Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Corso di Laurea Magistrale in Informatica

Verso il supporto di miglioramenti iterativi dei documenti strutturati per testi giuridici

Relatore:
Chiar.mo Prof. Fabio Vitali

Presentata da:
Mattia Venturini

Correlatori:
Chiar.ma Prof.ssa Monica Palmirani
Dott. Luca Cervone

Anno Accademico 2017-18
Sessione II

"We are victims of our own success. We have let technology lead the way, pushing ever faster to newer, faster, and more powerful systems, with nary a moment to rest, contemplate, and to reflect upon why, how, and for whom all this energy has been expended."

— Donald Norman

Contenuti

Capitolo 1

Introduzione

Nel contesto dell'Informatica Giuridica, disciplina che mette assieme l'ambiente giuridico con le tecnologie dell'informazione, trovano particolare importanza gli standard di rappresentazione dei documenti legali in formato digitale, i quali specificano come memorizzare insieme contenuto e metadati per garantire diverse proprietà. Tra questi troviamo *Akoma Ntoso*, uno standard studiato appositamente per poter essere utilizzato in ogni contesto giuridico nel mondo, fornendo allo stesso tempo una visione unificata dei concetti e una buona flessibilità strutturale, oltre che preservare il valore semantico dei documenti nel tempo.

L'obiettivo di questa dissertazione è di porre l'attenzione sul processo di modifica e di perfezionamento dei documenti Akoma Ntoso, in cui si rivela importante rispettare lo standard in tutti i livelli di conformità al fine di garantire la massima interoperabilità dei documenti e degli strumenti che ne fanno utilizzo. In particolare cercherò di dimostrare la tesi secondo cui *disporre di un metodo per la modifica controllata dei file Akoma Ntoso favorisce lo sviluppo di strumenti informatici per la creazione di documenti di alta qualità*, cioè che rispettano tutti i principi dello standard.

La necessità di avere dati digitali per rappresentare ogni sorta di informazione del mondo reale trova la sua motivazione nel fatto che questi possono essere comodamente memorizzati, elaborati, distribuiti e visualizzati all'interno di un complesso ecosistema di dispositivi digitali, quasi sempre collegati

tra loro grazie alla vasta rete di comunicazione mondiale nota come Internet.

Non fanno eccezione a questo fenomeno i dati inerenti all'Informatica Giuridica. Essa ha il duplice scopo di supportare i processi legali con strumenti informatici e regolamentare con mezzi giuridici l'uso delle tecnologie dell'informazione. In questo ambito la forma più importante di dati è quella dei documenti legali, ovvero quei documenti creati, modificati o comunque emanati da un'entità giuridica autorevole. In passato tali documenti avevano come unica rappresentazione quella cartacea; sono stati convertiti in seguito in forma digitale per essere salvati su dispositivi di memorizzazione come i nastri magnetici, al fine ultimo di avere delle copie di backup su cui fosse possibile il recupero rapido dei testi richiesti; infine, a partire dallo sviluppo delle tecniche di Information Retrieval, il formato digitale ha acquisito sempre più importanza rispetto a quello fisico, fino a divenire il formato rappresentativo ufficiale del documento.

Un documento giuridico può essere rappresentato in vari modi, ma l'approccio più utile è quello dei documenti strutturati: si tratta di formati di rappresentazione che includono all'interno del documento stesso, oltre al contenuto, dei *metadati* utili all'elaborazione da parte di programmi software, che possono sfruttarli per compiere analisi e dedurre nuove informazioni.

La tecnologia più indicata per rappresentare documenti strutturati è XML, un linguaggio di markup creato come *open standard* su cui esistono numerosi strumenti che lo supportano e tecnologie che lo integrano con differenti funzionalità. Molti dialetti XML per documenti legali sono stati sviluppati negli ultimi anni, ma attualmente quello più completo è Akoma Ntoso. Per rendere possibile la sua diffusione sono necessari diversi strumenti specializzati per questo standard, da utilizzarsi nelle diverse fasi della vita di un documento:

1. la creazione, che avviene a partire dal markup di un file testuale o dalla conversione da un differente formato, solitamente come processo automatico da parte di uno strumento software;

2. la modifica ed il perfezionamento, come intervento successivo alla creazione, necessario per correggere le imprecisioni derivate da un convertitore imperfetto;

3. l'archiviazione, la pubblicazione ed il recupero, che devono essere coerenti con i nomi definiti sui documenti e trasparenti rispetto alla specifica architettura del sistema di memorizzazione;

4. l'analisi e la visualizzazione, in cui è utile disporre di tool in grado di astrarre sulla complessità dei documenti per ricavare le informazioni più significative per un certo utente.

Come indicato all'inizio, questa dissertazione si focalizza sul punto 2 della lista sopra riportata, con lo scopo di individuare un metodo per la modifica controllata dei file Akoma Ntoso, tenendo conto della correttezza semantica dei metadati, al fine di garantire la massima interoperabilità dei documenti e dei tool.

I risultati di tale studio si sono concretizzati nel mio progetto di tesi con la realizzazione di una libreria software, denominata *Akomando-Create*, in grado di fornire tramite API (Application Programming Interface) le funzioni per la verifica della correttezza dei metadati e la loro correzione in maniera controllata, cioè in modo da preservare la validità dei documenti. Questo strumento può essere utilizzato come colonna portante nella realizzazione di applicazioni concrete per gli uffici di redazione dei documenti legali, allo scopo di permettere il perfezionamento dei documenti esistenti con il minimo sforzo possibile da parte di un operatore esperto nel settore. Per mostrare l'utilizzo di Akomando-Create, ho progettato ed implementato una semplice demo in grado di interagire con una collezione di documenti, estraendone le informazioni utili riguardanti i metadati e la loro bontà.

Il progetto è stato realizzato all'interno del CIRSFID (*Centro Interdipartimentale di Ricerca in Storia del Diritto, Filosofia e Sociologia del Diritto e Informatica Giuridica*)[1], un centro interdisciplinare dell'Università di Bologna in cui sono presenti numerosi progetti di ricerca relativi all'Informatica Giuridica.

Nel corso della dissertazione mi occuperò di fornire il background tecnico e culturale per comprendere appieno il contesto in cui nasce la tesi, per poi

[1] http://www.cirsfid.unibo.it/

descrivere il lavoro da me svolto nel compimento del progetto e verificare come questo possa dare una risposta agli interrogativi evidenziati.

Nel capitolo 2 illustrerò il concetto di Data Analysis, mostrando come questa possa essere utile per trarre nuove informazioni e prendere decisioni; in particolare osserverò come questo processo è sempre più adoperato nel contesto della pubblica amministrazione e nell'ambito legislativo, al fine di venire incontro ai bisogni e agli scopi dell'Informatica Giuridica.

Nel capitolo 3 verrà approfondito il tema dei documenti strutturati, identificando le caratteristiche utili che si cercano nei formati di rappresentazione ed osservando come il linguaggio XML disponga di buone basi per tale scopo. Osserverò quindi le peculiarità di alcuni standard XML pensati per rappresentare documenti legali, come NormeInRete e MetaLex, per poi evidenziare i motivi che hanno portato alla scelta di Akoma Ntoso.

Nel capitolo 4 mi occuperò di descrivere gli obiettivi del progetto Akomando-Create, di evidenziare come questo sia stato pensato come plugin della libreria esistente *Akomando*, e di raccontare i punti più salienti relativi alla sua progettazione e realizzazione.

Nel capitolo 5 descriverò la realizzazione della demo, che ho denominato ANANAS (*Akoma Ntoso Analysis of Names And Semantics*), per mostrare un'applicazione concreta di Akomando-Create. Mi occuperò quindi di valutare il mio contributo scientifico ai fini della tesi evidenziata.

Infine nel capitolo 6 verranno tratte le conclusioni e considerazioni finali, accennando anche ai possibili sviluppi futuri per il progetto.

Capitolo 2

Importanza della Data Analysis per l'Informatica Giuridica

La società di oggi è caratterizzata dalla presenza di un'immensa quantità di dati, in particolare di dati digitali [Eco10]. Si trovano nei nostri portatili e nei nostri smartphone, nelle nostre auto e nelle lavatrici, per non parlare della vasta quantità di server appartenenti a qualche azienda, società o ente pubblico. Se analizzati con metodo ed ingegno, i dati portano con sé un enorme valore. Ma l'analisi dei dati non è qualcosa di nuovo: è un idea nata con l'avvento dei primi computer, che ha trovato la sua importanza con il passare del tempo e l'evolversi della tecnologia.

Questo capitolo inizia presentando il concetto di Data Analysis, allo scopo di comprendere cosa significa in ambito scientifico e analizzando i fattori che ne hanno favorito lo sviluppo. Successivamente si porterà l'attenzione su un particolare ambito di interesse, ovvero l'Informatica Giuridica, disciplina che mette assieme l'informatica e il mondo giuridico, osservando in particolare come la Data Analysis possa supportarla se applicata ai documenti legali.

2.1 Perché la Data Analysis oggi

Un importante riferimento alla Data Analysis risale al matematico ed esperto in statistica John Tukey, che nel 1962 la nomina come proprio interesse centrale e la delinea come: "Procedure per analizzare dati, tecniche per in-

terpretare i risultati di tali procedure, modi di pianificare la raccolta dati per renderne l'analisi più facile, più precisa o più accurata, e tutti i meccanismi e risultati della statistica che si applicano nell'analizzare dati." [Tuk62].

In letteratura troviamo molteplici definizioni di Data Analysis, con diverse sfumature e differenti relazioni con altri termini correlati, come Data Analytics, Data Science e Big Data. Una visione semplice e al contempo efficace si trova in [Ric06], che la definisce come un processo composto dalle seguenti fasi:

- **Purpose.** Descrizione di cosa si vuole fare e perché.
- **Questions.** Specifica di cosa si vuole sapere. Esistono diversi tipi di domande (a risposta chiusa o aperta), che richiedono diversi tipi di dati.
- **Data Collection.** Processo di raccolta dei dati necessari a rispondere alle domande.
- **Data Analysis Procedures and Methods.** Qui avviene l'analisi vera e propria, tramite metodi analitici, procedure e strumenti mutuati dal mondo della statistica, della matematica e dell'informatica. Questa può essere a sua volta suddivisa in sotto-discipline quali: Data Processing, Data Cleaning, Exploratory Data Analysis e Confirmatory Data Analysis.
- **Interpretation/Identification of Findings.** I risultati dell'analisi vengono osservati allo scopo di comprendere specifici fenomeni. A seconda dei casi possono essere necessarie determinate rappresentazioni: a volte il risultato è di per sé chiaro e non ha bisogno di ulteriori elaborazioni, mentre in altri casi è necessario mostrare graficamente tali risultati, usando appropriate tecniche di Data Visualization, o fornire ulteriori informazioni utili all'interpretazione.
- **Writing, Reporting, and Dissemination.** L'intera procedura svolta, assieme ai risultati ottenuti, viene descritta in modo logicamente comprensibile.

- **Evaluation.** Vengono valutate le precendenti fasi, al fine di individuare quali decisioni prese si sono dimostrate buone e quali no.

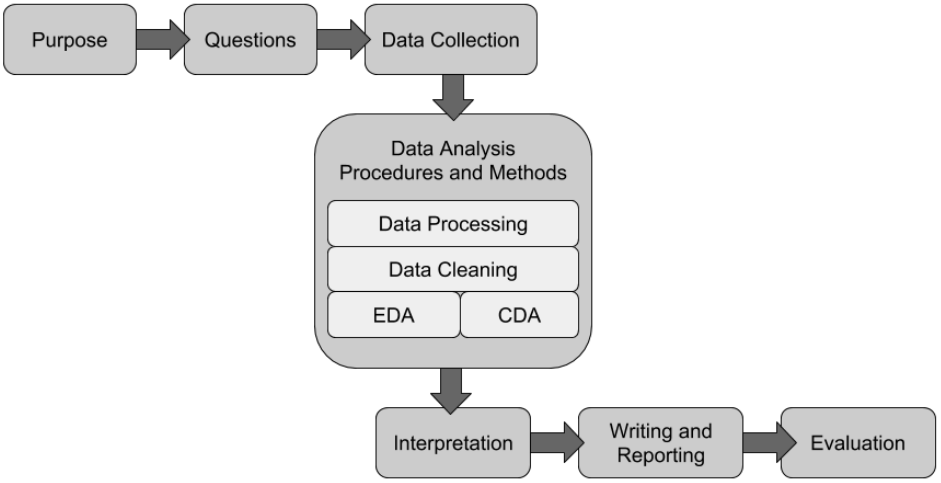

Figura 2.1: Il processo di Data Analysis.

Nella sezione seguente verranno descritte con maggior dettaglio le componenti indicate.

2.1.1 Il processo di Data Analysis

Abbiamo visto che il concetto di Data Analysis può essere formulato come processo che mira a dare risposte a delle domande, utilizzando come punto di partenza un insieme di dati, al fine di raggiungere un determinato obiettivo. All'interno di tale processo si distinguono diverse componenti, le quali contribuiscono allo scopo finale con diversi metodi.

Data Collection. La raccolta dei dati ai fini dell'analisi può avvenire da diverse sorgenti (database interni, dataset di terze parti, pubblici o a pagamento, interviste, sensori, ecc.). Si distinguono tre categorie di dati: strutturati, non strutturati e semi-strutturati [Mon18].

I dati strutturati sono in forma tabellare e provengono tipicamente dai database relazionali, ovvero la forma più tradizionale di gestione dei dati. Essi hanno uno schema rigido che si separa nettamente dal contenuto dei dati, garantiscono una serie di utili proprietà ai processi che li elaborano (le cosiddette proprietà ACID[1]) e vengono indicizzati in maniera efficente.

I dati non strutturati sono oggetti digitali che non forniscono una struttura fissa, come ad esempio i dati in forma testuale, le immagini, gli audio e i video; vengono elaborati con tecniche di Information Retrieval. A tale categoria appartengono i cosiddetti Big Data, che verranno trattati più avanti nella sezione 2.1.2.

I dati semi-strutturati si trovano nel mezzo tra le precedenti categorie, ovvero permettono di distinguere tra contenuto e meta-dati, ma lasciano una buona flessibilità a questi ultimi. Un esempio di tipo di dato semi-strutturato è XML (eXtensible Markup Language) che permette di definire schemi con cui è possibile vincolare i meta-dati e su cui esistono linguaggi di ricerca come XQuery. XML è approfondito in maggior dettaglio nel capitolo 3.2.

Data Processing e Data Cleaning. Queste due fasi si collocano prima dell'analisi vera e propria ed hanno come scopo il miglioramento della qualità dei dati [Wik18]. Il Data Processing consiste in una fase preliminare di elaborazione dei dati al fine di riportarli in un formato più utile alle tecnologie di analisi che si hanno a disposizione. Ad esempio si potrebbe processare un'insieme di dati semi-strutturati, ricavandone informazioni in forma tabellare su cui agire con maggior facilità.

Il Data Cleaning, invece, si occupa di migliorare la qualità complessiva dei dati individuando quelli incompleti o inaccurati, per poi correggerli o rimuoverli [Wik18a]. Questo serve per evitare che le analisi successive portino a conclusioni errate.

Exploratory e Confirmatory Data Analysis. È possibile classificare due tipi di analisi: l'Exploratory Data Analysis e la Confirmatory Data Analysis [Tuk80]. La prima ha lo scopo di comprendere al meglio il significato dei

[1] Atomicità, Coerenza, Isolamento e Durabilità

dati e le loro principali caratteristiche, per fornire allo studioso le idee necessarie a formulare ipotesi e individuare modelli statistici adeguati. Si tratta di un'attitudine che richiede una certa flessibilità, e non di un processo ben definito. La CDA, al contrario, parte da un'ipotesi precisa e utilizza i dati a disposizione per confermarla o confutarla.

Secondo Tukey [Tuk80], essi sono due approcci complementari, pertanto dovrebbero essere entrambi presenti in un processo di Data Analysis efficace.

Data Visualization. Si tratta della visualizzazione grafica dei dati ad un duplice scopo:

1. Metterne in evidenza determinate caratteristiche: in tal caso la Data Visualization risulta uno strumento utile ai fini della Exploratory Data Analysis;

2. Comunicarne il significato a persone terze: utile, e spesso fondamentale, nella fase di Reporting, quando si vogliono comunicare i risultati del processo di analisi.

In entrambi i casi la Data Visualization, per essere efficace, richiede una conoscenza minima nel campo della cognizione e della percezione visiva [Few13].

2.1.2 Data Analysis e Big Data

Big Data è probabilmente una delle parole chiave più usate nell'informatica degli ultimi anni, e deve il suo successo alla rapida crescita tecnologica che le ha permesso di diffondersi nel mondo delle aziente prima ancora di essere un concetto maturo a livello accademico [GH15].

In generale quando si parla di Big Data ci si riferisce ad insiemi di dati di dimensione e complessità tali da non essere possibile utilizzare i tradizionali strumenti software per raccoglierli, gestirli e processarli in un tempo utile [SMR12]. In [Lan01] viene raffinata la definizione caratterizzando il concetto con tre dimensioni: Volume, Velocità e Varietà. Successivamente alle "Three V" se ne sono aggiunte altre in letteratura (vedi [GH15]) e in totale si può trovare:

- Volume: la quantità di dati si misura solitamente in terabyte o petabyte;
- Velocità: i dati vengono generati molto rapidamente, in quanto possono provenire da una vasta quantità di sensori in un certo ambiente oppure da utenti all'interno di un social network o una piattaforma web; per tale motivo essi perdono utilità in fretta e devono essere analizzati velocemente, talvolta in tempo reale;
- Varietà: si riferisce alla forte eterogeneità dei dati, che richiede spesso tecniche di machine learning per analizzarli;
- Veridicità: si riferisce alla poca affidabilità di alcune sorgenti di dati, come ad esempio i sentimenti espressi dagli utenti su un certo social media, che, trattandosi di giudizi umani, sono per natura imprecisi;
- Variabilità: il flusso di dati può variare anche di molto, alternando cali e picchi;
- Valore: i dati presi in piccole parti hanno poco valore, ma per la loro quantità, una volta analizzati opportunamente, possono avere una enorme utilità.

Con la diffusione dei Big Data la Data Analysis si è evoluta introducendo nuove tecniche e nuovi strumenti di analisi, come il Cloud Computing su piattaforme distribuite quali Hadoop e Spark. In tal caso si parla si Big Data Analysis [Trn14].

2.1.3 Analisi di dati pubblici

Un altro fenomeno importante derivato dall'esposione tecnologica del Web 2.0, che questa volta riguarda direttamente il settore pubblico, è il passaggio da e-Governement a Open Government [HGB12], già avvenuto o comunque avviato in molti paesi del mondo. Mentre con l'e-government si intende generalmente l'uso delle nuove tecnologie dell'informazione per ottimizzare i processi delle Pubbliche Amministrazioni (PA), il focus dell'Open Government è sulla trasparenza e sulla partecipazione attiva dei cittadini al fine di

aumentare la fiducia verso le istituzioni pubbliche. Punto centrale di questo movimento, nonché centro di interesse per la Data Analysis, è la diffusione di portali contenenti i dati delle PA, detti Open Government Data, resi accessibili al pubblico in formato *open*, *machine-readable* e fornito di metadati [FP16].

Grazie a questi dati è possibile, sia da parte delle PA che di privati, fornire strumenti utili ai cittadini per i propri bisogni. Inoltre ha permesso ai governi di sviluppare un modo di lavorare più responsabile nei confronti dei cittadini, che di conseguenza possono sentirsi più partecipi e più fiduciosi verso le istituzioni.

Nella prossima sezione è descritto un esempio molto recente di piattaforma di Data Analysis orientata ai dati pubblici italiani.

2.1.4 Il Data Analytics Framework (DAF)

Il Data Analytics Framework (DAF) è uno strumento in fase di sviluppo da parte del Team per Trasformazione Digitale, in collaborazione con l'Agenzia per l'Italia Digitale, il cui obiettivo è di abbattere le barriere esistenti nell'inter-scambio dei dati pubblici tra le diverse PA ed ottimizzare i processi di analisi dati, al fine di valorizzare al meglio il patrimonio informativo pubblico dell'Italia [AT18].

Il DAF si compone di due elementi:

- Una piattaforma Big Data costituita da un *data lake* contenente diversi insiemi di dati generati o utilizzati dalle PA, nel rispetto della normativa sulla protezione dei dati sensibili; questi possono essere gestiti con diversi strumenti, tra cui delle API che li rendono fruibili ad applicazioni di terze parti.

- Un portale web, detto Dataportal, tramite cui è possibile accedere ed interfacciarsi alle funzionalità della piattaforma. È possibile creare, visualizzare e scaricare i dataset, analizzarli tramite strumenti di data analysis e data visualizzation e integrare diverse visualizzazioni all'interno di dashboard e storie.

Strumenti del Dataportal. All'interno del Dataportal sono al momento integrati i seguenti strumenti:

- *Metabase*: strumento che permette dei creare delle visualizzazioni grafiche a partire da dataset tabellari, sulla base di query (dette *question*).

- *Superset*: strumento open-source che permette di eseguire query SQL su dati tabellari, per poi visualizzare i risultati su grafici detti *slice*.

- *Jupyter*: permette di creare notebook dove alternare parti di testo e parti di codice, ed eseguire queste ultime in modo interattivo;

- *Widget*: permette di visualizzare i grafici creati con Metabase e Superset;

- *Storie* e *Dashboard*: sono due modi di creare documenti che includono testo e widget; le dashboard servono ad affiancare dati e grafici allo scopo di mettere in risalto certi risultati o caratteristiche, mentre le storie sono maggiormente orientare alla creazione di un documento discorsivo.

Utenti del DAF. Il DAF si rivolge principalmente alle PA, che possono usarlo per diversi scopi: visualizzare i dati presenti a supporto del *decision making*, sviluppare nuovi servizi per aziende e cittadini, usare applicazioni data-oriented per migliorare i propri processi interni e utilizzare i servizi del Dataportal per la pubblicazione dei propri Open Data. Oltre a questi, si aggiungono altre tipologie di utenti, come i *data journalist*[2], che utilizzano insiemi di dati per realizzare le proprie tesi giornalistiche, i cittadini, che possono accedere a storie e dashboard già realizzate per informarsi su un determinato argomento, le aziende e community, che hanno a disposizione strumenti e dati per creare applicazioni e servizi, e i ricercatori, che possono essere coinvolti su tematiche di interesse pubblico.

[2]Per approfondire vedi Data Jouralism - Wikipedia: https://en.wikipedia.org/wiki/Data_journalism

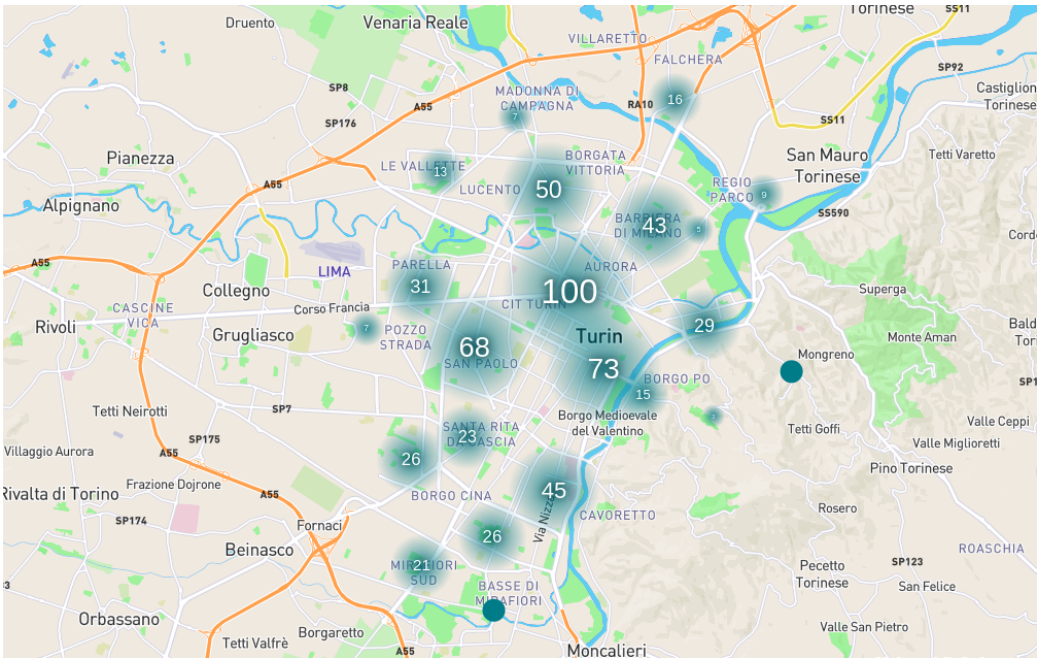

Figura 2.2: Esempio di visualizzazione ottenuta sul Dataportal utilizzando Superset. Raffigura la mappa di Torino con il numero di scuole presenti in ogni quartiere.

2.2 Data Analysis ed Informatica Giuridica

L'informatica giuridica è l'area interdisciplinare che mette assieme l'informatica con il mondo giuridico e legislativo [Sar16]. Essa si divide a sua volta in due macro-aree: il Diritto dell'Informatica, che si occupa di regolamentare le tecnologie e i prodotti dell'informatica, e l'Informatica del diritto, che applica le scienze informatiche ai contesti giuridici per facilitare il lavoro del giurista.

Strumenti di Data Analysis possono essere utilizzati nell'Informatica del diritto, come si avrà modo di vedere più dettagliatamente nella sezione 2.2.2, che ne riporta alcuni esempi. In 2.2.1 viene invece approfondita maggiormente la definizione di questa disciplina.

2.2.1 In cosa consiste l'Informatica Giuridica

Come riportato in [Sar16], l'Informatica giuridica si divide in due macro-aree.

Diritto dell'Informatica. In questa area vengono regolamentati, ad esempio, i temi della proprietà intellettuale informatica, della tutela dei dati, dei documenti digitali, dell'e-commerce e dei reati informatici. Le azioni giuridiche in quest'ambito si possono distinguere in azioni *ex-post* ed *ex-ante*: le prime servono a regolamentare un innovazione tecnologica dopo che questa ha cominciato ad avere un impatto sulla società (es. regolazione dei nomi di dominio sul web); le altre invece portano conseguenze sulla società che si sviluppano in seguito come innovazioni nel mondo tecnologica (es. invio telematico dei bilanci alle camere di commercio).

Informatica del Diritto. In quest'area appartengono i sistemi informativi sviluppati in supporto alle attività di giuristi, come quelli per la redazione e ricerca di documenti legali digitali e i modelli di *legal reasoning*.[3] Storicamente risulta essere suddivisa nelle seguenti fasi:

- Previsionale: nata negli anni '50 nei paesi del common law, avendo come obiettivo la creazione di applicazioni per la previsione delle decisioni giudiziarie;
- Documentaria: nata negli anni '60 con i primi database di documenti legali, ripresa in seguito con lo sviluppo delle tecniche di Information Retrieval;
- Logico-Decisionale: nata negli anni '70 per lo sviluppo di sistemi logici per la rappresentazione delle leggi e la formalizzazione del ragionamento giuridico;
- Gestionale: legata all'avvento dei Personal Computer e delle reti locali di calcolatori negli anni '80, si occupa della creazione di applicazioni software per automatizzare le procedure d'ufficio del mondo giuridico.

[3] Meccanismi di ragionamento che portano giudici, corti ed avvocati a prendere decisioni, tenendo conto della legislatura, dei casi precedenti e di ogni dettaglio significativo del caso in questione.

- Comunicativa: legata alla diffusione di Internet e all'esplosione tecnologica iniziata negli anni 2000, che ha spinto le pubbliche amministrazioni a creare servizi web per ridurre la distanza tra i cittadini e le organizzazioni nella realizzazione dell'e-government [BFP08].

Al fine di comprendere meglio come far intersecare l'Informatica Giuridica alla Data Analysis, osserveremo, nel corso della sezione seguente, alcuni progetti esistenti in cui è stata svolta analisi nel contesto giuridico, individuando le tecnologie utili a tale scopo.

2.2.2 Esempi di Data Analysis nel contesto giuridico

In seguito sono riportati alcuni progetti che si sono occupati di Data Analysis su dati giuridici.

Code4Italy.[4] Si tratta di un progetto svolto presso il CIRSFID con lo scopo di visualizzare graficamente alcuni elementi significativi del procedimento legislativo del Parlamento italiano. I dati erano presenti in due endpoint SPARQL, appartenenti rispettivamente alla Camera e al Senato, e sono stati reperiti con query, esportati in differenti formati e raffinati con alcuni script, per essere poi visualizzati graficamente.

Socievole.[5] Un altro progetto svolto al CIRSFID, questa volta avente come oggetto gli atti pubblici disponibili sul portale ufficiale del Regno Unito.[6] I documenti in questione erano in formato Akoma Ntoso e sono stati processati usando Python per estrarne titoli e parole chiave, al fine di mostrare come gli argomenti delle leggi sono variati nel tempo.

Altri progetti di Data Visualization sulle leggi. Fra gli altri progetti interessanti possiamo trovare l'iniziativa francese La Fabrique de la Loi[7], che visualizza su una timeline gli eventi relativi al ciclo di vita di ogni proposta

[4]Code4Italy Project's site: http://code4italy.cirsfid.unibo.it/
[5]Socievole: http://sinatra.cirsfid.unibo.it/socievole
[6]http://www.legislation.gov.uk/
[7]https://www.lafabriquedelaloi.fr/

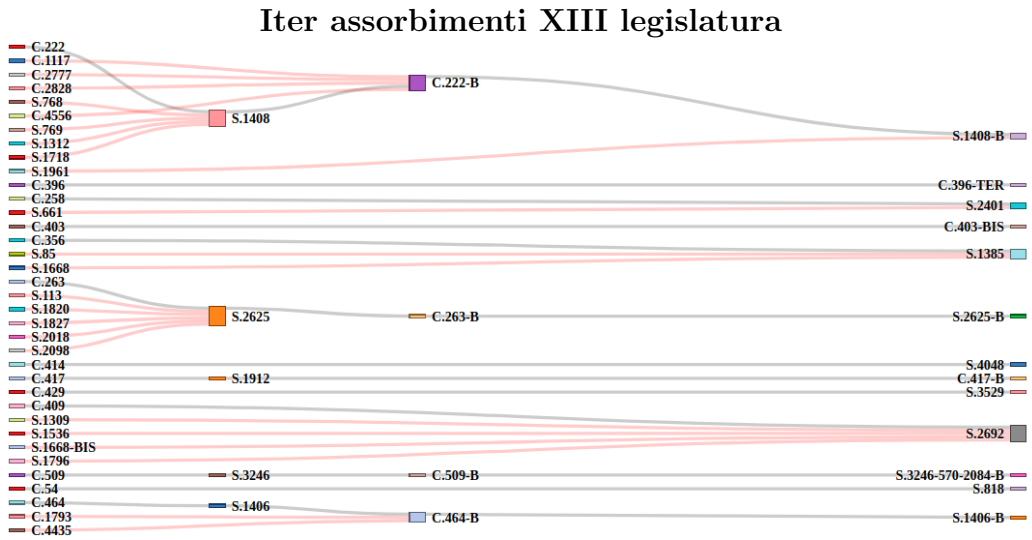

Figura 2.3: Una delle visualizzazioni realizzate nel progetto Code4Italy. Rappresenta graficamente le relazioni tra i diversi progetti di legge e l'atto principale che li ha accorpati

di legge, estraendo i dati da un catalogo Open Data in CSV. L'idea di questa visualizzazione è ispirata al piccolo progetto del tedesco Gregor Aisch, denominato "The Making of a Law"[8], il quale illustrava graficamente l'evoluzione della legge tedesca sui partiti politici dal 1967 ad oggi.

Strumenti di ricerca di leggi. Sempre più paesi dispongono di portali online che permettono l'accesso alla legislazione in formato digitale, spesso in formati machine-readable. Questo ha permesso lo sviluppo di applicazioni di ricerca di leggi in base al tema, parole chiave ed altri metadati, come ad esempio il tool canadese Kromos[9] e quelli industriali americani Fastcase[10] e Ravel Law[11].

[8] http://visualisiert.net/parteiengesetz/index.en.html
[9] http://knomos.law/
[10] https://www.fastcase.com/
[11] https://home.ravellaw.com/products

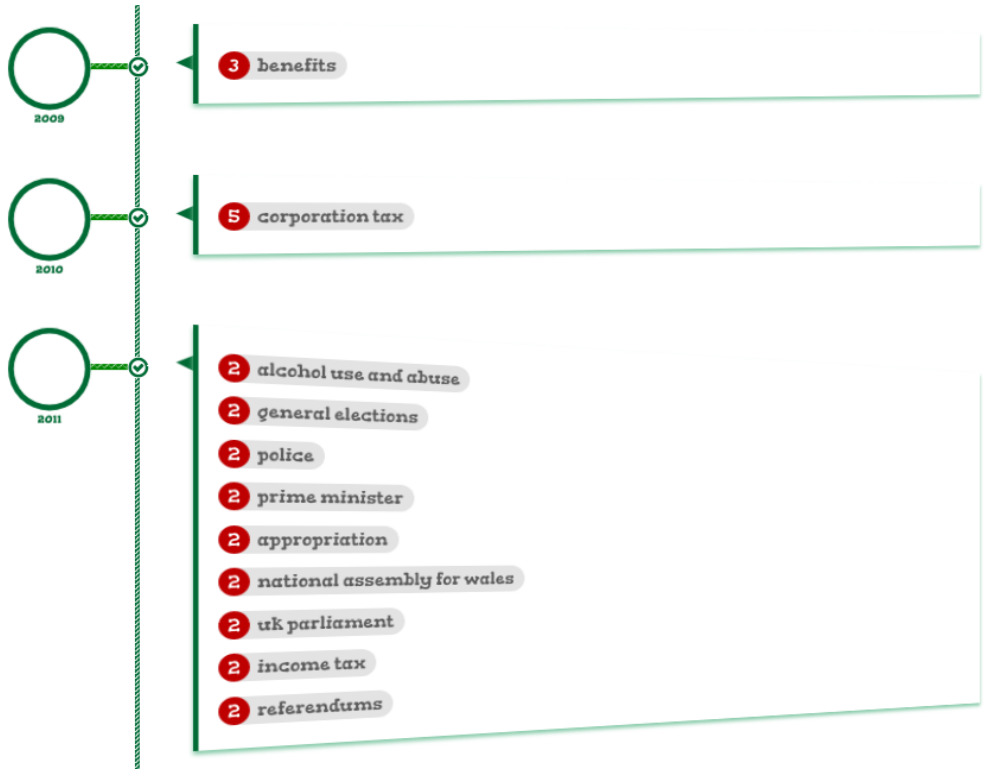

Figura 2.4: Screenshot di una parte della timeline di Socievole.

Network Analysis sulle leggi. Consiste nell'analizzare e visualizzare graficamente le numerose relazioni tra le diverse leggi di un sistema giuridico, ovveri quali di esse ne citano, modificano o abrogano altre. Esistono diversi lavori di questo tipo in ambito accademico. In [GV13] viene realizzato un parser per costruire documenti XML a partire da file testuali e da questi vengono costruiti e visualizzati i collegamenti tra documenti. In [Hoe13] invece si utilizza un dataset disponibile in duplice formato MetaLex[12] e RDF[13]; quest'ultimo risulta particolarmente efficace nella costruzione di grafi, in quando già di per sé rappresenta un grafo tra risorse, ed infatti sono sufficienti delle query SPARQL piuttosto semplici per raggiungere lo scopo.

[12]Un dialetto XML pensato per lo scambio di documenti legali, approfondito in "Standard XML per documenti legali" (sezione 3.2.3).

[13]Resource Description Framework, un formato per la rappresentazione della conoscenza, utilizzato nel Semantic Web.

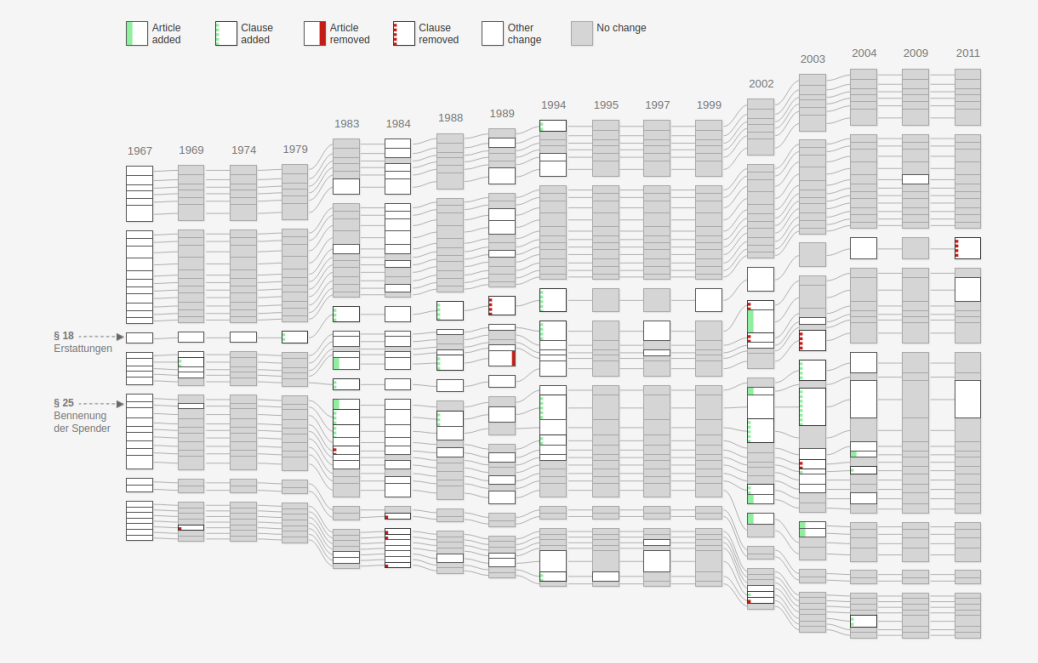

Figura 2.5: Screenshot del grafico mostrato in "Making of a Law" di Gregor Aisch.

2.2.3 Strutturare documenti giuridici per favorirne l'analisi

Come riportato nella sezione precedente, molti lavori di Data Analysis nel contesto giuridico hanno come oggetto dell'analisi le leggi o altri tipi di documento legale. Questi sono documenti testuali pensati per essere scritti e letti da esseri umani, e solo in seguito adattati per essere *machine-readable*; essi devono essere strutturati in modo che i metadati possano essere compresi e processati efficacemente da un programma software.

Nel prossimo capitolo si osserverà in maggior dettaglio quali sono i *desiderata* di un formato digitale per documenti legali e come l'XML sia il linguaggio più indicato a rappresentare tali documenti strutturati.

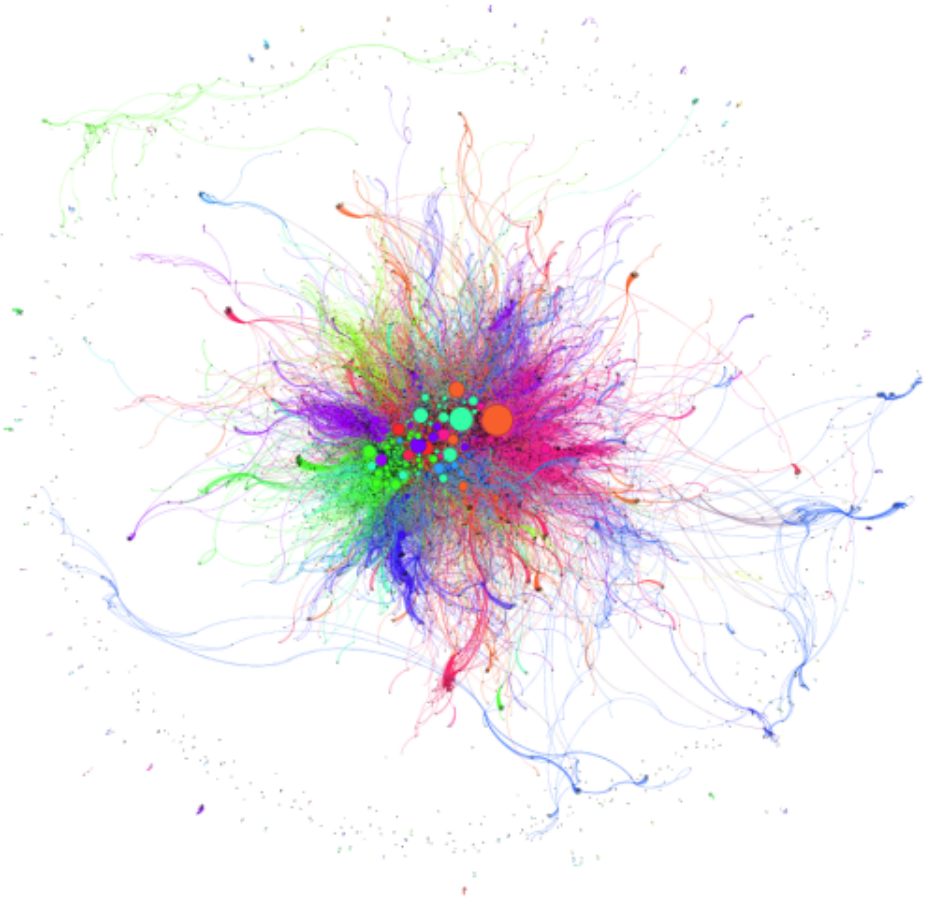

Figura 2.6: Grafo delle citazioni ottenuto in [Hoe13], riguardante la normativa tedesca.

Capitolo 3

Strutturazione di documenti giuridici

Quando si trattano documenti in formato digitale bisogna riconoscere l'importanza che hanno i suoi metadati, ovvero tutte le informazioni che non fanno strettamente parte del contenuto e che servono a dare un particolare significato al documento o ad una sua parte. Si parla di *documenti strutturati* quando i metadati si trovano all'interno del documento stesso, rappresentati usando un qualche linguaggio di markup. Possiamo distinguere tre categorie generali di markup [CRD87]:

- **Presentational markup.** Si tratta del tipo di markup usato tradizionalmente dai word processor. Inserisce all'interno del documento le opzioni di formattazione e visualizzazione in codice binario, come ad esempio la spaziatura, il colore, il tipo di carattere, ecc.; esse vengono immediatamente visualizzate in modo che siano, di solito, completamente trasparenti all'autore, dando l'effetto *WYSIWYG* ("What You See Is What You Get").

- **Procedural markup.** Inserisce all'interno del documento delle istruzioni; per ottenere la visualizzazione il sistema scorre il testo e le esegue nel momento in cui le incontra. Permette inoltre all'utente di definire macro e subroutine per operazioni frequenti.

- **Descriptive markup.** Questa forma di markup si occupa di etichettare parti del documento per dar loro un significato, piuttosto che instruzioni di processing. In questo modo si separa la struttura logica dallo stile di visualizzazione, che viene implementato dal un programma di formattazione tramite una serie di regole. Così è possibile modificare lo stile di presentazione del documento senza modificarne la struttura logica.

Il descriptive markup offre numerosi vantaggi nell'ambito dei documenti giuridici, tra cui la maggiore portabilità e manutenibilità, oltre che permettere analisi sulla logica degli stessi, cosa impossibile con gli altri tipi di markup. Questo ha permesso l'affermarsi di XML, che rappresenta markup descrittivo, come principale linguaggio per i documenti giuridici in formato digitale.

Questo capitolo si articola come segue: in 3.1 verranno approfondite le caratteristiche necessarie nei documenti giuridici, per capire quali qualità deve avere uno standard per documenti strutturati; in 3.2 è descritto in maggior dettaglio XML, riportando le tecnologie ad esso legate e i dialetti esistenti per l'informatica giuridica; infine in 3.3 si osserverà nel dettaglio lo standard Akoma Ntoso, un dialetto XML pensato per essere un possibile standard universale.

3.1 Documeni giuridici in formato digitale: caratteristiche e desiderata

I documenti giuridici sono quei testi emanati da un'entità autorevole di natura giuridica, come ad esempio un Parlamento, una Corte o un Tribunale. In generale tali testi compiono almeno una delle seguenti funzionalità: evidenziano un evento legalmente valido, impostano una regola, garantiscono un diritto, registrano una procedura legale, forniscono una decisione legale rendendola accessibile o riportano una comunicazione ufficiale tra due parti [Pal12]. Leggi, sentenze, emendamenti e report di dibattiti parlamentari sono quindi possibili esempi di documento giuridico. Essi vengono visti e manipolati da diversi attori, seguendo precise procedure legali spesso lunghe

e articolate svolte da organi competenti, e una volta approvati da un'ente autorevole devono rimanere immutati nel contenuto (anche nel caso di emendamenti, che sono testi legali rappresentanti la modifica di documenti preesistenti e non sovrascrivono fisicamente il testo precedente). In seguito sono individuati i principali *desiderata* di uno standard per documenti strutturati:

- Deve trattarsi di un formato aperto, in modo da essere accessibile ad ogni parte interessata alla creazione o visione del documento, anche da parte di persone con disabilità;
- Deve supportare la gestione di un documento all'interno del suo *workflow*, ovvero ogni fase di creazione e modifica attuata da organi competenti deve essere facilmente tracciabile;
- Deve favorire la stesura di documenti di qualità: ogni tipo di documento, infatti, possiede delle regole di stesura, ufficiali o non ufficiali, che spesso variano notevolmente tra diversi paesi; un buono standard deve permettere di verificare l'aderenza dei documenti a tali regole;
- Deve essere facile nel suo utilizzo, semplificando le operazioni ripetitive;
- Deve essere possibile utilizzarlo su diverse piattaforme e al variare delle tecnologie; in particolare non deve dipendere da stardard o strumenti proprietari;
- Deve essere sufficientemente preciso da dare una buona semantica al contenuto, ma allo stesso tempo abbastanza flessibile per essere usato da diversi sistemi giuridici con diverse tradizioni;
- Deve distinguere chiaramente il testo approvato dall'ente autorevole dai metadati aggiunti in seguito, al momento della stesura o della pubblicazione.

3.2 L'eXtensible Markup Language (XML)

XML sta per *eXtensible Markup Language* [BPS97]. Si tratta di un metalinguaggio per la definizione di linguaggi di markup, derivato da SGML (*Stan-*

dard Generalized Markup Language) e divenuto standard del W3C (*World Wide Web Consortium*). XML, come SGML, vanta di essere sia *human-readable* che *machine-readable*. I metadati vengono rappresentati da tag che racchiudono al loro interno parti di testo e/o altri tag, formando una struttura ad albero utile per rappresentare sia documenti strutturati che insiemi di dati semi-strutturati. Inoltre ogni tag può possedere degli attributi caratterizzati da un nome e da un valore testuale. Mentre l'HTML (altro discendente di SGML) ha uno scopo specifico, quello di rappresentare pagine Web e, più generalmente, ipertesi, l'XML permette di definire sottolinguaggi specifici per certi contesti; essi vengono rappresentati attraverso file DTD (Document Type Definition) o XSD (XML Schema Definition), che indicano: quali sono i tag ammessi, in quale ordine e in quale gerarchia, quali attributi possono o devono contenere, ed altri vincoli sintattici.

Un documento XML può rispettare diversi livelli di conformità rispetto ad uno standard: (0) esso si dice ben formato quando i tag rispettano la struttura ad albero, cioè vi è una sola radice e i tag non si sovrappongono; (1) si dice valido se rispetta il DTD o schema XML che lo definisce; (2) possono esserci ulteriori vincoli e convenzioni indicati dallo standard ma non definibili nello schema, come ad esempio la sintassi dei valori contenuti negli attributi.

3.2.1 Strumenti e tecnologie per la gestione di dati in XML

La diffusione di XML ha portato alla nascita di diversi strumenti di gestione, manipolazione e interrogazione di tale formato. In seguito ne sono riportati alcuni:

- XSLT (eXtensible Stylesheet Language Transformation)[1] è un linguaggio che describe come trasformare documenti XML in altri documenti, che possono essere a loro volta XML, HTML, testo semplice o altri formati.

[1] https://www.w3.org/TR/xslt-10/

- XPath[2] è un linguaggio di query che serve ad estrarre nodi o valori all'interno di un documento XML specificando un percorso di ricerca come insieme di step. Ad ogni step si procede attraverso un asse, cioè un insieme di nodi che hanno una certa relazione con quello corrente (di default è *child*, altri assi sono ad esempio *descendant* o *parent*), che vengono selezionati in base a nome del tag, valori degli attributi o predicati specifici.

- XQuery[3] è un linguaggio concepito come linguaggio di query per collezioni di dati XML, ma divenuto di fatto un linguaggio funzionale completo in grado di elaborare altri formati, come ad esempio JSON.

Tali strumenti hanno contribuito ulteriormente alla sua popolarità.

3.2.2 Markup di documenti XML

Il processo di creazione di un documento strutturato XML avviene utilizzando almeno uno dei seguenti strumenti: converter ed editor [BFP08].

Converter. Il converter ha la funzione fondamentare di convertire nello standard XML di interesse i documenti già esistenti nel formato precedente. È uno strumento fondamentale per l'affermazione di un nuovo standard non solo per questi documenti *legacy*, ma per convincere l'ente giuridico in questione che i vantaggi offerti dallo standard superano notevolmente i costi necessari per questo passaggio. La conversione è da considerare come un processo semi-automatico, in quanto il converter non è completamente esente da errori; esso deve analizzare le regolarità testuali e tipografiche del documento, per dedurre il maggior numero possibile di informazioni semantiche, e lasciare ciò che è impreciso o non deducibile ad un correttore umano.

Editor. Un altro strumento fondamentale è l'editor, che permette ad un operatore umano di modificare un documento strutturato, in genere all'interno di uno di questi scenari:

[2]http://www.w3.org/TR/xpath10
[3]http://www.w3.org/XML/Query

- Come interfaccia per controllare, verificare ed eventualmente correggere il risultato della generazione automatica di un documento da parte del converter;

- Come strumento per marcare manualmente un documento disponibile in un altro formato;

- Come applicazione per la creazione di un nuovo documento, in cui inserire contemporaneamente testo e markup.

Gli editor per XML possono essere distinti in due categorie: editor generici e editor di markup [Cer13]. I primi sono indipendenti dallo standard utilizzato e permettono di agire direttamente sui tag XML, per poi verificare se sono ben formati ed eventualmente se sono validi rispetto ad un certo schema; il loro uso richiede una certa competenza tecnica sul linguaggio XML. Gli editor di markup, invece, mostrano i documenti secondo uno stile grafico simile al risultato finale, nascondendo all'utente i tag XML. Da una parte si comportano quindi come i word processor classici, mentre dall'altra permettono la creazione di soli documenti validi per lo schema di interesse, impedendo da interfaccia utente l'inserimento di elementi non validi. Un esempio di editor di markup è LIME (*Language Independent Markup Editor*), che, essendo parametrico, può essere utilizzato per diversi standard XML.

3.2.3 Standard XML per documenti legali

Come già anticipato, il linguaggio XML risulta essere particolarmente adatto per la rappresentazione di documenti giuridici [Pal12]. Esso soddisfa già da sé alcuni dei requisiti descritti nella sezione 3.1 ("Documeni giuridici in formato digitale: caratteristiche e desiderata"): si tratta di un formato aperto e dispone di diversi strumenti e tecnologie utili alla sua gestione, pur restandone indipendente. Altri requisiti invece devono essere ricercati negli stardard XML.

Esistono diversi standard applicati ai documenti legislativi, ma pochi possiedono tutte le caratteristiche ricercate. Alcuni non distinguono la rappresentazione logica da quella estetica; altri sono specifici per un certo pae-

se e non si adattano ad altri contesti giuridici. A seconda delle diverse caratteristiche, possiamo distinguere quattro classi di standard:

I La prima generazione, in cui troviamo ad esempio Formex, descrive la struttura dei testi legali in modo simile ai modelli dei word processor e dei database tradizionali.

II La seconda generazione comprende quegli standard che descrivono metadati, struttura e testo, senza però aver fatto un'analisi completa sull'astrazione da usare, causando la creazione di numerosi tag con regole complicate e talvolta persino sovrapposte; un esempio di questo tipo è lo standard italiano NIR.

III La terza generazione si basa sui pattern. Un pattern rappresenta le proprietà di una classe di elementi, fornendo un content-model, un comportamento ed una gerarchia rispetto alle altre classi. Ogni tag quindi, compresi eventuali tag personalizzati aggiunti in seguito, appartiene ad un pattern ed ha delle regole chiare e ben definite. Standard di questo tipo sono Akoma Ntoso e MetaLex.

IV La quarta generazione combina i pattern a grammatiche co-constraint per fornire vincoli prescrittivi, su cui la terza generazione è carente. Esempi di questa categoria sono RELEX NG e Schematron.

In seguito sono riportati alcuni standard ritenuti particolarmente significativi: NIR, un formato diffuso nel contesto italiano, MetaLex, ideato come formato di scambio, ed Akoma Ntoso.

Norme In Rete. Norme In Rete (NIR) è uno standard XML italiano per documenti normativi [BFST03], realizzato con lo scopo finale di creare un unico punto di accesso sul Web per il recupero delle leggi. Il lavoro, coordinato dall'Agenzia per l'Italia Digitale, ha portato alla definizione di due standard:

1. Uno standard URN (Uniform Resource Name) che stabilisce come associare ad ogni documento legale un identificativo univoco. Questi nomi hanno il duplice scopo di garantire univocità e referenziabilità ad ogni

documento legale; non sarebbe possibile infatti utilizzare come indentificatore gli URL, in quanto: (1) la locazione esatta potrebbe essere sconosciuta, (2) la locazione potrebbe cambiare nel tempo, (3) il documento potrebbe non avere una locazione accessibile in quanto non ancora pubblicato. Per usare gli URN in un sistema di gestione di documenti NIR è necessario un meccanismo di risoluzione di nomi, detto RDS (Resolver Discovery Service).

2. Uno schema per la definizione documenti XML, costituito da tre DTD con crescente livello di profondità:

   I. Un DTD flessibile (*nirloose*), non contiene particolari regole ed è usato per vecchi documenti che non seguivano le linee guida di formattazione;

   II. Un DTD base (*nirlight*), contiene un sottoinsieme dei tag totali ed è utile per la formazione degli utenti;

   III. Un DTD completo (*nirstrict*), contiene tutti i tag e le regole di formattazione delle norme.

In seguito a questi standard, il progetto ha portato allo sviluppo di un intero ecosistema di strumenti e applicazioni per le gestione di documenti NIR. Un esempio è Norma-System [PB07], un sistema in grado di gestire il documento legale per l'intero processo di creazione, supportando attività di front-end, attraverso un editor specializzato detto Norma-Editor, e di back-end, attraverso un portale detto Norma-Server. Norma-Editor è un editor eseguito su Microsoft Word che supporta la creazione del documento XML con operazioni di scrittura, markup, validazione e semantic-check. Norma-Server invece consente di memorizzare i documenti su un database, permettendo di gestire metadati, versioni e di effettuare operazioni di ricerca e di navigazione all'interno della collezione.

Il difetto di NIR rimane tuttavia evidente: si tratta di uno standard pensato per la legislazione italiana, pertando non può godere dei benefici dati dall'interoperabilità con altri sistemi giuridici.

MetaLex. MetaLex è uno standard XML per documenti legali generico, estensibile e indipendente da giurisdizioni, progettato non come formato di markup delle leggi originali, ma come formato di scambio tra enti che usano diversi standard [BWV08]. Lo schema di MetaLex cerca di imporre una visione standardizzata sui diversi formati esistenti, traendo le migliori pratiche dai precedenti standard, come ad esempio la netta distinzione tra contenuto e metadati.

MetaLex si basa sul concetto di *content model*: anziché dare una semantica ai nomi dei tag, gli elementi descrivono un ruolo indicando il relativo content model tramite l'attributo *type* e un nome semantico tramite l'attributo *name*. Per comprendere meglio, osserviamo che ogni elemento descritto nel codice seguente risulta equivalente per lo standard.

```
<clause metalex:type="metalex:container" metalex:name="
    clause"/>
<clause metalex:type="metalex:container"/>
<metalex:container metalex:type="metalex:container"
    metalex:name="clause"/>
<metalex:container metalex:name="clause"/>
```

Un documento si dice conforme a MetaLex se: (1) risulta valido rispetto ad uno schema XML che include lo schema MetaLex, (2) è possibile, tramite sole sostituzioni semplici; ottenere un documento XML che usa solo tag gererici MetaLex ed è valido rispetto al relativo schema; (3) è conforme alle linee guida di MetaLex, le quali non si possono formalizzare all'interno dell'XSD.

Akoma Ntoso. Akoma Ntoso è uno standard XML nato in un progetto dell'*United Nations Department for Economics and Social Affairs* per incrementare l'efficienza degli organismi legislativi di alcuni paesi dell'Africa nello svolgimento delle proprie funzioni, con l'aiuto dell'ICT. Al momento è nella versione 3 ed è diventato uno standard OASIS con il nome di *LegalDocML*.

È rappresentato formalmente da uno schema XML e da una Naming Convention che stabilisce come assegnare identificatori ai documenti e agli elementi interni ad essi. Si dice che un documento ha un livello di conformità 1 se è valido rispetto allo schema e 2 se è conforme alla Naming Convention.

A differenza di MetaLex, che cerca di essere un formato per lo scambio di documenti lasciando ad ogni giurisdizione l'utilizzo dello standard preesistente, Akoma Ntoso è pensato per essere al contempo preciso nella semantica ed estremamente flessibile, per essere in principio utilizzabile in ogni contesto giuridico possibile; per questo motivo è oggetto di numerosi progetti presso il CIRSFID.

Akoma Ntoso è descritto in maggior dettaglio nella sezione 3.3, dove vengono descritti i principi progettuali e le caratteristiche di questo standard.

3.3 Lo standard Akoma Ntoso

Tra tutti gli standard XML esistenti per documenti giuridici, Akoma Ntoso[4] è progettato specificamente per essere un possibile standard universale, in grado di adattarsi ad ogni sistema giuridico di ogni paese. Il suo nome, che significa "cuori uniti" nella lingua africana Akan, è un acronimo per *Architecture for Knowledge-Oriented Management of African Normative Texts using Open Standard and Ontology.*

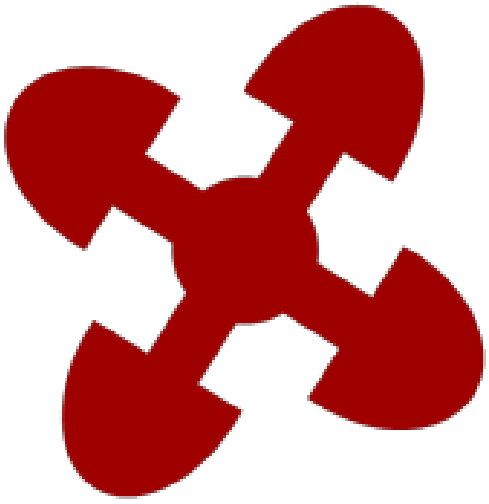

Figura 3.1: Il logo di Akoma Ntoso è il simbolo usato dalle popolazioni Akan dell'Africa occidentale per rappresentare comprensione e accordo.

[4]Akoma Ntoso — Akoma Ntoso Site: http://www.akomantoso.org/

Nella prossima sezione viene motivata l'esigenza di uno standard universale, per poi osservare come Akoma Ntoso cerca di raggiungere tale obiettivo attraverso: una struttura basata su pattern, che vedremo in 3.3.2, ed una naming convention, descritta brevemente in 3.3.3.

3.3.1 Vantaggi di uno standard universale

L'obiettivo ambizioso di creare uno standard universale trova la sua giustificazione nei numerosi vantaggi che si avrebbero per le diverse categorie di utenti, come spiegato più dettagliatamente in [BFP08] (Capitolo 4).

Gli utenti che avranno a che fare con lo standard saranno:

1. il "legislatore", ovvero un membro di un ente legislativo o un suo assistente, il quale non è a conoscenza dello standard (e neanche gli interessa), ma vuole avere degli strumenti facili ed efficaci per ricercare, navigare e visualizzare i documenti al fine di compiere il suo dovere in maniera efficiente;

2. il "redattore legale", ovvero un membro dell'ufficio incaricato della stesura del documento in formato digitale; egli è un esperto nel settore giuridico, ha alcune competenze informatiche, ma conosce lo standard in maniera superficiale;

3. il "toolmaker" è colui che ha il compito di realizzare software specializzato per l'ente giuridico, in particolare per il redattore legale; ha alte competenze tecniche e può accedere alla documentazione ufficiale dello standard;

4. il "cittadino" potrebbe essere un avvocato, un giudice, un impiegato pubblico o un qualunque individuo che ha bisogno di accedere ad un testo legale; non sa cos'è XML e non ha particolari competenze informatiche, ha solamente bisogno di uno strumento di navigazione facile e veloce da usare;

5. il "futuro toolmaker" è colui che dovrà sviluppare strumenti per lo standard dopo parecchi anni, in cui lo stesso può essersi evoluto e la

documentazione potrebbe non essere altrettanto chiara e completa come lo era per il toolmaker; il futuro toolmaker, quindi, ha bisogno di uno standard chiaro e auto-esplicativo, in modo da comprenderne i principi e il funzionamento dagli esempi esistenti.

Uno standard universale porterebbe numerosi vantaggi ad ognuno di questi utenti: il toolmaker si occuperebbe semplicemente di indivuare ed eventualmente riadattare software già sviluppato o in via di sviluppo presso altri paesi, piuttosto che reinventare la ruota, risparmiando tempo e risorse; il redattore legale potrebbe riutilizzare strumenti di creazione già adottati in altri paesi; il cittadino e il legislatore potrebbero godere di strumenti di ricerca e visualizzazione più stabili e disponibili in minor tempo, oltre che accedere con gli stessi a documenti di altre giurisdizioni; il futuro toolmaker, infine, avrebbe a disposizione una quantità di materiale maggiore su cui basare il proprio lavoro. Inoltre si garantirebbe la possibilità di progetti transnazionali.

Interoperabilità è quindi la parola chiave. Interoperabilità semantica si ha dall'unificazione dei concetti e parole chiave tra documenti di ogni tipo e giurisdizione, e permette efficacemente di effettuare analisi e confronto. Interoperabilità tecnica consiste invece nella possibilità di unificare sistemi, strumenti ed interfaccie sull'utilizzo dei medesimi linguaggi e tecnologie.

3.3.2 Struttura, pattern e gerarchie

Le specifiche di Akoma Ntoso delineano una struttura a livelli, dove ognuno di questi è indirizzato su di un singolo problema: il livello di testo contiene una rappresentazione fedele del testo legale originale, il livello di struttura fornisce un'organizzazione gerarchica delle parti di testo, il livello di metadati associa informazioni ontologiche ai livelli inferiori [BCD10]. Lo scopo di tale struttura è di bilanciare estendibilità con chiarezza e semantica, e pone una netta distinzione tra contenuto (il quale è stato scritto o almeno approvato da un ente autorevole) e metadati (che sono aggiunti da terze parti e dipendenti, per loro natura, dall'interpretazione).

Pattern. "Un pattern è un'astrazione e distillazione di esperienze passate"[5]

[5]Tratto dal sito ufficiale di Akoma Ntoso (http://www.akomantoso.org/?page_id=47)

Akoma Ntoso utilizza sei pattern nel content model degli elementi, ovvero ogni elemento XML appartiene ad una di queste categorie:

- **Marker**: identifica elementi senza contenuto, il cui significato è dato dalla loro posizione, il loro nome ed i loro attributi; si suddividono in placeholders, elementi che possono comparire ovunque nel testo (es: `noteRef`), e metadati, che possono comparire solo all'iterno della sezione apposita;

- **Inline**: identifica contenitori di testo posizionati all'intero di un block, che danno un particolare significato logico o strutturale al testo che contengono (es: `num`, `person`);

- **Block**: indentifica contenitori di testo separati strutturalmente l'uno dall'altro, e che possono contenere elementi inline (es: `paragraph`, `p`);

- **Container**: raggruppa un insieme di elementi di diverso tipo; il content-model varia nei diversi casi (es: `act`, `meta`, `attachments`);

- **Hierarchical Container** o **HContainer**: rappresenta una gerarchia di contenitori, ognuno dei quale può contenere altri hcontainer o block, ma non direttamente del testo (es: `body`, `chapter`, `paragraph`, `clause`); in Akoma Ntoso l'ordine degli elementi nella gerarchia è libero;

- **Popup** o **Subflow**: si tratta di elementi che inseriscono, all'interno di un flusso di testo, un intera struttura indipendente (tratta ad esempio da un altro documento) che non interagisce con il testo che la circonda (es: `quotedStructure`, `embeddedStructure`).

Un'eccezione a questi pattern è l'elemento `li`, che può contenere sia testo che liste annidate, scelta fatta per conformità alla pratica diffusa in HTML.

3.3.3 La Naming Convention e la gestione degli Id

Oltre allo schema XML, lo standard Akoma Ntoso definisce una naming convention [VPP17] che stabilisce come rappresentare gli IRI[6] dei documenti e gli ID degli elementi XML, necessari per avere il livello di conformità 2.

Identificazione dei documenti. Gli IRI servono ad identificare univocamente le risorse in Akoma Ntoso, e sono importanti in quanto significativi, cioè descrivono logicamente la risorsa stessa e non la sua posizione, persistenti, nel senso che non devono essere modificati nel tempo, e invarianti, nel senso che derivano da proprietà immutabili della risorsa. Essi si basano sul modello concettuale FRBR (*Functional Requirements for Bibliographic Records*), che distingue quattro livelli in cui è possibile identificare un lavoro intellettuale:

a. *Work.* È il concetto astratto di una specifica creazione intellettuale (es. l'atto 3 del 2015). L'IRI di un Work si ottiene concatenando, separandole con un carattere "/", le seguenti parti: il prefisso `/akn`, un codice rappresentante il paese o l'entità internazionale (come ad esempio `it` per l'Italia), il tipo di documento (ad esempio `bill`, `act` o `debateReport`), una specificazione del sottotipo di documento (opzionale), l'ente emanante (opzionale), la data di creazione, il numero o titolo del documento (se necessario per disambiguare) e la componente, nel caso base `!main` (opzionale).

b. *Expression.* Si tratta di una realizzazione del Work che può variare per diverse ragioni, come la lingua o la versione (es. l'atto 3 del 2015 dopo l'emendamento del 03-06-2016). L'IRI di una Expression si ottiene concatenando a quello del Work: la lingua in cui è scritta (tre caratteri), il carattere `@`, la data o il numero di versione ed eventualmente altre informazioni sull'autore.

c. *Manifestation.* Consiste in un formato fisico o elettronico di rappresentazione di una Expression (es. la rappresentazione in formato PDF

[6]Internationalized Resource Identifier, generalizzazione di URI in grado di utilizzare caratteri Unicode.

dell'atto 3 del 2015 dopo l'emendamento del 03-06-2016). L'IRI della Manifestation si ottiene concatenando a quella dell'Expression: l'identificativo dell'autore (opzionale), la data di creazione (opzionale), ulteriori informazioni sul markup (opzionale) e l'estensione del file separata da un punto.

d. *Item*. Si tratta di un esemplare della Manifestation (es. il file act3-2015.pdf sul mio portatile). Non avendo alcuna assunzione sulla modalità di memorizzazione, non è possibile definire delle regole per l'assegnazione di un IRI all'Item, ma possiamo supporre che sia il suo URL (per cui servirà un meccanismo di risoluzione dei nomi per renderlo accessibile via Web).

Listing 3.1: Un esempio di rapprentazione degli IRI all'interno di un documento Akoma Ntoso (dentro all'elemento `<meta>`)

```
<identification source="#source">
    <FRBRWork>
        <FRBRthis value="/akn/it/act/legge/stato
            /2011-07-18/137/!main"/>
        <FRBRuri value="/akn/it/act/legge/stato
            /2011-07-18/137"/>
        <FRBRalias value="urn:nir:stato:legge:2011
            -07-18;137" name="urn:nir"/>
        <FRBRalias value="eli/id/2011/08/11/011G0171/sg
            " name="eli"/>
        <FRBRdate name="" date="2011-07-18"/>
        <FRBRauthor as="#author" href=""/>
        <FRBRcountry value="it"/>
    </FRBRWork>
    <FRBRExpression>
        <FRBRthis value="/akn/it/act/legge/stato
            /2011-07-18/137/ita@2011-08-26/!main"/>
        <FRBRuri value="/akn/it/act/legge/stato
            /2011-07-18/137/ita@2011-08-26"/>
        <FRBRdate name="" date="2011-08-26"/>
```

```
        <FRBRauthor as="#author" href=""/>
        <FRBRlanguage language="ita"/>
    </FRBRExpression>
    <FRBRManifestation>
        <FRBRthis value="/akn/it/act/legge/stato
            /2011-07-18/137/ita@2011-08-26/!main.xml"/>
        <FRBRuri value="/akn/it/act/legge/stato
            /2011-07-18/137/ita@2011-08-26.xml"/>
        <FRBRdate name="" date="2018-10-03"/>
        <FRBRauthor as="#author" href=""/>
    </FRBRManifestation>
</identification>
```

Identificazione di altre entità. La naming convention fornisce anche una sintassi per rappresentare altre entità che non siano documenti, come persone, organizzazioni e concetti. Esse vengono classificate in una delle cosiddette TLC (*Top Level Classes*): Person, Organization, Concept, Object, Event, Location, Process, Role, Term e Reference; tali classi sono disgiunte e non vengono definite ulteriori sottoclassi, in quanto non sarebbe possibile fornire una gerarchia univoca.

L'IRI di un'instanza di una TLC si costruisce concatenando con un "/" le seguenti componenti: il prefisso `/akn/ontology`, il nome della TLC, una sequenza arbitraria di sottoclassi ritenute significative ed un ID, che deve essere univoco all'interno della rispettiva TLC.

Listing 3.2: Esempi di istanze di elementi TLC (all'interno dell'elemento `<meta>`)

```
<references>
    <TLCOrganization id="un" href="/akn/ontology/
        organizations/un/unitedNations" showAs="United
        Nations"/>
    <TLCPerson id="somebody" href="/akn/ontology/
        persons/somebody" showAs="somebody"/>
```

```
    <TLCRole id="author" href="/akn/ontology/roles/akn/
        author" showAs="Author of Document"/>
</references>
```

Identificazione degli elementi. Per riferirsi ad elementi interni ad un documento, si utilizzano degli ID, rappresentati come attributi sui tag XML. Akoma Ntoso definisce tre tipi di identificatori:

- eId (Expression-level identifier): rappresenta univocamente un elemento all'interno dell'Expression ed è definito in base al suo ruolo nella struttura del documento; pertanto richiede di essere aggiornato se, nella versione successiva, cambia di posizione o di significato (es. da articolo a clausola);
- wId (Work-level identifier): permette di identificare lo stesso elemento tra Expression diverse del medesimo Work, qualora gli eId venissero modificati; serve definire quindi una master Expression, i cui eId saranno uguali ai wId delle altre versioni;
- GUID (Globally Unique Identifier): si tratta di un ID opzionale senza alcun vincolo di sintassi o semantica, che può essere utilizzato per scopi locali ad una certa applicazione, come ad esempio mantenere gli identificativi ereditati dal vecchio formato.

Per essere conforme alla naming convention, è necessario che gli eId e i wId siano costruiti secondo una precisa sintassi: un prefisso seguito da "__", un valore rappresentante il nome del tag ed un codice di numerazione preceduto da "_".

Il prefisso rappresenta il contesto in cui compare l'elemento ed è spesso necessario per disambiguare l'ID; tale contesto può essere l'identificativo del documento singolo all'interno di un documento composto, l'eId di un elemento padre o di un elemento precedente che resetta la numerazione. Sono definite delle buone norme su quali elementi fanno parte o meno del contesto: tutti gli elementi che rappresentano la classe (`act`, `bill`, ecc.) resettano la numerazione, i tag inline invece non lo fanno mai (ad eccezione del tag `mod`).

Gli elementi di tipo subflow complicano la gestione, in quanto cambiano il contesto con quello di un altro documento.

La parte centrale dell'ID è la più facile da formalizzare, in quanto è semplicemente il nome del tag con alcune abbreviazioni ben conosciute dai giuristi (ad es. `article` diventa `art`, `chapter` diventa `chp`, `judgementBody` diventa `body`, ecc.). Vi sono tuttavia alcune eccezioni per quando riguarda gli elementi relativi alle TLC e ai tag `componentData`, `keyword` e `component`, in cui è permesso all'autore del markup di inserire come ID una qualsiasi etichetta che ritenga significativa.

Il codice di numerazione serve a distinguere i diversi elementi omonimi all'interno dello stesso contesto. Vi sono tre casi possibili:

- Elementi unici nel loro contesto: in questo caso il numero si può omettere;

- Elementi esplicitamente numerati, ovvero contengono al loro interno il tag `<num>`: in questo caso si utilizza il numero stesso, eventualmente abbreviato rimuovendo separatori non signifiativi (ad esempio se un articolo contiene `<num>Art. 11.2 bis</num>` il suo ID diventerà `"art_11-2bis"`);

- Elementi implicitamente numerati: la numerazione viene fatta contando le occorrenze del tag all'interno del contesto; è compito dell'autore della Manifestation stabilire se un elemento è da contare globalmente (cioè il suo contesto è l'intero documento) o localmente ad un elemento che lo contiene.

3.4 Gestione di documenti Akoma Ntoso

In questo capitolo si è osservata l'importanza dei documenti strutturati all'interno dell'Informatica Giuridica, si è visto che XML permette di rappresentare efficacemente tali documenti e in particolare Akoma Ntoso vanta differenti caratteristiche utili a divenire un possibile standard universale, i cui vantaggi si riassumono nell'interoperabilità dei documenti, nel riutilizzo degli

strumenti per la loro gestione e nella creazione di una visione unificata dei metadati.

Per contrastare la sua complessità e venire incontro alle diverse categorie di utenti che si approcciano per la prima volta a questo standard, è utile avere diversi strumenti specializzati. Il primo tassello per la realizzazione di questi strumenti potrebbe essere una libreria software che unifichi l'accesso ai documenti Akoma Ntoso e fornisca un insieme di operazioni comuni su di essi. È proprio questo lo scopo di *Akomando*, una libreria javascript che fornisce l'accesso in sola lettura ai documenti Akoma Ntoso, come si vedrà meglio nella sezione 4.1. Akomando può essere utilizzato come struttura portante nella realizzazione di strumenti che interagiscono con documenti legali conformi ad Akoma Ntoso, ma non aiuta quando è necessaria la correzione di documenti incompleti, magari risultato di un'operazione di conversione dal vecchio formato di rappresentazione.

Per sopperire a questa mancanza si è pensato di realizzare una libreria software che estendesse Akomando con funzionalità di scrittura, denominata Akomando-Create. Nel prossimo capitolo è spiegato in maggior dettaglio lo scopo di questo progetto e la sua realizzazione.

Capitolo 4

Akomando-Create: una libreria per il perfezionamento di documenti Akoma Ntoso

Il processo di creazione di un documento Akoma Ntoso comincia, come già citato, con la realizzazione di un file XML, creato annotando manualmente il markup su di un file testuale o tramite una conversione automatica con un apposito tool. Una volta che il documento è ben formato e conforme allo standard, si puo usare Akomando per accedere comodamente alle principali informazioni e ai metadati, ma spesso è necessaria una fase intermedia di verifica della bontà del documento, in quanto potrebbero essere presenti degli errori e imperfezioni derivate dal processo di creazione non completamente preciso e raffinato. Molti errori si possono individuare utilizzando lo schema XML per validare il documento, mentre non esiste uno strumento automatico per verificare l'aderenza alla naming convention. Si è pensato quindi di sopperire a questa mancanza fornendo una libreria software capace di verificare e raffinare i documenti per migliorare il livello di conformità allo standard. Ai fini del mio progetto di tesi ho realizzato Akomando-Create, una libreria javascript costruita sopra Akomando in modo da estendere quest'ultimo con operazioni in scrittura.

Questo capitolo è strutturato come segue: in 4.1 è spiegata in maggior dettaglio la libreria Akomando, mentre in 4.2 vengono spiegati meglio gli

obiettivi del progetto Akomando-Create e in 4.3 viene descritta l'architettura del software implementato. Infine in 4.4 si accenna all'idea di valutazione della libreria.

4.1 Akomando: una libreria document-centered per Akoma Ntoso

Akomando[1] è una libreria open-source per gestire documenti Akoma Ntoso, di proprietà di BitNomos[2], scritta in javascript e pubblicata su *npm* con licenza MIT. Si tratta di una libreria *document-centered*, nel senso che agisce su singoli documenti, e *read-only*, in quanto fornisce solo operazioni in lettura.

4.1.1 Modalità d'uso ed API

Akomando può essere utilizzato in tre differenti modalità:

1. Nell'ambiente Node.js, importandola con uno dei seguenti comandi:

   ```
   var akomando = require('akomando');       // ES5
   import { createAkomando } from ('akomando'); // ES6
   ```

2. In un'applicazione web, ovvero in ambiente browser, importando lo script compilato `bundle.akomando.browser.js` nella pagina HTML;

3. Come comando della CLI (Command Line Interface), eseguendo:

   akomando <command> <file>

   dove `command` è l'operazione da eseguire, `file` è il nome del file XML da aprire e ci possono essere ulteriori opzioni per il comando.

Le API permettono di aprire documenti Akoma Ntoso tramite la funzione `createAkomando`, la quale restituisce un oggetto javascript che incapsula il DOM ed espone le diverse operazioni possibili:

[1] https://www.akomando.bitnomos.eu/

[2] BitNomos S.R.L. è un'azienda spin-off dell'Università di Bologna, esperta in informatica giuridica. Sito ufficiale https://www.bitnomos.eu/

- `getAkomaNtoso`: fornisce l'intero documento, in diversi possibili formati, tra cui oggetto DOM, stringa XML, serializzazione HTML, JSON e testo semplice;

- `getDocType`: restituisce il tipo di documento (act, bill, documentCollection, debateReport, ecc...);

- `getMeta`: restituisce tutti i metadati;

- `getHierIdentifier`: restituisce gli identificatori di tutti gli elementi di tipo gerarchia (article, paragraph, ecc...), eventualmente filtrati per nome e contenuto;

- `countDocElements`: fornisce informazioni sul numero di elementi presenti, sia in totale che per ogni tag;

- `getReferencesInfo`: fornisce tutti i riferimenti indicati nell'elemento `<references>` all'interno dei metadati e, per ognuno di essi, gli elementi che vi fanno riferimento;

- `getComponents`: restituisce la lista delle componenti che formano il documento, nel caso si abbia un documento composto.

4.1.2 Tecnologie utilizzate

In seguito vengono indicate le principali tecnologie utilizzate da Akomando, per comprendere il ruolo che queste hanno all'interno della libreria.

DOM.[3] Acronimo per *Document Object Model*, è uno standard del W3C che fornisce un interfaccia ai documenti XML ed HTML accedendo direttamente alla struttura ad albero che questi rappresentano. Per quanto riguarda il linguaggio javascript, DOM è implementato dalla libreria `xmldom`.

JSON. Acronimo per *JavaScript Object Notation*, è un formato per la rappresentazione e lo scambio di dati sottoforma di oggetti semi-strutturati. È leggero e nativamente supportato da javascript.

[3] https://www.w3.org/DOM/

Node.js.[4] Si tratta di un runtime javascript basato sul motore di Google Chrome; permette di utilizzare codice javascript tramite CLI ed eseguire programmi javascript su server, rendendo possibile la creazione di applicazioni full-stack con un solo linguaggio. L'ambiente di esecuzione porta con sé un vasto ecosistema di librerie, gestite da *npm* (Node Package Manager).

Babel.[5] È un compilatore javascript in grado di convertire codice dalla versione ECMAScript 6 (2015) o successivo in una versione precedente e compatibile con i browser. Akomando infatti è realizzato in ES6 per sfruttare al meglio le funzionalità avanzate del linguaggio, tra cui: `import` di moduli e oggetti JSON, costanti, *arrow functions* (sintassi concisa per funzioni anonime), assegnamenti con *pattern matching* e molte altre.

Webpack.[6] Si tratta di un *module bundler* per applicazioni javascript, ovvero un programma che prende in input un insieme di file sorgenti con diverse dipendenze e li raggruppa un'unico file *bundle*. Questa operazione ha un duplice scopo: (1) ottimizzare lo spazio di memorizzazione di un applicazione, facilitandone la portabilità e permettendone l'utilizzo sul front-end di un'applicazione web; (2) effettuare durante il processo una serie di trasformazioni utili, come ad esempio ottimizzare il codice per un certo ambiente di esecuzione o convertirlo in una versione compatibile (i.e. utilizzando Babel).

Mocha e Chai. Si tratta di due librerie disponibili su npm che permettono la realizzazione di *unit test* per la libreria in modo veloce e intuitivo [Har18], tecnica fondamentale durante lo sviluppo. Per realizzarli, si definisce come il software dovrebbe comportarsi in una determinata circostanza (tipicamente cosa dovrebbe restituire una funzione dato un certo input); dopodiché i test dovrebbero essere eseguiti ad ogni cambiamento del codice, al fine di individuare immediatamente se una certa modifica ha compromesso una o più funzionalità.

[4] https://nodejs.org/it/
[5] https://babeljs.io/
[6] https://webpack.js.org/

Mocha[7] permette di eseguire i test su ambiente Node e browser, riportandone i risultati e mostrando quale asserzione è stata la causa del fallimento, mentre Chai[8] fornisce un set di funzioni utili a descrivere asserzioni facilmente leggibili e comprensibili.

Listing 4.1: Un esempio unit test per Akomando scritto con Mocha e Chai. Si noti che le funzioni `describe` e `it` servono a descrivere i test, mentre `expect` definisce le asserzioni che li implementano.

```
describe('#AkomantosoDocumentHandler Document
   document_it.xml (Italy)', function() {
   let myAkomando;
   before((done) => {
      getFile('document_it.xml', (content) => {
         myAkomando = createAkomando({
            aknString: content,
         });
         done();
      })
   });

   it('After having parsed the XML string the Akn
      Document Element must exist', function() {
      const result = myAkomando.getAkomaNtoso().
         documentElement.nodeType;
      expect(result).to.equal(1);
   });

   it('The document type must be act', function() {
      const docType = myAkomando.getDocType();
      expect(docType).to.equal('act');
   });
});
```

[7] https://mochajs.org/
[8] https://www.chaijs.com/

4.1.3 Verso la correzione dei documenti Akoma Ntoso

Akomando risulta essere uno strumento valido per accedere alle principali informazioni contenute nei documenti Akoma Ntoso senza aver bisogno di studiare approfonditamente lo standard; tuttavia esso non aiuta nella creazione di documenti, che avviene invece a partire da un vecchio formato di memorizzazione. Questo processo è tuttora in corso in diversi paesi, principalmente per mezzo di convertitori, sviluppati per creare documenti Akoma Ntoso a partire dal precedente formato. La conversione, tuttavia, è soggetta ad errori, e tuttora sono presenti molti documenti imprecisi e non pienamente conformi alla naming convention. Akomanto-Create nasce per sopperire a questa mancanza, estendendo Akomando con funzionalità di scrittura per la correzione dei documenti, in modo che ogni ente interessato possa integrarla all'interno dei propri strumenti di gestione dei testi legali, personalizzandola, se necessario, per le proprie particolari usanze ed obiettivi. Nella prossima sezione mi occuperò di descriverne le funzionalità.

4.2 Funzionalità di Akomando-Create

Akomando-Create è pensato come un plugin di Akomando in grado di effettuare operazioni in scrittura al fine di perfezionare un documento Akoma Ntoso; in particolare si cerca di raffinare i documenti per migliorarne il livello di conformità allo standard e massimizzarne così la bontà semantica. Il software è stato progettato e sviluppato cercando di raggiungere i seguenti obiettivi:

1. Fornire operazioni basilari secondo il modello CRUD (Create, Read, Update, Delete), perché vengano eseguite in modo controllato senza rompere la struttura del documento XML;

2. Fornire funzionalità per la verifica della conformità degli ID alla naming convention di Akoma Ntoso (vedi sezione 3.3.3) ed eventualmente correggerli, in modo automatico o semi-automatico, ossia con un numero minimo di interventi manuali da parte da parte di un esperto del settore;

3. Fornire funzionalità per il controllo dei riferimenti interni al documento stesso, con possibilità di correzione, automatica o semi-automatica;

4. Permettere la verifica di validità del documento rispetto allo schema XML (individuando automaticamente la versione da utilizzare).

Akomando-Create utilizza le funzionalità read-only di Akomando, ma allo stesso tempo necessita di entrare direttamente all'interno del DOM per effettuare le modifiche; bisogna quindi fare particolare attenzione per non rompere gli invarianti presenti nei meccanismi interni di Akomando. La soluzione a questo problema è la seguente: ogni modifica in scrittura al DOM comporta la creazione di un nuovo oggetto Akomando; così facendo Akomando-Create non utilizzerà un solo oggetto Akomando per un documento, ma dovrà mantenere una lista di oggetti che rappresentano la storia delle modifiche all'interno della sessione di lavoro, permettendo di adempire banalmente a funzionalità di backtrack. Tale funzionalità potrebbe essere utile ad esempio nel caso in cui si volesse realizzare un editor specializzato basato su questo plugin.

4.2.1 Modalità d'uso ed API

Akomando-Create può essere utilizzato sia in modalità Node che in modalità browser, con l'unica differenza che in quest'ultima non è possibile validare direttamente i documenti, a causa di una dipendenza della libreria di validazione compilata in linguaggio nativo C e quindi non supportata dai browser.

La libreria realizzata permette di costruire un oggetto `AkomandoCreator` attorno ad un documento XML già esistente, in modo simile a quello di Akomando, che espone come metodi le seguenti funzioni di API:

- `getAkomando`: fornisce l'oggetto Akomando relativo al documento, nella sua versione successiva a tutte le modifiche effettuate all'interno di Akomando-Create, permettendo di accedere alle API descritte in precedenza (sezione 4.1.1);

```
import AkomandoCreator from 'akomando-create';
```

```
var xml = fs.readFileSync('myAkomantoso.xml').
    toString();
var aknCreator = new AkomandoCreator({ aknString:
    xml });
aknCreator.getAkomando().getAkomaNtoso('JSON');
```

- `handle`: permette di utilizzare le operazioni CRUD, specificando come primo parametro l'operazione richiesta, tra *create*, che permette di inserire un nuovo elemento all'interno di un elemento già esistente, *read*, che restituisce un elemento a partire dal suo XPath, *update*, che consente di inserire testo o attributi in un elemento esistente, eventualmente rimuovendo quelli già presenti, e *delete*, che elimina un nodo dal documento;

```
// function CREATE
aknCreator.handle('p', 'create', {
        position: '/akomaNtoso[1]/act[1]/preamble
            [1]',
        index: 0,
        content: 'new element with CREATE!',
});

// function READ
var elem = aknCreator.handle('meta', 'read', {
    position : '/akomaNtoso[1]/act[1]/meta[1]'
});

// function UPDATE
aknCreator.handle('num', 'update', {
        position: '/akomaNtoso[1]/act[1]/body[1]/
            article[1]/num[1]',
        content : 'modified with UPDATE!',
        attributes: { name: 'Numero' },
        removeOld: true
});
```

```
// function DELETE
aknCreator.handle('lifecycle', 'delete', {
   position:
    '/akomaNtoso[1]/act[1]/meta[1]/lifecycle[1]'
});
```

- `checkIds`: verifica l'aderenza degli ID presenti nel documento alla naming convention di Akoma Ntoso (sezione 3.3.3), agendo in una modalità tra *force*, che cerca di correggerli automaticamente, e *interactive*, che si limita a restituisce gli errori presenti;

```
var idErrors = aknCreator.checkIds();

// shape of the resulting object
idErrors == [
  {
    element: Element {...},
    idName: 'eId',
    value: 'ro1',
    suggestion: 'original',
    message: 'Bad id eId="ro1": expected ...'
  },
  ...
]
```

- `fixId`: prende in input un errore di cattiva formattazione di un ID (nel formato resituito da `checkIds`) e lo corregge come specificato, avendo cura di aggiornare gli elementi che vi fanno riferimento;

```
aknCreator.fixId(elem, 'eId', 'metadati', '
   act__meta');
```

- `fixAllIds`: corregge automaticamente gli errori restituiti da `checkIds` (equivalente a chiamare `checkIds` in modalità *force*);

```
aknCreator.checkIds('wId', 'interactive');
aknCreator.fixAllIds();
```

- `checkReferences`: controlla la presenza di riferimenti interni errati, ovvero che puntano ad ID non esistenti nel documento corrente;

```
var refErrors = aknCreator.checkReferences();

// shape of the resulted object
refErrors == [
  {
    refersTo: 'roma',
    referenceLinkedInAttribute: 'refersTo',
    nodeInfo: {
      name: 'location',
      eId: 'location_1',
      wId: '',
      GIUD: '',
      text: 'Roma',
      xpath: '/akomaNtoso[1]/act[1]/conclusions[1]/
         p[1]/location[1]'
    }
  },
  ...
]
```

- `fixReference`: consente di modificare un riferimento, specificando il nuovo ID a cui puntare (assicurandosi della sua esistenza);

```
aknCreator.fixReference(refErrors[0], 'somebody');
```

- `validate`: consente di verificare la validità del documento rispetto allo schema XML di Akoma Ntoso e, in caso sia invalido, restituisce gli errori di validazione.

```
let validErrors = aknCreator.validate();

// example of result
validErrors == [
  { Error: "Element 'proprietary': Character
     content other than whitespace is not allowed
     because the content type is 'element-only'. "
     },
  { Error: "Element 'paragraph': This element is
     not expected. Expected is one of (ref, mref,
     rref, mod, mmod, rmod, remark, recordedTime,
     vote, outcome)." },
  ...
]
```

Nella prossima sezione verrà approfondita l'architettura software con cui è stato realizzato Akomando-Create, osservando come queste funzionalità interagiscono tra loro e con quelle di Akomando.

4.3 Architettura ed Implementazione

In questa sezione è descritta l'architettura di Akomando-Create, che si può vedere rappresentata in figura 4.1. Ogni blocco rappresenta un modulo, costituito da un file javascript che implementa ed esporta delle funzioni, spesso racchiuse all'interno di una classe (oppure di una classe statica, qualora non siano presenti campi).

4.3.1 Akomando-Validate

La prima cosa che si può notare è la presenza di un ulteriore package denominato Akomando-Validate: esso si occupa di validare i documenti XML rispetto allo schema XML della versione più opportuna, deducibile dal namespace definito nel file stesso. Questa separazione si è rivelata necessaria a

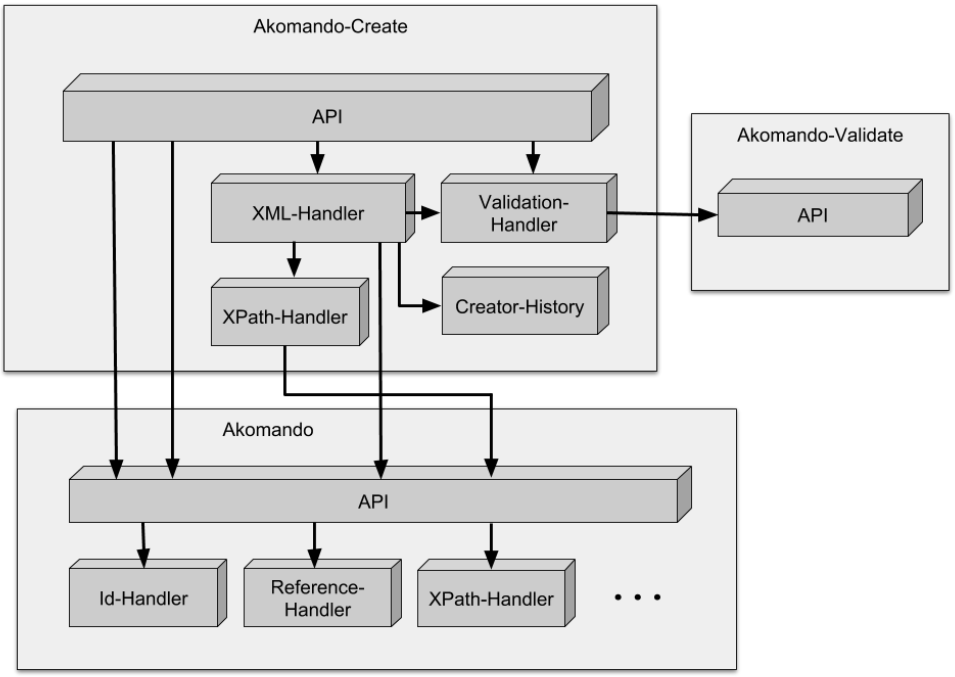

Figura 4.1: Rappresentazione astratta dell'architettura di Akomando-Create, in relazione con Akomando e Akomando-Validate.

causa della libreria *libxmljs*, utilizzata per la validazione; essa è un wrapper in javascript per la libreria *libxml* scritta in C, pertanto non è possibile includerla nel bundle utilizzando webpack, né tantomeno eseguirla su di un browser. Per questo motivo si è ritenuto opportuno creare un pacchetto apposito, che viene utilizzato da Akomando-Create nella versione per Node, mentre non è possibile includerlo nella versione browser. È possibile tuttavia creare un servizio back-end per la validazione usando un server Node ed effettuare un binding con Akomando-Create presente nel front-end.

Akomando-Validate si compone di un solo modulo, il quale definisce una classe `AkomandoValidator`, che fornisce l'operazione `validate` su una stringa XML. È possibile utilizzarlo sia come API che tramite linea di comando.

Listing 4.2: Esempio di utilizzo di akomando-validate tramite API.

```
import AkomandoValidator from 'akomando-validate';
```

```
var xml = fs.readFileSync('myAkomantoso.xml')
          .toString();
var validator = new AkomandoValidator();
var errors = validator.validate(xml);
```

Listing 4.3: Esempio di utilizzo di akomando-validate tramite Command Line Interface.

```
node akomando-validate.bin.bundle.js myAkomantoso.xml
```

4.3.2 Nuove funzionalità per Akomando

Per realizzare le funzionalità di Akomando-Create, ho proposto ed implementato alcune nuove operazioni per Akomando, ritenendo che, trattandosi di funzioni in sola lettura, potessero risultare utili anche in altri contesti.

Le nuove funzioni di API realizzate sono le seguenti:

- `getIdReferences`: fornisce la lista di nodi che fa riferimento a qualche ID attraverso un proprio attributo; a differenza di `getReferencesInfo`, già presente nella libreria, questa funzione individua anche i riferimenti ad ID non presenti nel documento, permettendo quindi di idenficare eventuali errori;

  Listing 4.4: Esempio di utilizzo di getIdReferences. Il formato del risultato è equivalente a quello illustrato nel codice di `checkReferences`.

  ```
  var refs = myAkomando.getIdReferences();
  ```

- `getElementsFromXPath`: restituisce uno o più nodi individuati dall'XPath passato come input; per motivi di comodità ed efficienza si gestiscono solamente i path completi ed assoluti, rinunciando a path relativi, ricerche in profondità, attributi, wildcard, predicati ed assi differenti da quello di default (child);

  Listing 4.5: Esempi di utilizzo di getElementsFromXPath.

```
// Esempi funzionanti
myAkomando.getElementsFromXPath('/akomaNtoso/act/
    body/article');
myAkomando.getElementsFromXPath('/akomaNtoso[1]/act
    [1]/meta/classification/keyword[2]');

// Esempi non funzionanti
myAkomando.getElementsFromXPath('./article');
myAkomando.getElementsFromXPath('//article');
myAkomando.getElementsFromXPath('/akomaNtoso/act/
    body/article[@art="art_1"]');
myAkomando.getElementsFromXPath('/akomaNtoso/act/
    meta/*');
myAkomando.getElementsFromXPath('/akomaNtoso/act/
    descendant::meta/');
myAkomando.getElementsFromXPath('/akomaNtoso/act/
    body/article[last()]');
```

- `getElementById`: restituisce l'elemento, se presente, con un certo ID, di cui è possibile specificarne il nome (di default è *eId*); pertanto, non essendo richiesto di utilizzare ID univoci, è possibile che vengano restituiti più elementi;

Listing 4.6: Esempio di utilizzo di getElementById.

```
myAkomando.getElementById({
    idName: 'eId',
    idValue: 'art_1'
});
```

- `checkIds`: verifica se gli ID (*eId*, *wId* o entrambi) sono ben formati rispetto alla naming convention, restituendo ogni errore individuato.

Listing 4.7: Esempio di utilizzo di checkIds. Il formato del risultato è equivalente a quello illustrato nel codice di `checkIds` di Akomando-Create.

```
var idErrors = myAkomando.checkIds({
    idName: 'wId'
});
```

Le nuove funzionalità hanno portato alla creazione di nuovi moduli, o alla modifica di quelli esistenti.

- Il modulo **Reference-Handler** è stato ampliato con la funzione che implementa `getIdReferences`.

- Il nuovo modulo **XPath-Handler** è stato creato per implementare le funzionalità sugli XPath semplificati, in modo da non utilizzare la libreria standard *xpath*, così grande da risultare scomodo includerla nel bundle.

- Il modulo **id-Handler** è stato realizzato per implementare le funzioni sugli ID (`getElementById` e `checkIds`). Permette di scansionare l'intero DOM, salvando tutti gli ID relativi ad un certo attributo in una mappa, con cui è possibile di accedervi in maniera efficiente.

4.3.3 Moduli di Akomando-Create

Una volta completate le funzioni di Akomando su cui cui si basano quelle di Akomando-Create, ho potuto implementare i seguenti moduli:

- **XML-Handler**, che implementa e gestisce le operazioni *create*, *read*, *update* e *delete* sul DOM del documento; ad ogni modifica (ovvero ogni operazione ad eccezione di *read*) crea un nuovo oggetto Akomando e lo include nella history. È possibile impostare le operazioni in modo che verifichino la validità del documento modificato e facciano backtrack nel caso risulti invalido.

- **Creator-History** memorizza la lista di oggetti akomando risultati da ogni operazione, con un meccanismo FIFO (*First In First Out*) per fare backtrack.

- **XPath-Handler** gestisce le ricerche tramite XPath completi, allo stesso modo di Akomando.
- **Validation-Handler** funziona da interfaccia al pacchetto Akomando-Validate; chiama direttamente la validazione se si è in ambiente Node, mentre permette di definire una funzione custom che faccia da servizio, se si usa su un browser.

4.4 Valutare Akomando-Create attraverso un'applicazione demo

Una volta implementata la libreria Akomando-Create, ci si è interrogati riguardo alla sua utilità e facilità d'uso nello sviluppo di un'applicazione concreta. Si pensato quindi di realizzare una demo per mettere alla prova le funzionalità e l'efficacia del software realizzato. Lo scopo è di utilizzare la libreria su una collezione di file XML in Akoma Ntoso, per verificarne la validità rispetto allo schema e la buona formattazione degli ID e dei riferimenti, creando in seguito alcune visualizzazioni grafiche in grado di fornire una idea complessiva della bontà del dataset. I documenti utilizzati, che sono stati gentilmente forniti dal CIRSFID, sono risultati di un processo di marcatura automatica di testi legali tramite parser; per questo motivo è possibile la presenza di errori dati da piccole imprecisioni di tali strumenti.

La realizzazione di questa applicazione è riportata nel prossimo capitolo, in cui verrà anche valutata l'utilità di Akomando-Create in relazione alla tesi iniziale.

Capitolo 5

Valutazione di Akomando-Create in un contesto applicativo concreto

Nel capitolo 2 si è potuto osservare come la Data Analysis possa essere applicata all'informatica giuridica, permettendo di individuare risultati e trend a partire da dati legali e di visualizzarli a beneficio di legislatori, operatori di uffici pubblici, data journalist e cittadini. Nel capitolo 3 si è notato che per avere un'analisi il più possibile semplice ed efficace è utile disporre di documenti strutturati, che un buon formato per rappresentarli è XML e che vi sono grandi vantaggi nell'utilizzare uno standard unico quale Akoma Ntoso. Tale standard permette di memorizzare, all'interno dei documenti stessi, dei metadati su cui è definita, seppur in maniera informale, una semantica, coerente ed utilizzabile in ogni diverso contesto giuridico.

Avendo a disposizione uno strumento di base come Akomando per accedere in modo facile ed uniforme ad un documento Akoma Ntoso, ho introdotto ed implementato Akomando-Create (capitolo 4) allo scopo di fornire un primo livello nella creazione di strumenti per la correzione ed il perfezionamento dei documenti nell'aderenza allo standard.

In questo capitolo illustrerò il progetto dimostrativo *ANANAS*, una dashboard che integra Akomando-Create in un contesto applicativo concreto, con lo scopo di realizzare e mostrare alcune Data Visualization riguardanti i

metadati di un set di documenti Akoma Ntoso.

Nella sezione 5.1 viene spiegato lo scopo di ANANAS, definendo l'ipotetico utilizzatore finale, le funzionalità da realizzare e la sua implementazione, in 5.2 vengono illustrati i grafici creati ed infine in 5.3 vengono discussi i risultati in relazione allo scopo iniziale di questo lavoro.

5.1 ANANAS: Akoma Ntoso Analysis of Names And Semantics

ANANAS, acronimo per *Akoma Ntoso Analysis of Names And Semantics*, nasce come demo per mettere alla prova le funzionalità di Akomando-Create su un set di documenti Akoma Ntoso, analizzandoli per verificarne la bontà sintattica e semantica, correggendo rapidamente alcuni tipi di errori e visualizzando semplici grafici per mostrare a colpo d'occhio le principali caratteristiche della collezione. Si tratta di una *dashboard*, ovvero, secondo la definizione di Stephen Few, uno strumento che permette di illustrare graficamente le informazioni maggiormente necessarie a raggiungere un obiettivo, disponendole su un'unica schermata così da essere comprese a colpo d'occhio [Few06]. Essa è realizzata tramite applicazione web composta da un lato *back-end*, in grado di accedere e fornire i documenti presenti all'interno di un certo database, e di uno *front-end*, che permette di interagire con tali documenti e mostrare grafici, tramite un'apposita interfaccia utente.

5.1.1 Funzionalità e casi d'uso

Come già accennato, ANANAS serve a mostrare come realizzare uno strumento per l'accesso ad una collezione di documenti Akoma Ntoso da parte di un utente interessato alla bontà dei metadati. In questo contesto l'utente finale interessato è il cosiddetto "redattore legale", cioè, come introdotto nella sezione 3.3.1 in riferimento ai fruitori di uno standard universale, un membro dell'ufficio incaricato alla stesura del documento digitale da parte del legislatore. Egli è dotato di specifiche competenze e caratteristiche, che vedremo meglio nel seguente esempio di caso d'uso.

Caso d'uso. L'ente legislativo X ha deciso, dopo lunghe consultazioni interne a cui hanno partecipato anche alcuni esperti nel campo dell'informatica del diritto, di utilizzare lo standard Akoma Ntoso come formato ufficiale per i propri documenti digitali. Prima, però, ha ritenuto necessaria la conversione dei documenti già esistenti, al fine di verificare l'impatto del nuovo standard sul processo di redazione. L'ufficio tecnico Y, a cui è stato assegnato questo incarico, ha individuato ed utilizzato il convertitore adeguato e lo ha adoperato per generare 200 documenti Akoma Ntoso, che ha conservato all'interno del proprio server; tuttavia l'entusiasmo iniziale da parte di X è scemato rapidamente e per oltre un anno non se n'è saputo più nulla. Un giorno è arrivata ad Y una telefonata con richiesta di sistemare i documenti, ormai quasi dimenticati, per pubblicarli sul nuovo portale Open Data in fase di rilascio. Antonio, incaricato di seguire questa richiesta, lavora nel settore da oltre 15 anni, è una persona precisa e diligente sul lavoro, dispone di ottime conoscenze giuridiche e buone competenze su strumenti informatici; si era informato tempo fa sulle principali caratteristiche di Akoma Ntoso, ma adesso i suoi ricordi sono vaghi e sfumati. Decide per scrupolo di validare i 200 documenti e scopre che diversi non superano il test; si chiede quindi se possa esistere uno strumento semplice da imparare che possa verificare la bontà di questi file e guidarlo nella loro correzione.

Funzionalità. Alla luce delle caratteristiche dell'utente finale e del caso d'uso individuato, ANANAS dovrà disporre almeno delle seguenti funzionalità:

1. Accedere ad un database inserendo, se necessario, le credenziali di accesso;
2. Visualizzare la lista dei documenti presenti ed accessibili sul database;
3. Scaricare uno o più documenti;
4. Eseguire l'upload di uno o più documenti;
5. Aprire un documento per visualizzarlo;
6. Eseguire un test di validità sintattica sul documento corrente;

7. Eseguire un test di conformità del documento corrente alla naming convention di Akoma Ntoso;

8. Performare delle semplici operazioni di correzione degli errori individuati, in modo automatico ove possibile, altrimenti in modo semi-automatico;

9. Scegliere e visualizzare un grafico tra una lista di Data Visualization predefinite, calcolate sulla lista corrente di documenti;

5.1.2 Architettura e componenti

ANANAS sarà formata dalle seguenti componenti:

a. Un'interfaccia al database, localizzata sul server, in grado di fornire la lista dei documenti presenti oppure il contenuto di un documento richiesto dal client per la visualizzazione o il download;

b. Uno strumento per la verifica dei documenti, in termini di validità e conformità, associato ad un meccanismo, automatico o semi-automatico, per la correzione di eventuali errori riscontrati;

c. Uno strumento in grado di aprire i documenti ed estrapolarne le informazioni utili alle visualizzazioni grafiche;

d. Una libreria per creare e mostrare grafici comprensibili all'utente, a partire dalle informazioni estratte da ogni documento.

Per quanto riguarda l'interfaccia al database, supponiamo di avere uno strumento che permetta di reperire ed archiviare documenti tramite un'apposita API; ai fini di questo lavoro il tipo di database utilizzato non è interessante.

Per ciò che concerne la visualizzazione di grafici, esistono diverse librerie per javascript utilizzabili lato client; una delle più usate è sicuramente Chart.js[1], una libreria open-source in grado di creare facilmente grafici a linee, a torta, istogrammi e molti altri.

[1] https://www.chartjs.org/

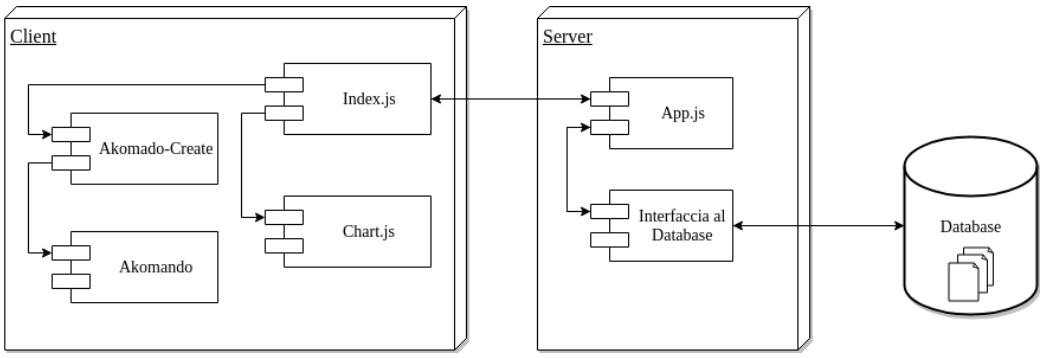

Figura 5.1: Architettura di ANANAS.

L'accesso alle informazioni mantenute nel formato Akoma Ntoso è realizzato tramite Akomando-Create, il quale fornisce operazioni di verifica e di correzione dei documenti e permette anche di accedere alle funzionalità presenti in Akomando.

5.1.3 Interfaccia utente

L'interfaccia utente di ANANAS è stata realizzata utilizzando il framework grafico semantic-ui[2] per impostare la grafica e le funzionalità di base degli elementi dell'interfaccia. Com'è possibile notare nelle figure 5.2 e 5.3, questa è composta da:

1. una barra superiore in cui indicare la posizione del dataset da cui recuperare i documenti;

2. una sezione laterare posizionata sulla sinistra che mostra la lista dei documenti presenti e permette di aprire uno di essi o di effettuare il download e l'upload dei file XML;

3. un'area principale che mostra i grafici selezionati dall'utente, oppure un'anteprima del documento aperto e gli errori presenti.

L'area principale si comporta in due modi possibili, a seconda dell'interazione con l'utente, che denominiamo *schermata collezione* e *schermata documento*. La prima è quella visualizzata nel momento in cui viene aperto

[2]https://semantic-ui.com/

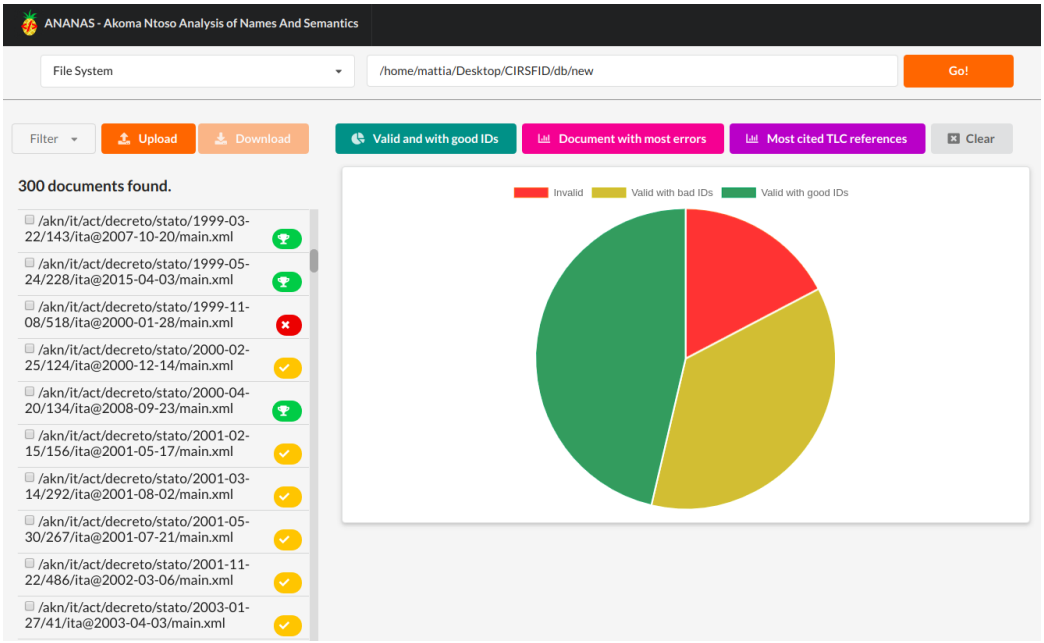

Figura 5.2: Interfaccia utente di ANANAS: una volta indicato il dataset da utilizzare è possibile selezionare uno dei grafici predefiniti o aprire uno dei documenti presenti.

il database (fig. 5.2), dopo aver individuato la lista di documenti presenti; permette all'utente di selezionare un grafico, tra un insieme predefinito, riguardante l'intera collezione. La seconda si attiva nel momento in cui l'utente clicca su uno dei documenti (fig. 5.3), mostra il documento XML e permette di calcolare ed illustrare i diversi tipi di errori presenti: (1) quelli di validazione, (2) quelli di ID, cioè dati da identificatori sugli elementi XML che non rispettano la naming convention, e (3) quelli di riferimento, ovvero quei tag che fanno riferimento, con un attributo, ad un ID non definito nel documento stesso. Gli errori di ID possono essere sistemati automaticamente con l'apposito pulsante "Fix all" nella sezione corrispondente, mentre quelli di riferimento possono essere corretti manualmente, creando una nuova entità con l'ID corrispondente a quello del riferimento stesso.

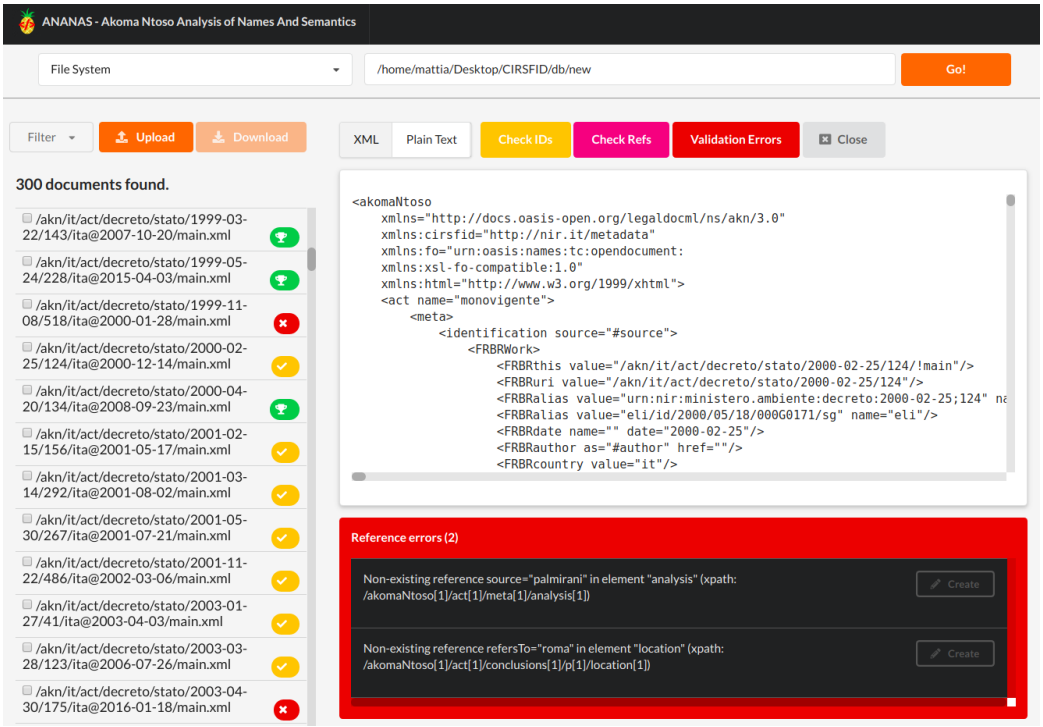

Figura 5.3: Interfaccia utente di ANANAS dopo aver selezionato un documento: è possibile visualizzarlo e verificare la presenza di errori sugli identificatori, sui riferimenti interni e sulla validazione del documento.

5.1.4 Cenni sulla realizzazione

In seguito sono riassunti i passi implementativi svolti per la realizzazione della demo:

1. Creazione dell'interfaccia utente utilizzando il framework grafico semantic-ui;
2. Realizzazione di un semplice server javascript tramite Node.js;
3. Individuazione di una libreria semplice per l'interfacciamento al dataset all'interno di una cartella locale, da utilizzare sul server;
4. Collegamento dell'interfaccia utente alle chiamate al server;
5. Apertura dei documenti ricavati sul client con l'API di Akomando-Create;

6. Accesso alle informazioni d'interesse di ogni documento tramite le funzioni di API, e conseguente raggruppamento delle stesse al fine di ottenere i valori necessari alla libreria Chart.js per le visualizzazioni grafiche desiderate.

Alcuni di questi punti rappresentano passaggi necessari, intrinsechi nella realizzazione di un'applicazione web e indipendenti dallo scopo di quest'ultima. Unico vincolo sotto questo aspetto sta nella necessità di utilizzare Node.js, dovuta all'esistenza di eseguire su server il modulo Akomando-Validate (già riportato in sezione 4.3.1); questo non ha rappresentato un limite per il progetto ANANAS, anzi, grazie alla presenza di numerose librerie javascript per la realizzazione di funzionalità server[3], è risultato un compito di immediata realizzazione.

Per quanto riguarda l'accesso al dataset ho potuto adoperare una libreria per Node sviluppata al CIRSFID ed in fase di rilascio, denominata Akomando-DB, la quale consente di archiviare e recuperare file Akoma Ntoso sfruttando la naming convention sui documenti, secondo il modello FRBR. Questa decisione comunque non è vincolante, ma sarebbe stato possibile l'utilizzo di altre tecnologie.

I punti critici nel comprendere l'efficacia di Akomando-Create nella realizzazione della demo sono il 5 e il 6, che richiedono uno studio minimo delle API sue e del sottostante Akomando. Non è possibile per me valutare oggettivamente la bontà delle API di Akomando-Create, ma nella mia esperienza quelle di Akomando sono risultate di facile comprensione grazie alla documentazione arricchita di esempi; pertanto è mia opinione che anche le funzionalità di Akomando-Create, il quale segue lo stile implementativo di Akomando, risultino di facile comprensione se ben documentate.

L'ultimo punto della lista dipende in particolar modo dalle visualizzazioni che si intendono realizzare. Nel mio caso ho avuto bisogno solamente delle funzioni di API di Akomando-Create, utilizzate su ognuno dei documenti tra-

[3]Nel mio caso ho adoperato *express* per l'esecuzione del web server, *express-session* per la gestione dei cookie e delle variabili di sessione ed *express-fileupload* per facilitare il trasferimento di file tra client e server.

mite funzioni di ordine superiore[4] per estrarre e raggruppare le informazioni necessarie a Chart.js per la costruzione del grafico.

5.2 Visualizzazioni su ANANAS

In questa sezione verranno illustrate alcune possibili visualizzazioni d'interesse per un redattore legale, che sono state implementate all'interno di ANANAS. Per ognuna di queste verrà spiegato lo scopo e la potenziale utilità per l'utente finale. Grazie alle funzionalità di Akomando-Create e Chart.js sono state necessarie meno di un centinaio di righe di codice per l'intera creazione dei tre grafici. Per chi volesse approfondire l'aspetto implementativo, il codice è riportato nell'appendice A.

I. **Validità dei documenti e bontà degli identificatori.**

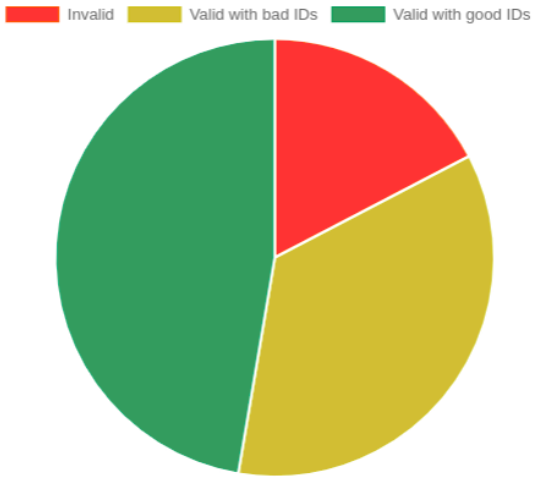

Figura 5.4: Grafico I, che mostra la percentuale di documenti invalidi, validi con ID non conformi e validi con ID ben formati.

Il primo grafico ha lo scopo di mostrare la situazione del dataset, per quanto concerne la validità e conformità dei documenti che ne fanno parte, e darne subito consapevolezza all'utente. I documenti vengono

[4]Le funzioni di ordine superiore servono ad applicare uniformemente una trasformazione ad ogni elemento di una collezione. Si tratta di una tecnica basilare del paradigma funzionale, introdotto oramai da tutti i principali linguaggi di programmazione, tramite operazioni native o apposite librerie.

partizionati in tre categorie: (1) quelli invalidi, ovvero che non superano il test di validità rispetto allo schema XML, (2) quelli validi ma con identificatori mal formati, che quindi non rispettano la naming convention di Akoma Ntoso, e (3) quelli validi e con ID ben formati. Le dimensioni in percentuale di queste tre classi vengono rese immediatamente comprensibili tramite un grafico a torta (figura 5.4).

II. **Documenti con maggior presenza di errori.**

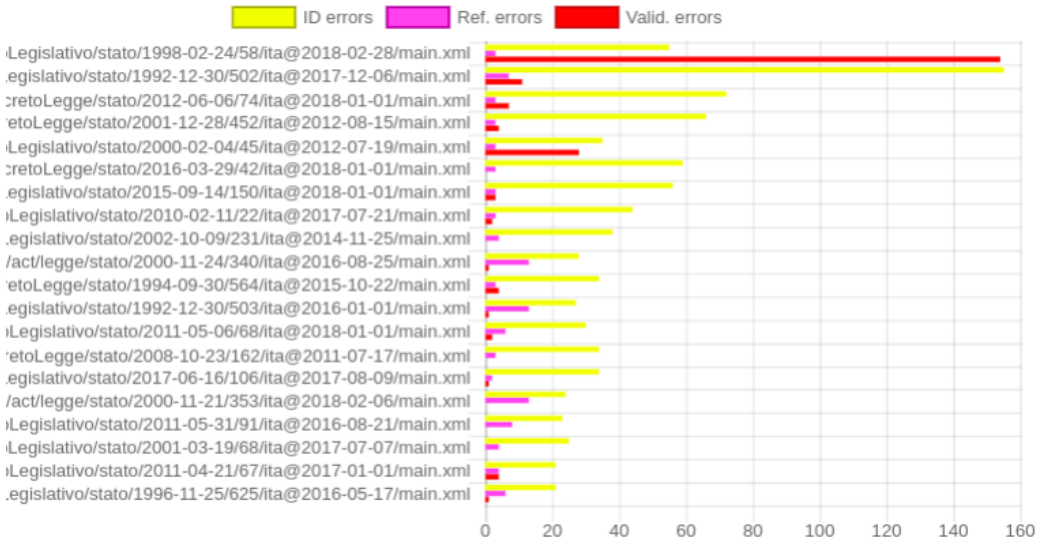

Figura 5.5: Grafico II, che mostra tramite barre orizzontali il numero di errori presenti, per ognuna delle tre tipologie, nei primi 20 documenti più errati.

Il secondo grafico serve al redattore legale per individuare quali documenti riportano un maggior numero di errori, per capire dove dovranno essere concentrati i propri sforzi, che siano nel perfezionamento del convertitore (nel caso in cui questo sia realizzato dallo stesso ufficio) o nella correzione manuale. Gli errori individuati sono quelli sugli ID, sui riferimenti interni e di validazione rispetto allo schema. I documenti vengono ordinati per numero decrescente di errori totali, e ne vengono visualizzati solo i primi 20 (figura 5.5).

III. **Risorse citate da più documenti.**

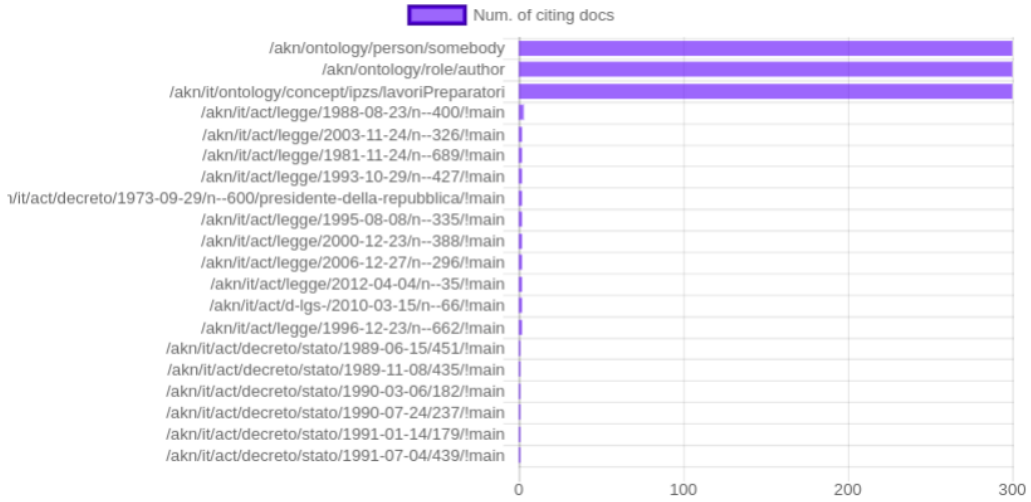

Figura 5.6: Grafico III.

Questo grafico mette alla luce alcune caratteristiche dei metadati, che non possono essere individuate direttamente con i file in pura forma testuale. Esso individua le risorse più utilizzate all'interno dei documenti, rappresentate da un certo IRI, e, contando il numero di occorrenze all'interno di tutto il dataset, mostra le 20 più frequenti. A differenza dei precedenti, non dà all'utente un riscontro diretto degli errori dei documenti, ma fornisce piuttosto una concezione del significato di alcuni dei metadati; è comunque possibile che questi portino all'individuazione di migliorie necessarie. Nel caso della figura 5.6, ad esempio, ci si rende subito conto che vi sono metadati imprecisi, come la risorsa più citata, identificata come *"somebody"*.

5.3 Valutazione del progetto

Grazie ai componenti individuati nella sezione 5.1.2 (Architettura e componenti), la creazione di ANANAS, esempio di un possibile strumento di analisi e correzione di un set di documenti legali, è risultata facile e relativamente veloce. Alla luce di questa demo, è mio interesse valutare Akomando-Create come componente di rilievo per la maggior diffusione dello standard Akoma

Ntoso e di documenti legali di alta qualità.

Come prima considerazione si pensi allo scopo principale di Akomando, ovvero quello di fornire ai programmatori un punto d'accesso uniforme alle principali caratteristiche dei documenti Akoma Ntoso, permettendo a chi realizza tool per questo standard di ridurre il carico di lavoro dato dallo studio preliminare dei pattern, dello schema e della naming convention. Questo obiettivo è stato considerato anche nella progettazione di Akomando-Create, seppur cambiando il focus delle funzionalità, da sola lettura a scrittura e modifica. Il concetto chiave è l'interoperabilità degli strumenti, motivo per cui si è scelto di utilizzare come standard di riferimento uno in grado di adattarsi ad ogni diverso contesto giuridico, che potesse mantenere la validità al variare del tempo e del luogo geografico. Con la diffusione di questo standard aumenteranno notevolmente le entità coinvolte nella realizzazione e nell'utilizzo di documenti Akoma Ntoso, portando alla nascita di numerosi tool per la creazione e conversione di questi a partire da un formato differente, ognuno con le proprie peculiarità derivate dalle particolari tradizioni di ogni sistema giuridico. Il rischio è la frammentazione dello standard in sotto-categorie di documenti XML che, pur mantenendo una grande similarità e magari risultando validi per lo stesso schema, sminuiscano le caratteristiche più sottili dello standard, quali la naming convention riguardante gli ID e le risorse. È importante quindi che questi aspetti siano tenuti in considerazione dai redattori legali durante l'intero processo di creazione e perfezionamento dei documenti, e che vi siano, quindi, strumenti adeguati a supporto di questo compito.

Per analizzare più concretamente la posizione di Akomando-Create rispetto a questo obiettivo, si considera il caso d'uso descritto nella sezione "Funzionalità e casi d'uso" di ANANAS (5.1.1) e si analizza come tale situazione possa essere risolta dall'utente di interesse, il redattore legale. Si considerano due scenari possibili, differenti per la sola disponibilità di Akomando-Create.

Caso 1. Avendo a disposizione Akomando-Create si è dimostrato semplice realizzare un'applicazione web per interagire con una collezione di documenti, eseguire test di validità e conformità, creare grafici riassuntivi sul dataset e correggere velocemente alcuni tipi di errore, quelli sugli ID e sui riferimenti.

Con i semplici grafici presenti in ANANAS, un redattore legale è in grado di individuare immediatamente i documenti che risultano più problematici, aprirli e visualizzare la lista degli errori presenti e correggere quelli più comuni in maniera controllata.

Infine, nel caso in cui nell'esecuzione del processo di elaborazione dei documenti giuridici si identificasse il bisogno di definire dei controlli *ad hoc* per i documenti, ad esempio al fine di renderli coerenti con le tradizioni o compatibili in qualche modo con un differente strumento, si potrebbe creare un tool al di sopra di Akomando-Create, in modo da garantire che non si rompano i principi più sottili dello standard.

Caso 2. Senza uno strumento specializzato nello standard, l'operatore avrebbe come principale alleato un validatore *stand-alone* per individuare gli errori definiti nello schema XML, eseguendo la validazione su ognuno dei documenti presenti nel dataset. Dovrebbe quindi utilizzare un editor XML per individuare e correggere gli errori trovati su ciascuno dei file; gli editor specializzati in XML solitamente integrano un validatore e dispongono di diverse funzioni utili a semplificare questo task, che risulta comunque scomodo.

La verifica dei riferimenti sarebbe un compito ancora più tedioso, in quanto, non disponendo di un tool di verifica automatica, dovrebbe essere aperto manualmente ogni documento per individuare eventuali link errati.

Il controllo degli ID è probabilmente il problema maggiore: non avendo a disposizione tecniche generiche per formalizzare tale verifica, l'operatore dovrebbe conoscere con precisione la naming convention e visualizzare manualmente ogni identificatore presente nel documento al fine di valutarlo ed eventualmente correggerlo. Nel caso di un elevato carico di lavoro, inoltre, non sarebbe insolito incorrere in distrazioni ed errori di battitura, data l'assenza di supporto da parte del generico strumento di modifica.

L'ultimo problema si evidenzia nel momento in cui l'ufficio di redazione dei documenti legali desideri implementare dei tool per le modifiche automatiche, specializzati per i propri bisogni: questi potrebbero rischiare di rompere o comunque non essere coerenti con i principi di Akoma Ntoso, portando alla perdita dell'interoperabilità dei documenti.

Alla luce di queste considerazioni è possibile dare un risposta affermativa alla tesi evidenziata nel capitolo 1, per cui *disporre di un metodo per la modifica controllata dei file Akoma Ntoso favorisce lo sviluppo di strumenti informatici per la creazione di documenti di alta qualità*. Il metodo in questione si è concretizzato come libreria software in grado di fornire, tramite API, funzionalità di controllo e di modifica dei metadati presenti nei file XML. Se utilizzata correttamente, essa rende possibile realizzare diverse applicazioni di supporto al lavoro del redattore legale in grado di mantenere la coerenza con i principi dello standard, pur adattandosi ai bisogni particolari degli utenti in questione. In questo modo è possibile salvaguardare l'interoperabilità dei documenti strutturati secondo questo standard universale.

Capitolo 6

Conclusioni

Nel corso di questa dissertazione si è posta l'attenzione sulla fase di perfezionamento dei documenti legali, con particolare attenzione a quelli in formato Akoma Ntoso.

Si è partiti dall'analisi dei dati digitali, processo largamente diffuso e di grande utilità rispetto agli scopi di differenti categorie di utenti. Ci si è quindi focalizzati sulla disciplina dell'informatica giuridica, introducendo una particolare forma di dati digitali: i documenti strutturati, utilizzati per rappresentare i testi legali incorporando diversi metadati utili per l'analisi e l'elaborazione da parte di sistemi software. Essi possono essere rappresentati efficacemente con uno dei dialetti XML per documenti legali. Tra questi il formato Akoma Ntoso si contraddistigue in quanto pensato e progettato per essere uno standard universale, grazie ad una flessibilità tale da poter essere utilizzato in ogni diverso contesto giuridico del mondo, pur fornendo una visione omogenea sui metadati.

Ho potuto osservare che, per favorire l'approccio di nuovi utenti allo standard e quindi la sua diffusione, sono necessari strumenti specializzati e di semplice utilizzo; questa necessità è particolarmente sentita nella fase di correzione e perfezionamento di quei documenti creati tramite un processo di conversione automatico, che per sua natura non potrà mai essere perfetto. Ho pensato quindi che *servisse un metodo per la modifica controllata dei file Akoma Ntoso per favorire lo sviluppo di strumenti informatici per la creazione di documenti di alta qualità.* Strumenti di questo tipo sarebbero utilizzati

dal cosiddetto "redattore legale", un membro dell'ufficio addetto alla stesura dei documenti legali in formato digitale, per assicurare la bontà semantica dei documenti e sfruttare appieno le qualità dello standard.

Ho realizzato quindi un pacchetto software in linguaggio javascript, denominato Akomando-Create, che estende la libreria già esistente Akomando con operazioni in scrittura. Akomando-Create fornisce delle API per la validazione, il controllo di ID e riferimenti interni al documento e la modifica controllata del DOM. Al momento questo software è sottoposto alla *community* open-source di Akomando per essere poi rilasciato su *npm*.

Per dimostrare l'efficacia del pacchetto ai fini della tesi ho implementato ANANAS, un'applicazione web che sfrutta le sue funzionalità. Questa demo fornisce operazioni utili agli scopi del redattore legale: accedere ai documenti presenti su un database, visualizzare grafici sulle loro caratteristiche principali, mostrare gli errori presenti su ciascuno e correggere velocemente alcuni di essi.

Akomando-Create è uno strumento nuovo, su cui sarà possibile aggiungere nuove funzionalità o raffinare quelle esistenti in base ai bisogni dei suoi utilizzatori. Nel corso del progetto ho riscontrato la possibilità di nuovi miglioramenti del software, elencati in seguito.

Ottimizzazione delle operazioni di modifica. Come spiegato nella sezione "Funzionalità di Akomando-Create" (sezione 4.2), le operazioni di modifica del documento generano un nuovo oggetto Akomando, inserito in una cronologia delle modifiche fatte. Questo permette di preservare la correttezza degli invarianti interni della libreria Akomando e fornisce banalmente operazioni di backtracking, ma porta con sé un costo computazionale non indifferente per documenti di grandi dimensioni o per numerose modifiche. Si potrebbe pensare quindi di ottimizzare queste funzioni implementando una *history* più efficiente che memorizzi solamente gli elementi modificati di volta in volta (con numerosi accorgimenti per mantenere la correttezza delle funzioni interne di Akomando e realizzare operazioni di backtrack). Potrebbe anche essere utile serializzare tale *history* su di un file per poi riprendere la sessione di modifica in un secondo momento.

Raffinamento e personalizzazione delle funzioni di controllo. Attualmente sono presenti solo controlli relativi a tre tipi di errori: di validazione, di ID e di riferimenti interni. Sarebbe interessante poter definire nuove regole di "buona scrittura" sul documento, derivate dalle *best practice* emesse dalle istituzioni, in modo che ognuna possa formalizzarle facilmente e verificarle all'interno di Akomando-Create. Al momento ciò si può fare costruendo uno schema XML *custom* che al suo interno importa lo schema ufficiale di Akoma Ntoso, anche se questa pratica non è incoraggiata.

Funzionalità relative alla collezione. Akomando-Create, così come Akomando, è una libreria *document-centered*, cioè fornisce le funzionalità relative ad un singolo documento. Sarebbe invece utile disporre di operazioni che tengono conto dell'insieme di documenti Akoma Ntoso e delle loro relazioni. Ad esempio sarebbe possibile realizzare una funzione che verifica l'esistenza di un documento a cui si fa riferimento, o di un ID interno ad esso. In questo modo si potrebbero individuare errori sui riferimenti esterni tra documenti, oltre che accedere facilmente a funzionalità di Network Analysis sull'insieme di leggi Akoma Ntoso.

Con questi miglioramenti si incrementerebbe maggiormente il supporto ai doveri del redattore legale, realizzando applicazioni specializzate in grado di interagire con intere collezioni di documenti in maniera semplice, efficace, corretta ed eventualmente personalizzata per il contesto giuridico specifico. Questa è la mia speranza per il futuro: così infatti si darebbe pieno compimento agli obiettivi con cui è nato Akomando-Create.

Riferimenti

[AT18] AgID, Team Digitale. "Piano di sviluppo DAF." `https://docs.italia.it/media/pdf/daf-piano-di-sviluppo/bozza/daf-piano-di-sviluppo.pdf`, 8 novembre 2018

[BCD10] G. Barabucci, L. Cervone, A. Di Iorio, M. Palmirani, S. Peroni, F. Vitali. "Managing semantics in XML vocabularies: an experience in the legal and legislative domain", *Proceedings of Balisage: The markup conference.* Vol. 5. 2010.

[BFP08] M. Biasiotti, E. Francesconi, M. Palmirani, G. Sartor, F. Vitali. "Legal informatics and management of legislative documents." *Global Center for ICT in Parliament Working Paper 2.* 2008.

[BFST03] C. Biagioli, E. Francesconi, P. Spinosa, M. Taddei. "The NIR project: Standards and tools for legislative drafting and legal document web publication." *Proceedings of ICAIL workshop on e-government: modelling norms and concepts as key issues*. 2003.

[BPS97] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. "Extensible Markup Language (XML)." *World Wide Web Journal 2.4: 27-66*. 1997.

[BWV08] A. Boer, R. Winkels, and F. Vitali. "MetaLex XML and the Legal Knowledge Interchange Format." *Computable models of the law. Springer, Berlin, Heidelberg*, 21-41. 2008.

[Cer13] L. Cervone. "Parametric editors for structured documents". *AMS Laurea*. 2013.

[CRD87] J. Coombs, A. Renear, S. DeRose. "Markup systems and the future of scholarly text processing." *Communications of the ACM 30.11: 933-947*. 1987.

[Eco10] The Economist. "Data, data everywhere". *The Economist*, `https://www.economist.com/special-report/2010/02/25/data-data-everywhere`. 2010.

[Few06] S. Few."Information dashboard design". *O'Reilly Media*. 2006.

[Few13] S. Few, "Data Visualization for Human Perception". *The Encyclopedia of Human-Computer Interaction, 2nd Ed*. The Interaction Design Foundation. 2013.

[FP16] F. Faini, M. Palmirani. "Italian Open and Big Data Strategy", *International Conference on Electronic Government and the Information Systems Perspective.* Springer, Cham, 2016.

[GH15] A. Gandomi, M. Haider. "Beyond the hype: Big data concepts, methods, and analytics." *International Journal of Information Management 35.2 (2015): 137-144*. 2015

[GV13] D. Gultemen, T. van Engers. "Graph-based Linking and Visualization for Legislation Documents (GLVD)." *Workshop "Network Analysis in Law"*. 2013.

[Har18] J. Hartikainen, "Unit Test Your JavaScript Using Mocha and Chai", `https://www.sitepoint.com/unit-test-javascript-mocha-chai/`, January 12, 2018

[HGB12] T. M. Harrison, S. Guerrero, G. B. Burke, M. Cook, A. Cresswell, N. Helbig, J. Hrdinova, T. Pardo. "Open government and e-government: Democratic challenges from a public value perspective". *Information Polity, 17:2, 83-97*. 2012.

[Hoe13] R. Hoekstra. "A Network Analysis of Dutch Regulations Using the MetaLex Document Server", *Workshop "Network Analysis in Law"*. 2013.

[Lan01] D. Laney. "3D data management: Controlling data volume, velocity and variety." *META group research note 6.70 (2001): 1.* 2001.

[Mon18] D. Montesi, "Dati semi-strutturati e non strutturati", `http://cs.unibo.it/~montesi/CBD/01IntroModelli.pdf`, visited on 29/10/2018.

[Pal12] M. Palmirani. "Legislative XML: Principles and technical tools", ed. Aracne, 2012.

[PB07] M. Palmirani, F. Benigni. "Norma-system: A legal information system for managing time." *Proceedings of the V legislative XML workshop.* 2007.

[Ric06] B. Richmond. "Introduction to Data Analysis Handbook." *Academy for Educational Development.* 2006.

[Sar16] G. Sartor. "L'informatica giuridica e le tecnologie dell'informazione: Corso di informatica giuridica (Vol. 2)". G Giappichelli Editore. 2016.

[SMR12] C. Snijders, U. Matzat, U. Reips. "Big Data: Big Gaps of Knowledge in the Field of Internet Science." *International Journal of Internet Science 7.1 (2012): 1-5.* 2012.

[Trn14] A. Trnka. "Big data analysis." *European Journal of Science and Theology 10.1: 143-148.* 2014.

[Tuk62] J. Tukey. "The future of data analysis", *The annals of mathematical statistics* 33.1: 1-67. 1962.

[Tuk80] J. Tukey. "We need both exploratory and confirmatory." *The American Statistician 34.1 (1980): 23-25.* 1980

[VPP17] F. Vitali, M. Palmirani, and V. Parisse. "Akoma Ntoso Naming Convention Version 1.0". *OASIS Committee Specification 01.* `http://docs.oasis-open.org/legaldocml/akn-nc/v1.0/cs01/akn-nc-v1.0-cs01.html`. 2017.

[Wik18] Data Analysis - Wikipedia, `https://en.wikipedia.org/wiki/Data_analysis`, visited on 23/10/2018.

[Wik18a] Data Cleansing - Wikipedia, `https://en.wikipedia.org/wiki/Data_cleansing`, visited on 30/10/2018.

Elenco delle figure

Listings

Appendice A

Codice sorgente per la creazione dei grafici

In seguito sono riportati per completezza i frammenti di codice relativi alla costruzione dei tre grafici descritti in 5.2.

Listing A.1: Implementazione del grafico I.

```
var slices = [
    model.docList.filter((doc)=> doc.info.valid ===
        false).length,
    model.docList.filter((doc)=> doc.info.valid ===
        true && doc.akomandoCreator.checkIds().length >
        0).length,
    model.docList.filter((doc)=> doc.info.valid ===
        true && doc.akomandoCreator.checkIds().length ===
         0).length,
];

model.chart = new Chart(context, {
    type: 'pie',
    data: {
        datasets: [{
            data: slices,
            backgroundColor: [
```

```
                'rgba(245, 65, 0, 0.8)',
                'rgba(200, 165, 0, 0.8)',
                'rgba(0, 120, 50, 0.8)',
            ],
        }],
        labels: [
            'Invalid',
            'Valid with bad IDs',
            'Valid with good IDs'
        ]
    },
    options: {
        responsive: true
    }
});
```

Listing A.2: Implementazione del grafico II.

```
model.docList.forEach((doc) =>
    doc.info.totErrors = doc.akomandoCreator.checkIds()
        .length + doc.akomandoCreator.checkReferences().
        length + doc.info.validErrors.length
);

var sortedList = _.sortBy(model.docList, ((doc) =>
    - doc.info.totErrors          // the minus is because
        we want descenting ordering
)).slice(0, 20);          // top 20

var bars = [
    sortedList.map((doc)=> doc.akomandoCreator.checkIds
        ().length),
    sortedList.map((doc)=> doc.akomandoCreator.
        checkReferences().length),
    sortedList.map((doc)=> doc.info.validErrors.length)
```

```
];

let labels = sortedList.map((doc, index) => doc.
    manifestation);

model.chart = new Chart(context, {
type: 'horizontalBar',
data: {
    labels: labels,
    datasets: [{
        label: "ID errors",
        data: bars[0],
        backgroundColor: 'Yellow',
        borderColor: 'rgba(170, 160, 0, 1)'
    },
    {
        label: "Ref. errors",
        data: bars[1],
        backgroundColor: 'Violet',
        borderColor: 'rgba(160, 0, 160, 1)'
    },
    {
        label: "Valid. errors",
        data: bars[2],
        backgroundColor: 'Red',
        borderColor: 'rgba(180, 0, 0, 1)'
    }],
},
options: options
});
```

Listing A.3: Implementazione del grafico III.

```
let allRefs = model.docList.map((doc) =>
```

```
    doc.akomandoCreator.getAkomando().getReferencesInfo
        ().references.list);
allRefs = _.flatten(allRefs);
const groupedRefs = _.groupBy(allRefs, (ref) => ref.
    href)
const keys = Object.keys(groupedRefs)
    .sort((x,y)=> groupedRefs[y].length - groupedRefs[x
        ].length)
    .slice(0,20);

model.groupedRefs = groupedRefs;

var bars = keys.map((k) => groupedRefs[k].length);

model.chart = new Chart(context, {
    type: 'horizontalBar',
    data: {
        labels: keys,
        datasets: [{
            label: "Num. of citing docs",
            data: bars,
            backgroundColor: 'rgba(0, 16, 245, 0.6)',
            borderColor: 'rgba(0, 12, 160, 1)'
        }]
    },
    options: options
});
```