline, che permetterà di spostare agevolmente elementi strutturali senza dover ricorrere al copia e incolla e/o ad una dispendiosa serie di operazioni di *Mark* e *Unmark*. L'operazione verrà affiancata dai set di regole impostate che si occuperanno di gestire l'approvazione delle azioni eseguite. Così facendo si eviterà di creare situazioni di incompatibilità nella struttura del documento.

Si vuole inoltre introdurre la gestione delle *documentCollection*, un particolare tipo di documento supportato da *Akoma Ntoso* che, come ci suggerisce il nome, non è nient'altro che una collezione di documenti che possono avere tipo e giurisdizione diversa. La particolarità di questa tipologia di documento è la possibilità di avere riferimenti ad altri documenti.

Per questo motivo si ha bisogno di adattare l'editor in modo da rendere possibile la visualizzazione e la modifica non soltanto della collezione intera, ma anche quella

dei singoli documenti al suo interno in modo indipendente, andando a risolvere il riferimento fornito.
Al momento l'implementazione fornita per il menù di markup è già in grado di adattarsi dinamicamente al tipo e alla giurisdizione del documento attualmente in modifica, permettendo quindi di avere sempre un menù dinamico contenente soltanto gli elementi validi per la struttura in cui ci si trova.

Un aspetto che sicuramente verrà rivisto in futuro è quello del controllo della correttezza del documento, in particolar modo nella gestione dei feedback forniti all'utente nel caso si incorra in situazioni di errore. Una possibile proposta emersa in seguito a mie indagini effettuate a riguardo è quella di andare ad integrare al messaggio di errore una possibile soluzione che renda valido il documento, fornendo ove possibile uno o più metodi di risoluzione in modo da facilitare la correzione. Questo approccio renderebbe l'applicazione utilizzabile anche da utenti meno esperti nell'ambito della marcatura di documenti.

Un'ulteriore funzionalità richiesta è quella di rendere possibile l'estensione degli elementi direttamente dalla grafica proposta per identificare il contenuto dell'elemento. L'idea proposta prevede di rendere i bordi di ogni elemento trascinabili, permettendo così di andare ad includere o escludere agevolmente testo o elementi dal contenuto dell'elemento. Questa funzionalità permetterà di risparmiare tantissimo tempo nei casi in cui la marcatura non sia avvenuta in modo preciso, infatti può capitare di includere o escludere accidentalmente del contenuto durante la marcatura. Al momento l'unica soluzione richiederebbe l'esecuzione di almeno un'operazione di rimozione della marcatura, seguita da operazioni di marcatura per ottenere il risultato desiderato.

Infine un ulteriore sviluppo riguarderà la creazione di documenti partendo da un template predefinito. Ogni template potrà essere del tutto personalizzato e modifi-

cabile, fornendo una struttura di base di elementi scelti in modo appropriato: per un template generico i più comunemente usati per quella tipologia di documento; per uno personalizzato quelli scelti in fase di definizione dello stesso. Questa soluzione potrà poi essere personalizzata a piacimento da ogni ente per adattarlo alla relativa giurisdizione ed alle specifiche esigenze delle singole.

Bibliografia

[BCP+10] Gioele Barabucci, Luca Cervone, Monica Palmirani, Silvio Peroni, and Fabio Vitali. Multi-layer markup and ontological structures in akoma ntoso. In Pompeu Casanovas, Ugo Pagallo, Giovanni Sartor, and Gianmaria Ajani, editors, *AI Approaches to the Complexity of Legal Systems. Complex Systems, the Semantic Web, Ontologies, Argumentation, and Dialogue*, pages 133–149, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[BFP+08] Mariangela Biasiotti, Enrico Francesconi, Monica Palmirani, Giovanni Sartor, and Fabio Vitali. Legal informatics and management of legislative documents. In Giovanni Sartor, editor, *Global Center for ICT in Parliament Working Paper No.2*, January 2008. `https://lexsummerschool.files.wordpress.com/2009/09/wp002_legislativeinformatics1.pdf`.

[Cer13] Luca Cervone. *Parametric editors for structured documents.* Tesi di laurea Magistrale CdS informatica, Università di Bologna, 2013. `https://amslaurea.unibo.it/id/eprint/6196`.

[CPV24] Luca Cervone, Monica Palmirani, and Fabio Vitali. Akoma ntoso website, 2024. `http://www.akomantoso.org/?page_id=25`, visitato a Febbraio 2024.

[D'A24] Andrea D'Arpa. *SIMPLEX Streamlining Legal Document Marking Software*. Tesi di laurea Magistrale CdS informatica, Università di Bologna, 2024. data di discussione 14 marzo 2024.

[Dzi22] Marek Patryk Dziedzic. *Redesign and implementation of an Akoma-Ntoso based web editor for the markup of legal documents*. Tesi di laurea CdS informatica, Università di Bologna, 2022. https://amslaurea.unibo.it/id/eprint/25313.

[PV12] Monica Palmirani and Fabio Vitali. Legislative xml: principles and technical tools. 2012. https://webimages.iadb.org/publications/english/document/Legislative-XML-Principles-and-Technical-Tools.pdf.

[VP24] Fabio Vitali and Monica Palmirani. Akoma ntoso: principi generali. Università di Bologna. Agenzia per l'Italia Digitale, 2024. https://www.agid.gov.it/sites/default/files/repository_files/documentazione/20150525_akoma_architettura_unibo_f_vitali_v_1_0.pdf,visitato a Febbraio 2024.

[VZ07] Fabio Vitali and F. Zeni. Towards a country-independent data format: the akoma ntoso experience. In *Proceedings of the V legislative XML workshop. Vol. 6786*. Firenze: European Press Academic, 2007. https://www.academia.edu/download/30706652/art5.pdf.