Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

SVILUPPO DI UN EDITOR DI METADATI PER SIMPLEX

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
ANDREA ZECCA

Sessione I
Anno Accademico 2021/2022

"We used to look up at the sky
and wonder at our place in the
stars, now we just look down
and worry about our place in the dirt."

- Cooper, 'Interstellar'.

Sommario

Il presente lavoro si basa su un progetto applicativo chiamato *Simplex*, un editor per documenti legislativi redatti nel formato Akoma Ntoso, basato su standard XML e nato con lo scopo di ovviare ai problemi del suo predecessore LIME.
Obiettivo principale del lavoro è la creazione di un componente fondamentale nell'editor Simplex, ovvero l'editor di metadati. Come suggerito dal nome, il componente ha il compito di interfacciarsi con il documento che si sta elaborando e in particolar modo con i metadati che descrivono la natura del documento, dando la possibilità all'utente di modificare questi ultimi. Durante lo sviluppo di questo progetto sono state create e definite le basi applicative su cui la tesi si è incentrata. Questo progetto è stato sviluppato tenendo in considerazione anche delle possibili future implementazioni che verranno discusse nel capitolo dedicato agli sviluppi futuri.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Le leggi sono sempre state alla base della vita di società evolute. Inizialmente erano tramandate attraverso rappresentazioni scritte su carta da oratori di legge, che rendevano la loro elaborazione possibile solo da esseri umani specializzati in ambiti giuridici.
L'avvento di nuove tecnologie e la loro rapida diffusione hanno portato ad una digitalizzazione delle risorse e la diffusione di Internet ha reso possibile l'accesso a queste ultime in qualsiasi momento. Per sfruttare appieno queste nuove tecnologie sono stati ideati e diffusi degli standard[1] che permettessero la stesura e la descrizione di leggi che risultassero comprensibili anche da calcolatori elettronici.
Tra le varie proposte troviamo lo standard Akoma Ntoso.
L'utilizzo di questi standard ha facilitato l'organizzazione dei documenti legali, ma ha anche scaturito il bisogno di creare degli ambienti speciali atti alla creazione e alla modifica dei documenti redatti seguendo queste regole.
Nascono così i primi editor testuali con strumenti di supporto per la stesura di documenti secondo questi nuovi standard.
Tra di essi troviamo **LIME**, acronimo di *"Language Independent Markup Editor"*, e il suo successore **Simplex**.

> "LIME è un editor di markup open-source, parametrico e indipendente da qualsiasi linguaggio XML. La sua architettura è basata sui pattern XML, su alcune linee guida per

[1] Insieme di specifiche ufficiose o ufficiali, prestabilite da un'autorità e riconosciute tali con lo scopo di rappresentare una base di riferimento

> la creazione di documenti XML e, soprattutto, su parametri inseriti all'interno di file di configurazione JSON. L'idea alla base di LIME è che è possibile astrarre un linguaggio XML assegnando ad ognuno dei suoi content model uno dei pattern utilizzati dal linguaggio Akoma Ntoso. Questa astrazione può essere descritta con parametri (array associativi) all'interno di file di configurazione JSON. In questo modo, in LIME, è possibile abilitare altri linguaggi XML semplicemente scrivendo un insieme di file JSON impacchettati in un modo comprensibile al software. Questo significa anche che ogni eventuale estensione non necessita di competenze specifiche in nessun linguaggio di programmazione."[2]

Il progetto LIME è alla base dell'applicativo utilizzato durante lo sviluppo di questa tesi, ovvero Simplex.
La necessità di modificare e aggiornare LIME deriva principalmente da un'architettura obsoleta che limita le sue funzionalità, il suo utilizzo e la sua eventuale capacità ad adattarsi ai cambiamenti evolutivi.
LIME risulta essere un editor *"rigido"*, poco flessibile e poco propenso a eventuali cambiamenti. È stato progettato facendo uso di tecnologie vecchie che rendono impossibile il suo aggiornamento a nuove versioni. Il codice di LIME è stato scritto in ExtJS: una libreria JavaScript per la costruzione di applicazioni Web interattive.
Nel corso degli anni, l'utilizzo di questa libreria ha reso quasi impossibile l'aggiunta di nuove funzionalità, rendendo quelle esistenti obsolete. Infatti, la progettazione e lo sviluppo di nuove funzioni richiede la scrittura del loro codice da zero, senza poter riutilizzare del codice già scritto per altre funzionalità, aumentando così i tempi di sviluppo.
Per ovviare a questi problemi, è stata sviluppata una nuova applicazione: Simplex.
Essa nasce per sostituire LIME, rendendola un'applicazione basata sui componenti. Il vantaggio di questi ultimi è la separazione che è possibile attuare tra i vari componenti dell'applicazione, rendendoli indipendenti gli uni dagli altri, in modo tale che lo sviluppo e/o la modifica di uno non interferisca con gli altri. Grazie ad un'architettura basata sui componenti è, infatti, possibile l'aggiunta di nuove funzionalità in maniera semplice e senza dover capire l'intera struttura dell'applicazione.
Lo sviluppo di Simplex prevede l'utilizzo di nuove tecnologie che permettano una rapida modifica all'intera applicazione, una totale compatibilità con lo standard Akoma Ntoso (come LIME) e la creazione di un file di configurazione che permetta di rendere l'applicazione indipendente dal contesto e modificabile all'occorrenza in maniera semplice.

Basta, infatti, modificare il file di configurazione, senza scrivere del nuovo codice o cambiare quello esistente, per ottenere una nuova applicazione con funzionalità differenti.
Facciamo un passo indietro e andiamo ad analizzare una delle caratteristiche che ha reso LIME uno dei miglior editor per documenti Akoma Ntoso.
Una delle più importanti funzionalità di LIME era data dalla presenza di un editor per i metadati. Quest'ultimo permetteva l'aggiunta, la rimozione e la modifica dei metadati associati al documento sul quale si stava lavorando.
I metadati ricoprono un ruolo fondamentale all'interno del Web Semantico, in quanto permettono una descrizione più accurata del documento, definendo delle proprietà intrinseche ad esso che possono essere interpretate da calcolatori per "*capirne*" il contesto e migliorare le ricerche.
Essendo Simplex un'applicazione ancora in fase di sviluppo, vi era la necessità di generare per essa un editor per i metadati che fosse il più simile possibile a quello di LIME, ma che fosse privo di tutti i suoi problemi e che rispettasse la logica alla base dello sviluppo del nuovo componente.
Obiettivo principale del progetto di tesi è stato, quindi, la realizzazione di un editor per i metadati estremamente parametrizzato per Simplex. L'idea alla base è quella del *"minimo sforzo, massimo risultato"*, ottenuta adottando la seguente logica: si genera un file di configurazione che descrive la struttura dell'editor di metadati e come i vari campi interagiscono con i metadati del documento su cui si vuole lavorare; successivamente il software genera dinamicamente l'applicazione seguendo i parametri passati in input tramite il file di configurazione.
L'utilizzo di un file di configurazione e lo sviluppo di Simplex basato interamente su di esso hanno fatto in modo che venissero superati i limiti imposti dal suo predecessore LIME. Infatti, è ora possibile modificare l'applicazione facilmente, essendo essa basata su componenti, ma è anche possibile generare delle nuove istanze semplicemente modificandone il file di configurazione. Allo stesso modo, è possibile modificare, sempre tramite file di configurazione, l'editor dei metadati, ottenendo varie versioni con diverse funzionalità.
Per capire il funzionamento dell'editor e come è possibile sfruttare il file di configurazione, bisogna dare uno sguardo alle varie tecnologie utilizzate nella realizzazione dell'applicazione e alla struttura dei documenti, focalizzandosi principalmente sulla sezione dei metadati, analizzandone la loro struttura e capendo come questi siano fondamentali

nella generazione dell'URI del documento, identificativo unico del documento stesso.
La tesi è divisa come segue:

- Il capitolo 2 offre delle informazioni aggiuntive riguardo l'ambito del problema, focalizzandosi maggiormente sulle precedenti tecnologie (in particolare LIME) e i loro relativi limiti, e su come l'utilizzo di software innovativi (l'editor Simplex) abbia contribuito a superare queste difficoltà;
- Nel capitolo 3 verrà fornita una visione ad alto livello dell'applicazione, analizzandone il file di configurazione e la sua sintassi. Verrà mostrato come al variare del file, cambino sia l'aspetto, che le funzionalità dell'editor di metadati;
- Nel capitolo 4 verrà spiegata l'implementazione dell'applicazione e la sua struttura;
- Nel capitolo 5 si analizzeranno le qualità del software. In particolare, verrà spiegato come è possibile verificarne la corretta funzionalità analizzandone l'effetto sui documenti che si stanno elaborando e come l'applicazione sviluppata sia facilmente modificabile tramite il file di configurazione;
- Infine, nell'ultimo capitolo, verranno tratte le conclusioni e date delle idee per sviluppi futuri dell'applicazione.

Capitolo 2

Markup di documenti legali

Simplex è un'applicazione per la stesura e per il successivo markup di documenti legali con informazioni aggiuntive che aiutano a descriverne la struttura e il contenuto.
Come anticipato, Simplex nasce dall'esigenza di modernizzare un editor già esistente per far fronte ai problemi legati ad esso.
In questo capitolo verrà data una descrizione generale dell'editor LIME e verranno analizzati i problemi principali legati ad esso. Successivamente, verrà illustrato il nuovo editor Simplex e ne verranno analizzati i punti di forza, i quali hanno permesso l'eliminazione dei problemi precedenti.
Prima di analizzare nel dettaglio i due editor testuali, cerchiamo di capire cosa significa eseguire il markup di documenti legali e come Akoma Ntoso ci aiuta in questa operazione.
Alla base del markup di documenti legali e di Akoma Ntoso, troviamo XML:

> "The Extensible Markup Language (XML) is a simple text-based format for representing structured information: documents, data, configuration, books, transactions, invoices, and much more. It was derived from an older standard format called SGML (ISO 8879), in order to be more suitable for Web use."[12] [1]

[1] Il linguaggio XML, acronimo di *"Extensible Markup Language"*, è un semplice formato testuale per rappresentare informazioni strutturate: documenti, dati, configurazioni, libri, transazioni, fatture e molto altro. È stato derivato da un vecchio formato standard chiamato SGML (ISO 8879), per essere più adatto all'uso Web

La rappresentazione di documenti legali tramite standard XML presenta dei vantaggi:

- Contiene informazioni che contribuiscono alla direzione del flusso di lavoro;
- Supporta la redazione legislativa nazionale e le *"best practices"*;
- Permette ad ogni istituzione di marcare i propri documenti;
- È arricchito dal contributo dei cittadini che possono aggiungere delle annotazioni sui file;
- Preserva la validità legale dei documenti;
- È accessibile a tutti tramite diversi canali e chiunque può accedere ai documenti per effettuare delle ispezioni;
- Può essere usato con dei tool comuni e sistemi di gestione di documenti.[9]

XML viene, quindi, utilizzato come base dei linguaggi per markup di documenti legali. Si possono creare delle connessioni tra i vari documenti definendo al loro interno dei link espliciti verso altre risorse. In questo modo, si viene a creare una specie di Web dei documenti. Questo è possibile grazie alla definizione e all'utilizzo di metadati utili a generare l'URI[2]/IRI[3] delle risorse.
Uno standard diffuso, per il markup di documenti legali e dei loro relativi metadati, è proprio Akoma Ntoso, basato su pattern XML.

Akoma Ntoso

Akoma Ntoso, acronimo di *"Architecture for Knowledge-Oriented Management of African Normative Texts using Open Standards and Ontologies"*, nasce nel 2004 da un progetto

[2]Uniform Resource Identifier. Identifica universalmente e univocamente una risorsa

[3]Un IRI è una forma generale di Uniform Resource Identifier costituita da una sequenza di caratteri appartenenti all'Universal Character Set. Ciò significa che al suo interno possono occorrere caratteri non appartenenti all'insieme ASCII.

dell'UNDESA[4], il quale ha come iniziativa quella di *"Rafforzare i sistemi informativi dei parlamenti in Africa"* [1]. Obiettivi principali di questo progetto sono:

1. Migliorare la qualità dei servizi parlamentari;
2. Facilitare il lavoro dei parlamenti;
3. Promuovere l'accesso ai processi parlamentare alla società.

Nasce come standard internazionale per rappresentare documenti legislativi, esecutivi e giudiziari. Il termine *Akoma Ntoso* deriva dalla lingua Akan parlata dai popoli di alcune regioni dell'Africa occidentale, significa *"cuori connessi"* e indica comprensione ed accordo [10].
Questo standard definisce una serie di rappresentazioni *"machine readable"*[5] ed è composto da:

- Un vocabolario XML per mappare la struttura dei documenti legali con il rispettivo termine XML;
- Uno schema XML che definisce la struttura dei documenti e le regole da rispettare per crearli.

Lo schema XML di Akoma Ntoso fornisce una descrizione dei documenti parlamentari, legislativi e giudiziari. Permette inoltre un markup aggiuntivo dei documenti, creando la struttura e i componenti semantici dei documenti legislativi digitali, permettendo loro di essere completamente *machine-readable*[11] e di poter far parte del Web Semantico. Ad ogni documento sono, infatti, associate informazioni e dati aggiuntivi che ne specificano il contesto semantico in un formato adatto all'interrogazione, all'interpretazione e all'elaborazione automatica.
Alla base di Akoma Ntoso troviamo il supporto alle seguenti funzionalità:

- Orientamento giuridico-legislativo: lo standard fornisce una descrizione della struttura principale di documenti legali e legislativi;

[4]United Nations Department of Economic and Social Affair
[5]Dati leggibili e comprensibili dalla macchina

- Affidamento a standard esistenti: Akoma Ntoso fa uso di standard efficienti e già largamente utilizzati come XML, URIs e RDF;
- Ontologia: qualsiasi informazione che specifichi il contesto giuridico o istituzionale in cui un documento svolge un ruolo.[11]

Ovviamente, la creazione dello standard AKN[6] ha aumentato il bisogno di avere strumenti e spazi adatti per lavorare su documenti redatti secondo lo standard stesso.
Nascono così i primi editor per documenti Akoma Ntoso, tra i quali troviamo BungeniX, LEOS, LegisPro e LIME.
BungeniX[8] viene sviluppato principalmente per i parlamenti Africani e consente la redazione di documenti legali e legislativi secondo lo standard AKN. I punti di forza di questo editor sono stati la possibilità di convertire documenti Word in AKN e la presenza di uno strumento di revisione che permetteva di tenere traccia dei cambiamenti.
Successivamente, viene sviluppato LEOS, utilizzato maggiormente all'interno dell'Unione europea. È un software open source progettato per aiutare coloro che sono coinvolti nella redazione della legislazione, che di per sé è un processo complesso ad alta intensità di conoscenza, supportando un'efficiente collaborazione online[7].
Uno dei più recenti editor è invece LegisPro, che fornisce ai redattori di leggi, di regolamenti e di altri documenti normativi essenziali una piattaforma di creazione basata su browser che offre un processo di modifica, pubblicazione e codifica senza interruzioni[5].
Oltre ai tre editor citati precedentemente, viene creato quello che sarà alla base dell'applicativo software utilizzato durante lo sviluppo del lavoro relativo a questa tesi. Il suo nome è LIME e verrà analizzato nella prossima sezione.

LIME

> "LIME is an extremely customizable web based editor that guides the user through the markup of non structured documents into well formed (optionally valid) structured XML document compliant to the language plugin chosen by the user. The LIME editor is an open source software and relies on many open source technologies. LIME is currently under development by the CIRSFID and the University of Bologna." [3]

[6] Abbreviazione di Akoma Ntoso

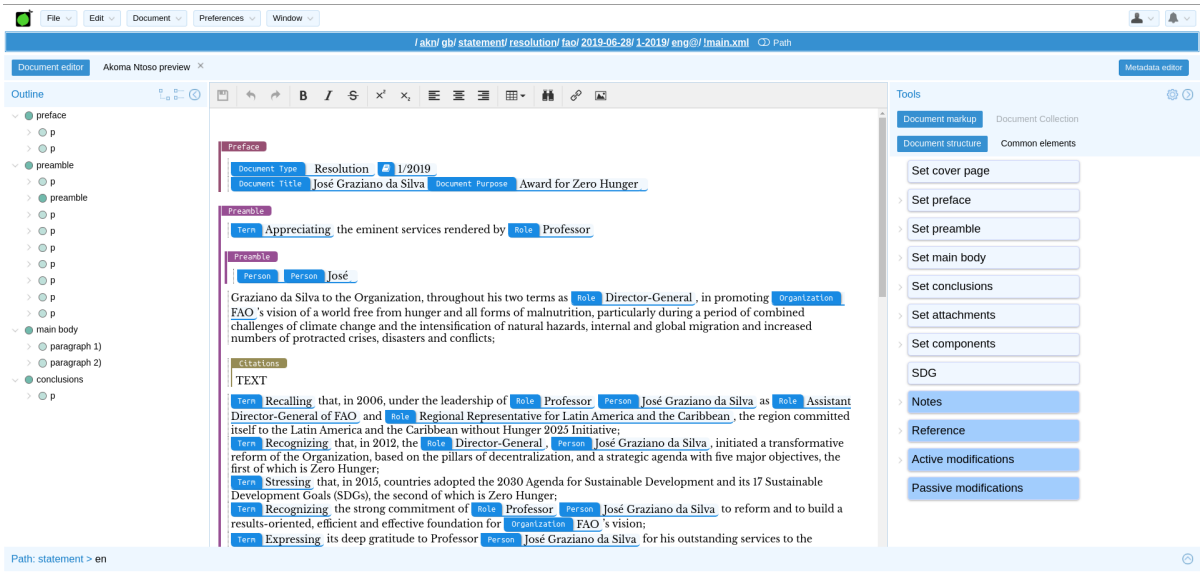


Figura 2.1: Editor LIME

Nella figura 2.1 è possibile vedere l'interfaccia dell'editor LIME.
Tutte le funzionalità dell'applicazione sono divise in cinque sezioni separate:

- La *toolbar*, posizionata in alto, contiene tutte le possibili operazioni come importazione/esportazione di documenti ecc.;
- Il pannello *Outline*, posizionato sulla sinistra, dà la possibilità di vedere la struttura del documento;
- L'editor testuale, posizionato al centro;
- Il pannello di markup, posizionato sulla destra, dal quale è possibile aggiungere elementi dello standard Akoma Ntoso al testo;
- L'editor di metadati, posizionato sul fondo (visualizzabile cliccando sul bottone *"Metadata editor"* in alto a destra).

Come già anticipato uno dei punti chiave dell'editor LIME è la presenza di una sezione che prende il nome di *"Editor di metadati"*. Essa ricopre un ruolo fondamentale in quanto poter aggiungere delle informazioni al documento, rappresentate dai metadati, permette una migliore descrizione dello stesso. Un calcolatore può, infatti, analizzare

i metadati, associati al documento, per capirne il contesto. In questo modo può effettuare delle ricerche basate sul contesto e non relative a frasi o contenuti presenti nel testo.

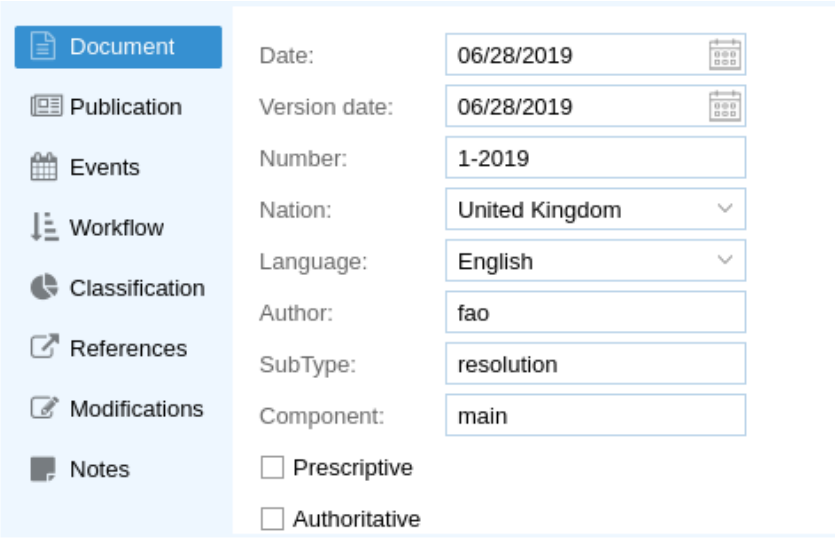


Figura 2.2: Esempio di editor di metadati per LIME

Nella figura 2.2 è possibile visualizzare l'editor di metadati di una particolare istanza di LIME, in questo caso stiamo considerando l'istanza LIME per *FAO*[7].
Come si può notare dall'immagine, è possibile lavorare con numerosi metadati di ambiti diversi.
La struttura dei campi relativi ai metadati verrà affrontata in una delle prossime sezioni, in cui vedremo come Akoma Ntoso gestisce i metadati di documenti legali.

Limiti di LIME

Il progresso tecnologico e la nascita di nuovi modi per marcare documenti legali hanno scaturito la necessità di mantenere LIME costantemente aggiornato.
Durante il suo sviluppo, l'utilizzo, di tecnologie obsolete ha limitato i possibili aggiornamenti, rendendoli difficili e macchinosi da effettuare.

[7] *Food and Agriculture Organization*: l'Organizzazione delle Nazioni Unite per l'alimentazione e l'agricoltura.

LIME è stato sviluppato appoggiandosi su ExtJS: una libreria JavaScript per la costruzione di applicazioni web interattive. L'utilizzo di questa libreria, unito ad una continua necessità di effettuare modifiche al software per implementare nuove funzionalità, ha reso, a distanza di tempo, il codice dell'applicazione intricato e di difficile comprensione.
Aggiungere una nuova funzionalità significa, quindi, scriverne il codice da zero, senza poter far riuso di codice già scritto precedentemente, diminuendo l'efficienza e la velocità dell'editor.
Per far fronte ai problemi causati dall'architettura di LIME, si è deciso di iniziare a sviluppare un nuovo editor.
Requisiti fondamentali della nuova versione dell'editor sono:

- Pieno supporto allo standard AKN;
- Creazione e/o modifica di documenti Akoma Ntoso;
- Avere le stesse funzionalità del precedente editor LIME;
- Limitare i problemi dovuti all'architettura, adottandone una moderna ed elastica;
- Presenza di un editor per i metadati dei documenti, fondamentale in LIME.

Dall'esigenza di soddisfare questi requisiti inizia lo sviluppo dell'editor Simplex, di cui parleremo nella prossima sezione.

Simplex

Simplex nasce per ovviare ai problemi creati da LIME, causati in particolare dalla sua architettura e dalla necessità di mantenerlo costantemente aggiornato.
Viene ideato come Web App[8] basata sui componenti.
L'utilizzo di questa nuova tecnologia ha ridotto drasticamente gran parte dei problemi dati da LIME. I principali vantaggi nello sviluppo di Web App basate sui componenti sono:

- Aumento della riusabilità dei componenti;

[8] Applicazione distribuita e fruibile tramite servizi Web per mezzo di una rete

- Riduzione del tempo di sviluppo;
- Riduzione dei costi;
- Miglior qualità del software.[6]

Inoltre, per sviluppare un nuovo componente (nel caso di questo studio si tratta dell'editor di metadati) non è necessaria una conoscenza totale dell'architettura e dell'applicazione. Ogni componente è indipendente e il suo sviluppo non è influenzato, né influenza, il comportamento degli altri componenti. Questo permette uno sviluppo più semplice e meno invasivo.

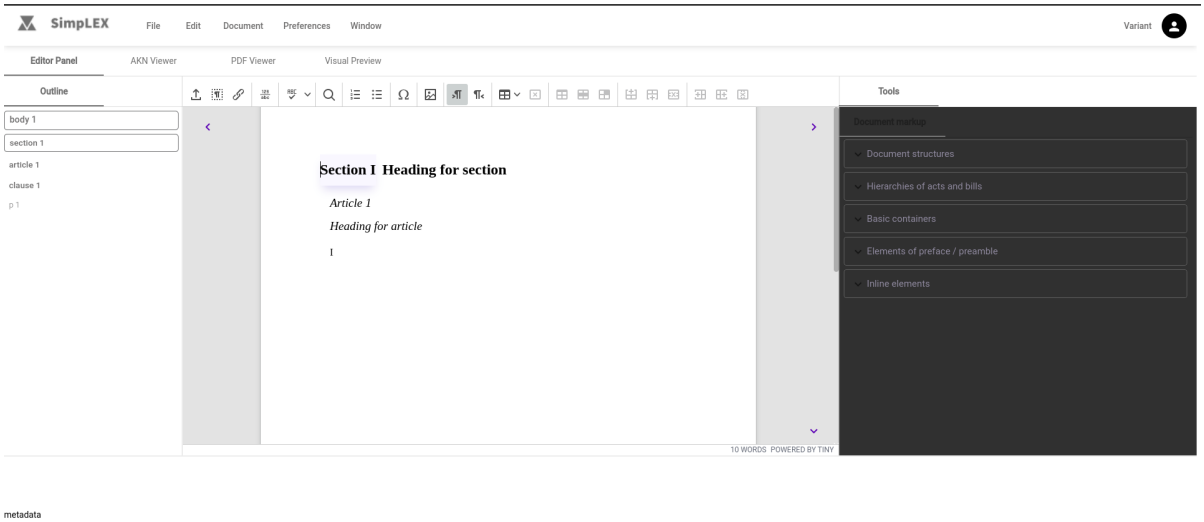


Figura 2.3: Editor Simplex

Nell'immagine 2.3 è possibile vedere l'interfaccia dell'editor Simplex.
Anche in questo caso l'editor è diviso in 5 sezioni:

- La *toolbar*, posizionata in alto, contiene tutte le possibili operazioni come importazione/esportazione di documenti ecc.;
- Il pannello *Outline*, posizionato sulla sinistra, dal quale è possibile vedere la struttura del documento;
- L'editor testuale, posizionato al centro;

- Il pannello di markup, posizionato sulla destra, dal quale è possibile aggiungere elementi Akoma Ntoso al testo;
- L'editor di metadati, assente nella prima versione di Simplex, posizionato sul fondo.

Il vero punto di forza del progetto Simplex è la definizione di un file di configurazione che descrive l'editor sia a livello grafico, che a livello funzionale.
La sola modifica del file di configurazione permette, infatti, di avere diverse versioni dell'editor senza modificare il codice sorgente.
Il file contiene una descrizione di ogni componente dell'editor, ad eccezione della descrizione della parte relativa all'editor di metadati.
Scopo del progetto di tesi è stato la scrittura di quest'ultima sezione del file e l'implementazione del componente relativo.
Il file di configurazione verrà analizzato in una sezione dedicata del prossimo capitolo.
Come già anticipato, il punto di forza di LIME era dato dalla presenza dell'editor per i metadati dei documenti. Per capire come funzionano questo editor e la sua nuova implementazione, dobbiamo analizzare la struttura dei metadati nello standard Akoma Ntoso.
Nella prossima sezione verrà descritto il funzionamento e il significato dei principali metadati dei documenti AKN.

Metadati in Akoma Ntoso

Lo standard AKN fornisce il blocco *<meta>* utilizzato per contenere i metadati per il documento e all'interno di questo sono presenti diversi elementi, ciascuno usato per descrivere un aspetto diverso dei metadati. I vari elementi sono:

- *<identification>*: contiene tutti i fatti rilevanti su date e autori del documento. Include anche informazioni su possibili traduzioni;
- *<publication>*: contiene informazioni riguardo la pubblicazione del documento;
- *<classification>*: dedicato all'assegnazione di parole chiave al documento in base all'argomento trattato nel contenuto legale;
- *<lifecycle>*: contiene tutti gli eventi che hanno modificato il documento nel corso del tempo;

- *<workflow>*: contiene la sequenza di flusso di un documento, ovvero la sequenza di passaggi che interessano un documento. Si differenzia da *<lifecycle>* perché non implica una modifica al documento;
- *<analysis>*: contiene descrizioni e annotazioni che qualificano semanticamente il contesto del documento;
- *<temporalData>*: contiene gruppi che hanno informazioni temporali sul documento;
- *<references>*: contiene tutti i riferimenti ad altri documenti o ad ontologie;
- *<notes>*: contiene le note editoriali (non autorevoli) fatte da istituzioni;
- *<presentation>*: contiene specifiche per il supporto alla resa visiva del documento;
- *<proprietary>*: permette di aggiungere qualsiasi altro tag personalizzato utile alla gestione di metadati aggiuntivi.[4]

In ogni documento, gli elementi *<identification>, <publication> e <references>* sono obbligatori, mentre i restanti possono essere omessi.
Analizziamo questi 3 elementi fondamentali.

<identification>

<identification> è l'elemento più importante del blocco di metadati di un documento. Contiene delle informazioni relative al documento e non al suo contenuto, come la data di creazione, l'autore e molto altro.
I sotto elementi del tag sono descritti tramite l'ontologia FRBR[9] e sono divisi in 3 macro sezioni:

- *<FRBRWork>*: metadati sulla creazione intellettuale distinta;
- *<FRBRExpression>*: metadati su una forma specifica della creazione intellettuale;
- *<FRBRManifestation>*: metadati sulla rappresentazione di un'espressione.

[9] *Functional Requirements for Bibliographic Records*: definisce uno schema concettuale realizzato tramite modello entità-relazione allo scopo di dare una rappresentazione alle informazioni bibliografiche.

Questi elementi contengono metadati espressi tramite ontologia FRBR, ad esempio FRBRdate per le date, FRBRauthor per l'autore, FRBRlangauge per la lingua e molto altro.

```
<identification source="#source">
    <FRBRWork>
        <FRBRthis value="/akn/un/statement/deliberation/fao/2017-07-07/2-2017/!main"/>
        <FRBRuri value="/akn/un/statement/deliberation/fao/2017-07-07/2-2017"/>
        <FRBRdate date="2017-07-07" name=""/>
        <FRBRauthor as="#author" href="#fao"/>
       <FRBRcountry value="un"/>
    </FRBRWork>
    <FRBRExpression>
        <FRBRthis value="/akn/un/statement/deliberation/fao/2017-07-07/2-2017/eng@/!main"/>
        <FRBRuri value="/akn/un/statement/deliberation/fao/2017-07-07/2-2017/eng@"/>
        <FRBRdate date="2017-07-07" name=""/>
        <FRBRauthor as="#author" href="#fao"/>
       <FRBRlanguage language="eng"/>
    </FRBRExpression>
    <FRBRManifestation>
        <FRBRthis value="/akn/un/statement/deliberation/fao/2017-07-07/2-2017/eng@/!main.xml"/>
        <FRBRuri value="/akn/un/statement/deliberation/fao/2017-07-07/2-2017/eng@.xml"/>
        <FRBRdate date="2017-10-24" name=""/>
        <FRBRauthor as="#author" href="#fao"/>
   </FRBRManifestation>
</identification>
```

Figura 2.4: Esempio di tag <identification>

Nella figura 2.4 si può vedere un esempio del blocco *<identification>*, insieme alla sua struttura e il suo contenuto.
L'importanza di questo blocco di metadati deriva dalla possibilità di generare l'URI basandosi su caratteristiche del documento che, messe insieme, lo rendono unico.
L'URI è rappresentato dal tag <FRBRuri> ed è presente in ognuno dei tre sotto elementi:

- FRBRWork URI:

  `Struttura:`
  `/akn/{jurisdiction}/{docType}/{subType}/{actor}/{date}/{docNumber}`

  jurisdiction: Codice di due lettere basato su standard ISO 3166-1[10] che identifica il Paese.
  docType: Identifica il tipo di documento.
  subType: Indica, se presente, il sottotipo del documento.
  actor: L'autore del documento.
  date: La data in cui è stato creato.
  docNumber: Il numero del documento.
  Un esempio di FRBRWork URI è il seguente:

[10]Standard che fornisce i codici per i nomi di Paesi.

```
<FRBRuri value="/akn/gb/statement/resolution/fao/2019-06-28/1-2019"/>
```

- FRBRExpression URI:

  `Struttura:`
  `[work URI]/{lang}@{date}`

  lang: Codice di tre lettere basato su standard ISO 639-2[11] che identifica la lingua.
  date: Data di creazione del documento.
  Un esempio di FRBRExpression URI è il seguente:

```
<FRBRuri value="/akn/gb/statement/resolution/fao/2019-06-28/1-2019/eng@">
```

- FRBRManifestation URI:

  `Struttura:`
  `[expression URI]/.{format}`

  format: Formato del documento.
  Un esempio di FRBRManifestation URI è il seguente:

```
<FRBRuri value="/akn/gb/statement/resolution/fao/2019-06-28/1-2019/eng@.xml"/>
```

<publication>

Il blocco <publication> contiene dati relativi alla pubblicazione del documento.

```
<publication
    name="Official Letter"
    date="2001-12-13"
    showAs="Official Letter"
    number="S-2001-1189"/>
```

Figura 2.5: Esempio di tag <publication>

Nella figura 2.5 si può vedere un esempio di un documento pubblicato con il nome *"Official Letter"* in data "2001-12-13".

[11]Standard per la classificazione dei linguaggi.

<references>

Il blocco <references> modella tutti i riferimenti ad altri documenti o a classi di ontologia.

```
<references source="#source">
    <TLCPerson eId="source" href="/akn/ontology/person/somebody" showAs="Somebody"/>
    <TLCTerm eId="noting" href="/akn/ontology/term/un/noting" showAs="Noting"/>
    <TLCTerm eId="decides" href="/akn/ontology/term/un/decides" showAs="Decides"/>
    <TLCReference eId="conference" name="" href="/akn/ontology/organization/un/conference" showAs="conference"/>
    <TLCOrganization eId="libya" href="/akn/ontology/organization/un/libya" showAs="Libya"/>
    <TLCConcept eId="paymentpurpose" href="/akn/ontology/concept/un/paymentpurpose" showAs="Payment"/>
    <TLCOrganization eId="libya" href="/akn/ontology/organization/un/libya" showAs="Libya"/>
    <TLCRole xmlns="" eId="author" href="/akn/ontology/role/author" showAs="Author"/>
    <TLCOrganization xmlns="" eId="fao" href="/akn/ontology/organization/un/fao" showAs="FAO"/>
</references>
```

Figura 2.6: Esempio di tag <references>

Nella figura 2.6 si può notare come vengono descritti i riferimenti alle ontologie. Per i riferimenti a soggetti giuridici (come organizzazioni, persone etc.), vengono usati tag speciali con prefisso TLC. Tramite i riferimenti ad altri documenti e/o ontologie è possibile per un calcolatore dedurre il contesto del documento senza doverlo analizzare.

Importanza dei metadati nel web

L'estensione del *WWW*[12] verso il Web Semantico ha come scopo principale la generazione di informazioni che non siano solo *"read-only"*[13], ma che possano essere utilizzate anche da calcolatori e motori di ricerca per aumentare l'efficienza delle ricerche. Ad esempio, senza l'aggiunta di queste meta informazioni l'unico modo che abbiamo per cercare un documento è tramite delle parole che sono contenute in esso. Invece, con la definizione di informazioni aggiuntive gli autori possono specificare delle caratteristiche del documento che sono interpretabili e utilizzabili dai motori di ricerca per migliorare le loro prestazioni e le ricerche.
Allo stesso modo, anche nei documenti legali la definizione di informazioni aggiuntive tramite i metadati ricopre un ruolo fondamentale. Questi ultimi servono infatti a generare l'URI del documento per il suo riconoscimento. Inoltre, tramite l'utilizzo del blocco <references>, è possibile capire il contesto del documento solo analizzandone i riferimenti.

[12] World Wide Web
[13] Un dato che può essere solamente letto e non permette operazioni su di esso

Capitolo 3

Simplex Metadata Editor

Definito l'ambito di lavoro e il contesto dell'applicazione, in questo capitolo analizzeremo come il sistema, chiamato con il nome *"Simplex Metadata Editor" (SMDE)*, risolve i problemi lasciati aperti dalle precedenti implementazioni.
In particolare, analizzeremo il file di configurazione, cercando di capirne la struttura, per scrivere, successivamente, la sezione del file relativa all'editor di metadati. Inoltre, vedremo le possibilità di utilizzo dell'editor di metadati da parte di un utente finale, come esso agisce sul documento su cui si sta lavorando e i principali vantaggi di questa implementazione. I dettagli tecnici e le spiegazioni riguardanti le modalità di lettura del file di configurazione e di creazione dell'editor di metadati verranno, invece, affrontate nel prossimo capitolo, relativo alla descrizione a basso livello dell'applicazione

Implementazione tramite file di configurazione

Il principale problema di LIME derivava dalla sua architettura rigida e basata su tecnologie obsolete che lo hanno reso, nel tempo, impossibile da mantenere aggiornato con nuovi componenti e funzionalità.
Durante lo sviluppo di Simplex è stato definito un file di configurazione che descrivesse l'intera applicazione e le sue funzionalità. In questa sezione analizzeremo il file di configurazione, cercando di capirne la struttura per scrivere, successivamente, la sezione del file relativa all'editor di metadati.

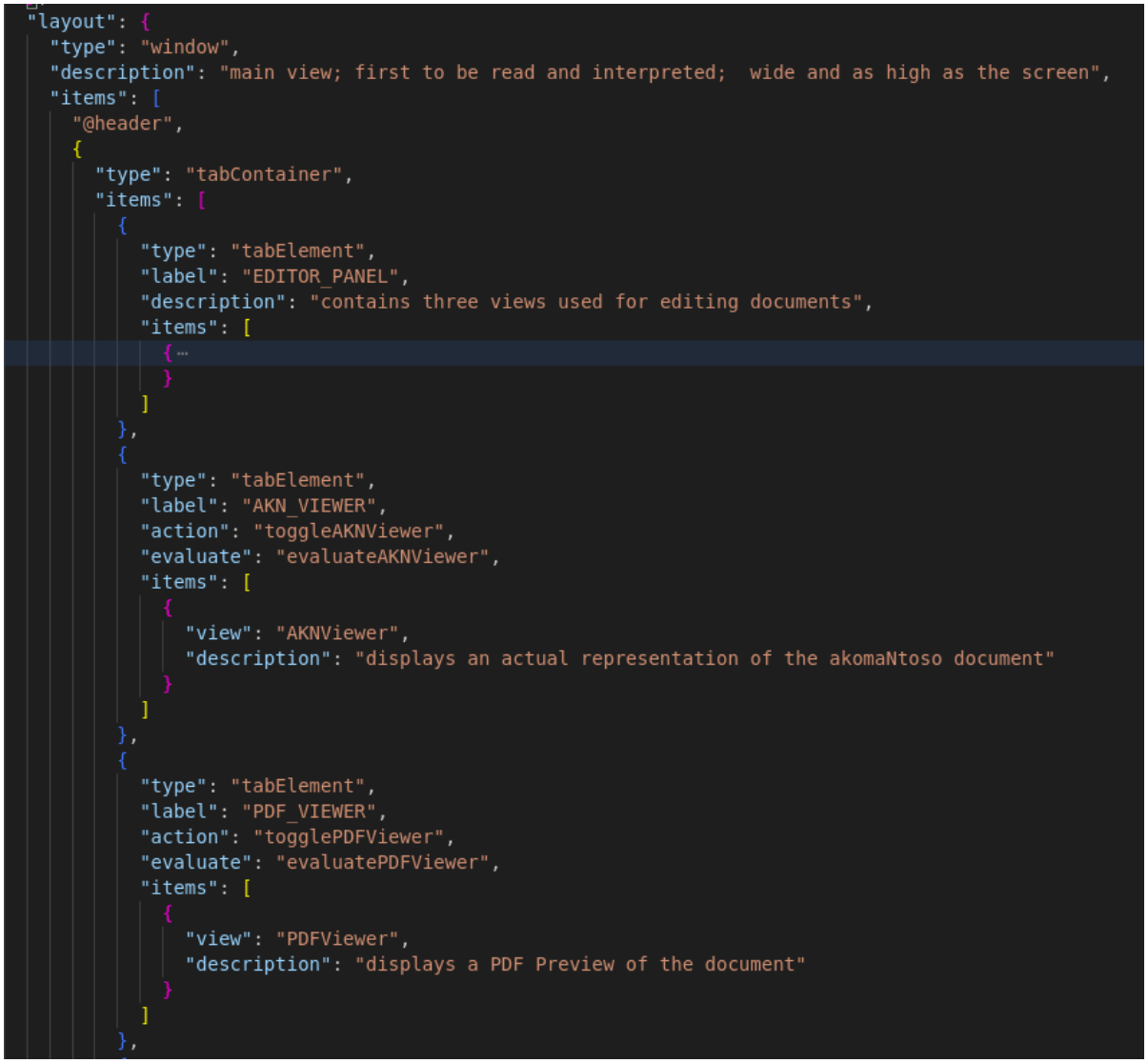


Figura 3.1: Parte del file di configurazione

Nella figura 3.1 è mostrato l'inizio del file di configurazione. In particolare, è possibile notare in che modo avviene la descrizione dei vari componenti dell'editor Simplex tramite questo file scritto in formato JSON[1].

In questa figura è riportata parte della descrizione del layout dell'editor Simplex. Nelle righe successive, passiamo in analisi il file per comprenderne la struttura.

Il campo *layout* contiene dei parametri quali *type* e *description* che identificano il tipo

[1]JavaScript Object Notation, è formato adatto all'interscambio di dati fra applicazioni client/server.

dell'oggetto descritto e una descrizione testuale (utile a chi legge il file).
Successivamente, vi è il campo *items* che contiene un array di oggetti, ognuno dei quali descrive una sezione dell'editor.
Ogni elemento del file, e di conseguenza ogni componente dell'editor, è descritto tramite questo file ed è caratterizzato da alcuni campi tra cui:

- *type*: indica il tipo dell'oggetto descritto;
- *label*: contiene un'etichetta che viene tradotta nel linguaggio selezionato con cui si vuole usare l'editor, tramite un sistema di traduzione;
- *description*: contiene una descrizione testuale dell'oggetto, usata per indicare a chi legge il file cosa si sta descrivendo e a cosa serve l'oggetto;
- *items*: se l'oggetto descritto è un oggetto complesso e strutturato, questo campo contiene un array con la descrizione di ogni sotto-elemento che compone l'oggetto descritto;
- *action*: se presente, indica la callback[2] da eseguire quando viene cliccato l'oggetto;
- *evaluate*: se presente, indica la funzione tramite la quale è possibile ottenere lo stato dell'oggetto che si sta descrivendo.

La descrizione dell'editor segue una logica ricorsiva. Quando si descrive un oggetto, oltre alle sue caratteristiche, vengono descritti anche i sotto-elementi che lo compongono seguendo la stessa struttura organizzativa e la stessa sintassi.
Per capirne meglio la descrizione, analizziamo nel dettaglio un oggetto.

[2]Funzione che viene eseguita se si verifica un evento specifico.

Figura 3.2: Analisi nel dettaglio di un oggetto nel file di configurazione

Nella figura 3.2 è presente la descrizione del bottone che esegue il *"toggle"* dell'editor di metadati. Come si può notare dalla descrizione, il campo *type* contiene il valore "button", mentre il campo *action* contiene il nome della primitiva che, eseguita, causerà il *toggle* dell'editor di metadati. Infine, il campo *items* contiene delle direttive che indicano quale componente renderizzare (in questo caso il componente con nome "metadataEditor"), oltre ad una descrizione testuale.
Grazie all'utilizzo di questo file di configurazione è possibile risolvere gran parte dei problemi che si riscontravano con l'architettura di LIME. Infatti, è possibile ottenere delle versioni distinte di Simplex semplicemente modificando i valori di questo file e senza dover cambiare il codice dell'applicazione. Si può, così, avere un'architettura flessibile e modificabile facilmente in qualsiasi momento.
L'utilizzo di tecnologie basate sui componenti risolve gli altri problemi legati al vecchio editor. È, infatti, possibile creare e aggiungere dei nuovi componenti con facilità, lavorando solo su essi e senza conoscere l'intero funzionamento dell'applicazione.
Lo sviluppo dell'editor di metadati, effettuato come progetto relativo a questa tesi, è stato possibile senza una conoscenza completa dell'applicazione: è stato possibile lavorare solo su di esso, senza dover interagire con gli altri componenti.
Nel prossimo capitolo verrà descritta nel dettaglio la parte del file di configurazione relativa all'editor di metadati.

Utilizzo editor di metadati ed effetti sul file

In questa sezione ci soffermeremo su come, per un utente finale, sia possibile utilizzare l'editor di metadati implementato e su come le modifiche effettuate tramite l'editor si ripercuotano sul file su cui si sta lavorando.
L'editor di metadati è composto da vari *tabs*[3], ognuno dei quali è legato a una diversa sezione di metadati dello standard AKN. Ogni tab, quindi, si interfaccia con un gruppo diverso di metadati relativo al documento.

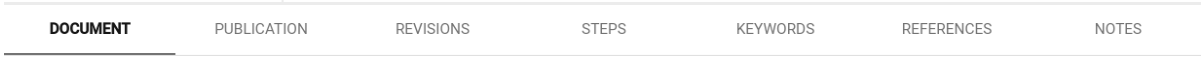


Figura 3.3: *Tabs* di una versione dell'editor di metadati

Nella figura 3.3 si possono vedere i *tabs* di una particolare istanza dell'editor di metadati. In questo caso sono presenti i seguenti sette *tabs*:

1. *DOCUMENT*: permette di modificare metadati associati al documento (tag <identification>).
2. *PUBLICATION*: relativo ai metadati che descrivono la pubblicazione del documento (tag <publication>).
3. *REVISIONS*: gestisce i metadati relativi alle modifiche che hanno cambiato il documento (tag <lifecycle>).
4. *STEPS*: relativo alla sequenza di passaggi effettuati dal documento (tag <workflow>).
5. *KEYWORDS*: aggiunge parole chiave al documento in base al contenuto.
6. *REFERENCES*: gestisce i metadati per i riferimenti ad altri documenti od ontologie (tag <references>).
7. *NOTES*: per i metadati relativi alle note editoriali sul documento (tag <notes>).

[3]Tab: controllo grafico (widget) detto di navigazione che permette all'utente di muoversi da un gruppo di controlli a un altro.

Per capire come utilizzare i vari *tabs* e come essi si interfacciano con il documento, modificandolo in base ai valori inseriti, analizziamo nel dettaglio alcuni *tabs* e vediamo che effetto hanno nella sezione del documento dedicato ai metadati.

Analisi tab *DOCUMENT*

Figura 3.4: Tab *DOCUMENT* dell'editor di metadati

In questa particolare istanza dell'editor di metadati, il tab *DOCUMENT* contiene i seguenti campi:

- DATE: data di creazione del documento;
- VERSION DATE: data di creazione della particolare espressione del documento;
- NUMBER: numero del documento;

- NATION: nazione in cui è redatto;
- LANGUAGE: lingua in cui è scritto;
- AUTHOR: autore del documento;
- SUBTYPE: sottotipo del documento;
- COMPONENT: nome del documento;
- PRESCRIPTIVE/AUTHORITATIVE: indicano se un documento è prescrittivo o autoritativo o entrambi.

Il seguente tab si interfaccia con i documenti presenti nell'elemento *<identification>* [vedi 2.4] e ogni campo del tab modifica a dovere un elemento contenuto nel blocco *<identification>*.
Ad esempio, partendo dai metadati seguenti [fig. 3.5a], l'editor assume la configurazione in 3.6b:

(a) Blocco metadati documento prima delle modifiche

(b) Tab *DOCUMENT* prima delle modifiche

Figura 3.5: Metadati ed editor prima delle modifiche

Andando a modificare tramite l'editor alcuni metadati, si ottiene il seguente effetto sul documento:

(a) Blocco metadati documento dopo le modifiche

(b) Tab *DOCUMENT* dopo le modifiche

Figura 3.6: Metadati ed editor dopo delle modifiche

Come è possibile notare, le modifiche effettuate tramite l'editor di metadati hanno ripercussioni istantanee sul file su cui si sta lavorando, rendendo così l'utente in grado di modificare i metadati dei documenti.
Modificando i campi *DATE*, *VERSION DATE*, *AUTHOR* e *PRESCRIPTIVE/AUTHORITATIVE*, sono stati cambiati i valori dei relativi campi associati nel blocco dei metadati del documento.

Analisi tab *NOTES*

Adesso analizzeremo un'altra tab dell'editor di metadati che si occupa di registrare eventuali note editoriali fatte sul documento. Partiremo da un documento che non contiene note e poi ne aggiungeremo alcune tramite l'editor di metadati per visualizzare il risultato.

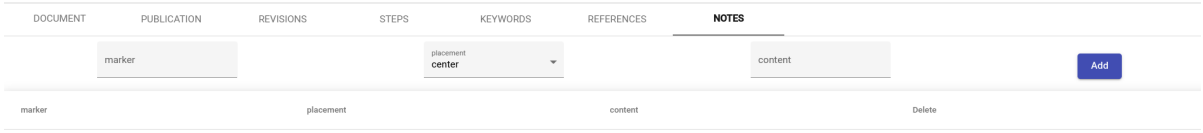


Figura 3.7: Struttura tab NOTES

Aggiungiamo delle note di test tramite l'editor e vediamo l'effetto finale sul documento.

```
<meta>
    <notes>
        <note marker="MarkerTest1" placement="center" content="Nota di test 1"/>
        <note marker="MarkerTest2" placement="right" content="Nota di test 2"/>
    </notes>
</meta>
```

(a) Blocco metadati notes dopo le modifiche

marker	placement	content	Delete
MarkerTest1	center	Nota di test 1	Delete
MarkerTest2	right	Nota di test 2	Delete

(b) Tab *NOTES* dopo le modifiche

Figura 3.8: Note post modifica

Capitolo 4

Architettura di Simplex Metadata Editor

In questo capitolo verranno fornite delle conoscenze di base per poter comprendere il codice e lavorare con esso.
Analizzeremo nel dettaglio la sezione del file di configurazione nella quale è descritto l'editor di metadati. Successivamente verrà spiegato come l'applicazione legge questo file e genera l'editor di metadati dinamicamente, facendo in modo di aggiornare il documento in contemporanea alle modifiche effettuate tramite l'editor.

Organizzazione file di configurazione

L'editor di metadati è caratterizzato da un file di configurazione che ne descrive i componenti principali e come essi interagiscono tra di loro. L'uso di questo file di configurazione permette la creazione di varianti personalizzate dell'editor in maniera semplice e lineare. Basterà, infatti, modificare il file di configurazione aggiungendo o rimuovendo dei blocchi di codice descrittivi, rispettando la sintassi usata, per creare delle varianti dell'editor. La descrizione dell'editor si trova all'interno del file

`@assets/configuration/config.json`

L'editor è organizzato tramite diverse *tabs*, ognuna delle quali descrive una sezione dell'editor e si interfaccia con un gruppo di metadati distinto.
Analizziamo una particolare istanza dell'editor di metadati per capirne la sintassi e come viene generata l'applicazione associata ad esso.

```
"@metadataEditor": {
  "@comment": "Metadata Editor Container",
  "label": "METADATA_EDITOR",
  "currentActive": "@metadataDocument",
  "items":  [
    {
      "type": "tabElement",
      "label": "DOCUMENT",
      "description": "@metadataDocument"
    },
    {
      "type": "tabElement",
      "label": "PUBLICATION",
      "description": "@metadataPublication"
    },
    {
      "type": "tabElement",
      "label": "EVENTS",
      "description": "@metadataEvents"
    },
    {
      "type": "tabElement",
      "label": "WORKFLOW",
      "description": "@metadataWorkflow"
    },
    {
      "type": "tabElement",
      "label": "CLASSIFICATION",
      "description": "@metadataClassification"
    },
    {
      "type": "tabElement",
      "label": "REFERENCES",
      "description": "@metadataReferences"
    },
    {
      "type": "tabElement",
      "label": "NOTES",
      "description": "@metadataNotes"
    }
  ]
},
```

Figura 4.1: Descrizione di una istanza dell'editor di metadati

Come affermato precedentemente, l'editor di metadati è organizzato in varie *tabs*, ognuna delle quali è descritta nel file di configurazione. Nella figura 4.1 si può vedere l'*"entry point"* della descrizione dell'applicazione. Esso è descritto tramite i seguenti campi:

- *@comment*: contiene un commento che descrive testualmente l'oggetto per aiutare il lettore;
- *label*: etichetta testuale;
- *currentActive*: indica su quale *tab* dell'editor posizionare il *focus* all'avvio;
- *items*: contiene un array in cui ogni oggetto indica la presenza di una *tab* diversa dell'editor. L'elemento *items* contiene i seguenti campi:
  - *type*: indica il tipo dell'elemento;
  - *label*: etichetta testuale;
  - *description*: indica quale componente, descritto nel file di configurazione sotto la voce *"component"*, contiene la descrizione della tab.

Attualmente, le *tabs* possono essere di due tipi:

- *formContainer*: sono organizzate come se fossero delle *form* in cui l'utente visualizza e modifica/inserisce nuovi dati, tramite degli elementi di *input*;
- *tableContainer*: sono organizzate tramite delle tabelle in cui vengono mostrate le informazioni aggregate. L'utente può aggiungere nuove informazioni visualizzate come righe delle tabelle.

Adesso analizziamo la descrizione di alcuni *tabs* dell'editor di metadati per capirne la sintassi e poter, successivamente, modificare l'editor adattandolo alle varie esigenze. Analizzeremo un esempio per entrambe le tipologie di *tab*: per le *formContainer* considereremo la *tab DOCUMENT*, mentre per *tableContainer* consideriamo la *tab NOTES*.

Descrizione tab DOCUMENT

Questa tab si interfaccia con i metadati contenuti nell'elemento <identification> [2.4] tramite l'ontologia FRBR. La tab è descritta nel seguente modo:

```
"@metadataDocument": {
  "id": "document",
  "type": "formContainer",
  "title": "DOCUMENT",
  "@comment": "Metadata Document Container",
  "wrapper": "identification",
  "items": [
    {
      "type": "datePicker",
      "label": "DATE",
      "description": "Document's date",
      "default": "today",
      "tag": "FRBRWork/FRBRdate",
      "tagAttribute": "date"
    },
    {
      "type": "datePicker",
      "label": "VERSION DATE",
      "description": "Document's version date",
      "default": "today",
      "tag": "FRBRExpression/FRBRdate",
      "tagAttribute": "date"
    },
    {
      "type": "textInput",
      "label": "NUMBER",
      "description": "Document's number",
      "default": "",
      "tag": "FRBRWork/FRBRnumber",
      "tagAttribute": "value"
    },
    {
      "type": "selectInput",
      "label": "NATION",
      "values": "./metadata/nations.json",
      "description": "Jurisdiction",
      "default": "it",
      "tag": "FRBRWork/FRBRcountry",
      "tagAttribute": "value"
    },
```

(a) Prima parte della descrizione

```
    {
      "type": "selectInput",
      "label": "LANGUAGE",
      "values": "./metadata/lang.json",
      "description": "Document's language",
      "default": "eng",
      "tag": "FRBRExpression/FRBRlanguage",
      "tagAttribute": "language"
    },
    {
      "type": "textInput",
      "label": "AUTHOR",
      "description": "Document's author",
      "default": "fao",
      "tag": "FRBRWork/FRBRauthor",
      "tagAttribute": "href"
    },
    {
      "type": "textInput",
      "label": "SUBTYPE",
      "description": "Document's subtype",
      "default": "",
      "tag": "FRBRWork/FRBRsubtype",
      "tagAttribute": "value"
    },
    {
      "type": "checkboxInput",
      "label": "PRESCRIPTIVE",
      "default": "false",
      "tag": "FRBRWork/FRBRprescriptive",
      "tagAttribute": "value"
    },
    {
      "type": "checkboxInput",
      "label": "AUTHORITATIVE",
      "default": "false",
      "tag": "FRBRWork/FRBRauthoritative",
      "tagAttribute": "value"
    }
  ]
},
```

(b) Seconda parte della descrizione

Figura 4.2: Descrizione *tab DOCUMENT*

La descrizione contiene alcuni campi come un *id* che identifica la *tab*, la tipologia, un titolo, un commento e un *wrapper*. Quest'ultimo indica il *tag* che racchiude i metadati associati al *tab*. In questo caso, dato che *DOCUMENT* modifica i metadati associati al *tag* <*identification*>, *wrapper* contiene esattamente il nome del *tag*, cioè *identification*.

Oltre a questi campi standard, vi è anche un campo *items* che contiene gli elementi che fanno parte del *tab*. Ogni elemento in *items* è descritto nel seguente modo:

- *type*: indica il tipo del documento, ad esempio *datePicker* per le date, *textInput* per il testo, *selectInput* per il menù a tendina e *checkboxInput* per delle selezioni multiple. Se il *type="selectInput"*, viene aggiunto un campo *values* che contiene un dizionario[1] in cui sono riportati i possibili valori presenti nel menù a tendina;
- *label*: etichetta testuale;
- *description*: una descrizione testuale dell'elemento;
- *default*: valore di default assunto dal campo in assenza di informazioni;
- *tag*: una coppia di *tag* separati dal carattere "/" che indica in quale elemento si trova il metadato. Ad es., *"FRBRWork/FRBRdate"* indica che il metadato si trova nell'elemento *FRBRWork* sotto la voce *FRBRdate*;
- *tagAttribute*: l'attributo del tag nel file XML. Ad es., con i campi *tag= "FRBRWork/FRBRdate"* e *tagAttribute="date"*, il tag XML corrispondente che troveremo nel file AKN sarà

```
<FRBRWork>
    <FRBRdate date="2022-06-21"/>
</FRBRWork>
```

Descrizione tab NOTES

Questa tab si interfaccia con i metadati contenuti nell'elemento <notes>[3.7].

[1] Insieme di coppie *<key,value>*

Figura 4.3: Descrizione *tab NOTES*

Così come un *tab* di tipo *formContainer*, i *tab tableContainer* contengono sempre un *wrapper*, ma, si differenziano dai primi in quanto, al posto del campo *items*, è presente un campo *schema* che definisce lo schema della tabella che conterrà i dati all'interno dell'editor.
Ogni elemento nello *schema* è descritto nel seguente modo:

- *name*: nome dell'attributo
- *type*: tipo di input;
- *attribute*: indica l'attributo *inline* del tag che conterrà il dato;
- *required*: indica se l'attributo è obbligatorio o meno nella creazione di nuovi dati;

- *comment*: commento per aiutare il lettore.

I campi *attribute* vengono utilizzati nella creazione di nuovi *tag XML*. In questo caso, si aggiunge una nuova nota:

```
<notes>
    <note marker="testMarker" placement="testPlacement" content="testContest"/>
</notes>
```

Generazione applicazione

In questa sezione verranno descritti i passi effettuati dall'applicazione, iniziando con la lettura del file di configurazione e arrivando ad avere un'applicazione funzionante.
Essendo Simplex una Web App scritta in Angular, ha bisogno di essere generata per poter effettuare il suo *deploy*[2]. Durante la generazione dell'editor di metadati, viene letto il file di configurazione e viene generato il codice dell'applicazione in base al contenuto del file.
I passi seguiti nella generazione dell'editor sono i seguenti:

1. Lettura file di configurazione e documento di lavoro;
2. Generazione di una tabella in memoria per lo *storage* dei metadati del documento;
3. Parsing del documento per recupero di eventuali metadati;
4. Generazione delle *tabs* dell'editor di metadati.

Analizziamo i vari passi.

Lettura file di configurazione

Il file di configurazione viene letto dal suo *path* e vengono inizializzate delle variabili di supporto che contengono le descrizioni dei *tabs* presenti in base alla descrizione seguita secondo lo standard che è possibile vedere tramite la figura 4.1. In questa fase viene anche scaricato dal database il documento sul quale l'utente sta lavorando in formato

[2]Insieme delle attività che rendono un sistema software disponibile per l'utilizzo.

XML. Questo file viene, poi, convertito in JSON e tenuto in memoria.
La conversione da XML a JSON permette di lavorare su una struttura più semplice da gestire e modificare. Successivamente, quando l'utente effettua una modifica, essa viene salvata in questa struttura JSON, che viene poi convertita nuovamente in XML con i dati aggiornati, pronta per essere caricata sul database.

Generazione tabella di supporto

In questa fase viene creato in memoria un dizionario (con nome *storageTable*) utile a tenere traccia dei metadati che vengono inseriti tramite l'editor. Vengono analizzate le varie *tabs* e, in base alla loro tipologia, viene creato un campo in questo dizionario:

- Per le *tabs* di tipo *formContainer* viene creato un campo nel dizionario con il nome:

  `{wrapper}.{tag}`

  dove *tag* viene *"splittato"*: la stringa viene separata in base al carattere "/". Ad esempio, una stringa del tipo "FRBRWork/FRBRdate" diventa "FRBRWork.FRBRdate". Quindi, se una *tab* ha campo *wrapper="identification"* e *tag="FRBRWork/FRBRdate"*, il suo valore nella tabella si trova accedendo a *storageTable* indicizzato con chiave *"identificaion.FRBRWork.FRBRdate"*

- Per le *tabs* di tipo *tableContainer* viene creato un campo nel dizionario con il nome:

  `{wrapper}.{tag}`

  che contiene i campi:

  - *schema*: contiene lo schema della tabella descritta nel file di configurazione;
  - *values*: contiene i valori che vengono inseriti dall'utente rispettando lo schema definito nel campo precedente.

I campi di questa struttura vengono associati, in fase di generazione del codice delle *tabs*, ai vari elementi di *input* per tenere traccia delle modifiche fatte dall'utente tramite l'interfaccia grafica.

Parsing del documento

In questa fase viene eseguito il *parsing*[3] del documento. Durante questa procedura, tramite l'uso di determinate espressioni regolari, vengono raccolti i primi metadati associati al documento e caricati in memoria nella tabella di supporto definita precedentemente. Questi dati sono mostrati all'utente tramite l'interfaccia grafica generata successivamente e sono modificabili. L'obiettivo principale dell'editor di metadati è dare all'utente la possibilità di inserire manualmente, dove il *parser di metadati* non riesce a fare *"pattern matching"*, i metadati mancanti.

Generazione grafica delle *tabs* dell'editor di metadati

In questa fase viene generata l'interfaccia grafica delle *tabs*. Essendo Simplex una Web App scritta in *Angular*, vengono sfruttate le direttive di questo linguaggio per semplificare il lavoro. Ad esempio, tramite *ngIf* si può verificare la tipologia di una certa *tab* e visualizzarla a seconda di essa.
Inoltre, in questa fase viene eseguito il *"binding"*[4] tra gli elementi di input dell'interfaccia e i relativi campi associati che si trovano nella tabella di supporto. In questo modo, quando l'utente, modifica, tramite l'interfaccia un metadato si ha una ripercussione istantanea sugli elementi contenuti nella tabella di supporto. Inoltre, ad ogni elemento di input è associata una callback per gestire la sua modifica: questa funzione aggiorna la struttura JSON che descrive il file XML successivamente converte il JSON in XML e aggiorna il file contenuto nel database.

[3] Analisi sintattica mirata al riconoscimento di pattern tramite espressioni regolari.
[4] In informatica il binding è il processo tramite cui viene effettuato il collegamento fra una entità di un software ed il suo corrispettivo valore.

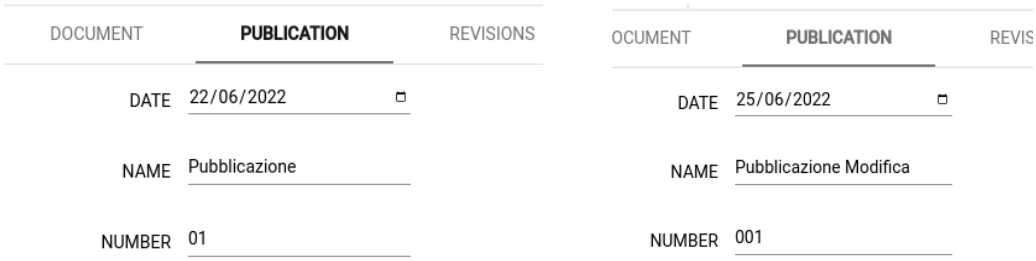
Capitolo 5

Valutazione SMDE

Una volta sviluppato il software non ci resta che verificarne la correttezza. Ma che significa valutare la correttezza di un programma?
Per dichiarare un programma come *"corretto"*, dobbiamo verificare se esso soddisfa i requisiti prefissati prima del suo sviluppo e se risponde alle necessità per le quali è stato creato.
In questo caso, per valutare la correttezza dell'editor di metadati verranno effettuati dei test *ad hoc*. Dobbiamo verificare, inizialmente, che la nuova versione dell'editor effettui delle modifiche valide ai documenti AKN, rispettandone la sintassi e la struttura. Successivamente, verificheremo l'efficienza: vedremo come è sufficiente modificare il file di configurazione per ottenere delle nuove istanze dell'editor.

Correttezza rispetto allo standard AKN

In questa fase del test proveremo a modificare i metadati di un documento tramite Simplex, analizzando i risultati ottenuti. Ciò che ci aspettiamo è una modifica che rispetti la sintassi di AKN e che risulti in un documento Akoma Ntoso valido.
Effettueremo, tramite un'istanza dell'editor di metadati, delle modifiche ad un file AKN e analizzeremo i risultati per verificarne la correttezza sintattica.



(a) Tab *DOCUMENT* prima delle modifiche

(b) Tab *DOCUMENT* in seguito alle modifiche

Figura 5.1: Editor Simplex prima e dopo le modifiche

Come è possibile notare, sono stati modificati i dati relativi alla pubblicazione del documento, in particolare: *DATE*, *NAME* e *NUMBER*.
L'effetto sul documento AKN è il seguente:

Figura 5.2: Documento prima e dopo delle modifiche

Le modifiche effettuate, oltre ad avere una ripercussione istantanea sul file, rispettano la struttura dei file AKN e la loro sintassi. Quindi, l'editor di metadati non altera la struttura dei file AKN, rispettando lo standard.

Modifica del file di configurazione

In questa fase del test verificheremo come sia possibile creare delle nuove istanze dell'editor di metadati, semplicemente modificandone il file di configurazione. In questo modo, verificheremo l'efficienza dell'implementazione, in quanto avremo raggiunto il nostro *claim* iniziale, creando un sistema flessibile e facilmente adattabile a nuove esigenze.

Per creare una nuova istanza dell'editor di metadati ci basta modificare il file di configurazione. Per questo test proveremo a rimuovere delle *tabs* e modificarne delle altre, cambiando il modo in cui esse interagiscono con i metadati del documento.
Modifichiamo la descrizione dell'editor in questo modo, eliminando alcune *tabs*:

Listing 5.1: Modifica dell'entry point dell'editor di metadati

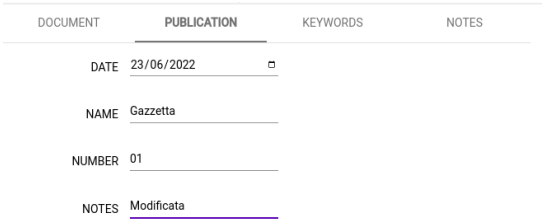
```
"@metadataEditor": {
    "@comment": "Metadata Editor Container",
    "label": "METADATA_EDITOR",
    "currentActive": "@metadataDocument",
    "items":  [
    {
        "type": "tabElement",
        "label": "DOCUMENT",
        "description": "@metadataDocument"
    },
    {
        "type": "tabElement",
        "label": "PUBLICATION",
        "description": "@metadataPublication"
    },
    {
        "type": "tabElement",
        "label": "CLASSIFICATION",
        "description": "@metadataClassification"
    },
    {
        "type": "tabElement",
        "label": "NOTES",
        "description": "@metadataNotes"
    }
  ]
},
```

Aggiungiamo un nuovo attributo nel *tab PUBLICATION* per aggiungere una nota sulla pubblicazione del documento.

Listing 5.2: Aggiungiamo un nuovo attributo del tag publication in items

```
"@metadataPublication": {
    "id": "publication",
    "type": "formContainer",
    "title": "PUBLICATION",
    "inline": "true",
    "@comment": "Metadata Publication Container",
    "tag": "publication",
    "items": [
    {
        "type": "datePicker",
        "label": "DATE",
        "description": "Document's date",
        "default": "today",
        "attribute": "date"
    },
    {
        "type": "textInput",
        "label": "NAME",
        "description": "Document's name",
        "default": "nameTest",
        "attribute": "name"
    },
    {
        "type": "textInput",
        "label": "NUMBER",
        "description": "Document's number",
        "default": "",
        "attribute": "number"
    },
    {
        "type": "textInput",
        "label": "NOTE",
        "description": "Publication's note",
        "default": "",
        "attribute": "note"
    }
  ]
}
```

Modificando la struttura dell'editor dall'apposito descrittore e la struttura della *tab PUBLICATION*, si ottiene il seguente risultato:



(a) Editor e *tab PUBLICATION* modificati

```
<meta>
    <publication
        date="2022-06-23" name="Gazetta"
        number="01" note="Modificata"/>
</meta>
```

(b) Metadati modificati tramite la nuova *tab*: è presente il nuovo attributo *note*

Figura 5.3: Ripercussione delle modifiche sull'editor

Come è possibile notare, bastano pochi semplici passi e una conoscenza quasi nulla dell'intera applicazione per modificare il file di configurazione creando delle nuove istanze dell'editor di metadati. Ciò permette un rapido adattamento del comportamento dell'editor alle esigenze dell'utente finale, garantendo una grande elasticità del codice.

Capitolo 6

Conclusioni e possibili sviluppi futuri

Lo scopo principale di questa tesi è stato quello di definire e realizzare un nuovo componente fondamentale di Simplex: l'editor per i metadati, assente nella attuale versione dell'editor.
Le caratteristiche principali dell'editor di metadati sono state implementate seguendo quelle che erano le funzionalità del precedente editor (LIME), cercando di ottenere, per Simplex, un editor per i metadati che fosse il più simile possibile, a livello funzionale, rispetto a quest'ultimo.
Il punto di forza di questa implementazione è la definizione di un file di configurazione che rende modulare l'editor, permettendo la sua modifica, sia a livello grafico che a livello funzionale, in pochi semplici passi. Infatti, è possibile generare diverse istanze di Simplex o del componente che gestisce l'editor di metadati, semplicemente modificando il file di configurazione alla base.
Inoltre, l'utilizzo di applicazioni basate su componenti permette, ad un futuro sviluppatore, di poter creare nuovi componenti per Simplex senza una conoscenza approfondita o totale dell'applicazione; infatti, per lo sviluppo del componente relativo all'editor di metadati, non è stato necessario comprendere l'intero editor.
L'editor di metadati è stato sviluppato con la consapevolezza che l'intera applicazione è ancora in fase di sviluppo e che potrebbero esserci dei cambiamenti per adattarlo alle

esigenze delle varie organizzazioni che utilizzeranno Simplex. Resta comunque un ottimo punto di partenza per continuare lo sviluppo dell'applicazione.
L'aspetto limitante di questa progettazione, basata sul file di configurazione, è sicuramente la struttura dei metadati nei documenti AKN: infatti, potrebbero essere presenti dei metadati complessi da descrivere e, di conseguenza, la descrizione dell'editor di metadati risulterebbe essere più difficile da comprendere e da realizzare. Inoltre, si potrebbero creare delle dipendenze tra la descrizione dell'editor tramite il file di configurazione e i metadati dei documenti AKN.
Una possibile modifica, che potrebbe migliorare la descrizione dell'editor di metadati, sarebbe quella di creare un'astrazione maggiore tra il file di configurazione e i metadati dei documenti, andando a realizzare un sistema in grado di generare il blocco di metadati solo in base alle informazioni fornite nel file di configurazione, separando maggiormente la descrizione dell'editor e i campi dei metadati che vengono influenzati.
Altre modifiche potrebbero avvenire a livello grafico, rendendo l'interfaccia grafica migliore di quella attuale. Durante lo sviluppo, infatti, è stato dedicato poco tempo alla creazione dell'interfaccia utente, ponendo l'aspetto grafico dell'editor in secondo piano, dando priorità alle funzionalità. Inoltre, si potrebbe rendere l'interfaccia utente accessibile per dare la possibilità di essere utilizzata da chiunque.

Bibliografia

[1] Gioele Barabucci et al. “Multi-layer markup and ontological structures in Akoma Ntoso”. In: *International Workshop on AI Approaches to the Complexity of Legal Systems*. Springer. 2009, pp. 133–149.

[2] Luca Cervone. “Parametric editors for structured documents”. In: (2013).

[3] CIRSFID. *About Lime*. [Online in data 16-06-2022]. URL: `http://lime.cirsfid.unibo.it/?page_id=50`.

[4] CIRSFID. *AKN Technical Specification V1. Guideline.* [Online in data 18-06-2022]. URL: `https://unsceb-hlcm.github.io/part1/`.

[5] Xcential Corporation. *A flexible and efficient legislative drafting solution.* [Online in data 29-06-2022]. URL: `https://xcential.com/legispro/technical/`.

[6] RADU Irina-Miruna. “Advantages and Challenges of Using Component–Based Software Development in the Vision of Building a Modern Educational System”. In.

[7] JoinUp. *LEOS - Open Source software for editing legislation.* [Online in data 29-06-2022]. URL: `https://joinup.ec.europa.eu/collection/justice-law-and-security/solution/leos-open-source-software-editing-legislation`.

[8] Bungeni Consulting LLP. *BungeniX-Ed - LEGISLATIVE DRAFTING EDITOR.* [Online in data 29-06-2022]. URL: `https://www.bungeni.com/legislative_drafting.html`.

[9] CIRSFID Monica Palmirani. “LEGISLATIVE XML”. In: (2012).

[10] Monica Palmirani e Fabio Vitali. “Akoma-Ntoso for legal documents”. In: *Legislative XML for the semantic Web*. Springer, 2011, pp. 75–100.

[11] Giovanni Sartor et al. *Legislative XML for the semantic web: principles, models, standards for document management*. Vol. 4. Springer Science & Business Media, 2011.

[12] W3C website. *W3C Introduction to XML*. [Online in data 16-06-2022]. URL: `https://www.w3.org/standards/xml/core`.