Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Fmx2Akn: un aggregatore per la fruizione ed elaborazione automatica di documenti legali multiformato sulla normativa europea

Relatore:
Prof. Fabio Vitali

Presentata da:
Vito Vitale

Correlatori:
Prof. Monica Palmirani
Dott. Luca Cervone

Sessione III
Anno Accademico 2015/2016

"Mio padre venne a prendermi a scuola un giorno, giocammo a hockey e poi andammo in spiaggia. Faceva troppo freddo per fare il bagno, quindi ci sedemmo sull'asciugamano a mangiare pizza. Arrivato a casa, le mie scarpe erano piene di sabbia, e le rovesciai sul pavimento della mia stanza. Non sapevo di sbagliare, avevo solo sei anni. Mia madre mi sgridò per il casino, ma lui non si arrabbiò. Mi disse che milioni di anni prima il moto della Terra e degli oceani portarono quei granelli di sabbia in quella spiaggia, e io l'avevo portata via. Mi ripeteva ogni giorno che stavamo cambiando il mondo. Ed è un bel pensiero, finché non penso a quanti giorni e quante vite mi servirebbero per portare una scarpa piena di sabbia a casa prima di svuotare la spiaggia, prima di fare la differenza per qualcuno: cambiamo il mondo quotidianamente, ma per riuscire a cambiare il mondo in maniera significativa, serve più tempo di quanto la maggior parte di noi abbia. È lento, è metodico, è estenuante: non tutti abbiamo abbastanza fegato per farlo."

Elliot Alderson - Mr. Robot S01E05

Indice

Capitolo 1

Introduzione

Il seguente progetto di tesi ha lo scopo di investigare e migliorare la distribuzione dei documenti legali. Tale ricerca rientra nell'ambito dell'*informatica giuridica*, che è la disciplina che applica le scienze informatiche ai contesti giuridici, migliorando il lavoro del giurista (questo ramo è chiamato *informatica del diritto*), e regolamenta l'introduzione nella società delle nuove tecnologie, e gli effetti da queste prodotte (questo secondo ramo prende il nome di *diritto dell'informatica*).

La redazione di questa dissertazione è avvenuta a seguito del completamento del progetto di tesi, che è consistito nella creazione di un prototipo di portale web per la fruizione ed elaborazione di documenti legali, che prende il nome di **Fmx2Akn**. La sigla "Fmx" è l'abbreviazione di *FORMEX*, la sigla "Akn" è l'abbreviazione di *AKOMA NTOSO*, invece il "2" al centro indica il ruolo di convertitore (in inglese il numero 2 si pronuncia come la parola "to", ovvero "da qualcosa verso qualcosa"). Da quanto appena detto si evince che il lavoro da svolgere è il seguente:

1. devo prendere questi documenti in un certo formato da qualche parte;
2. devo trasformare questi documenti in qualche modo;
3. devo riconsegnarli all'utente o alla macchina trasformati in qualche modo.

Prima di concentrare il discorso sul progetto di tesi, è necessario introdurre alcuni concetti.

AKOMA NTOSO, una locuzione che in lingua Akan (una lingua parlata nella parte ovest dell'Africa) significa "cuori legati", è il nome di un insieme di specifiche tecniche redatte e sviluppate all'interno del progetto "Africa i-Parliament Action Plan", un progetto promosso dal Dipartimento per l'Economia e gli Affari Sociali delle Nazioni Unite, con lo scopo di aiutare i parlamenti africani ad assolvere al meglio le loro funzioni democratiche tramite l'utilizzo di tecnologie aperte [Cer13], e più in generale per favorire l'utilizzo di tali tecnologie nei processi legislativi dei paesi africani. Il progetto è stato ideato per sopperire alle seguenti necessità: sviluppare uno standard comune ed aperto per lo scambio di dati fra diversi parlamenti; definire una struttura base di documento legale sulla quale possono essere costruiti i sistemi legislativi; realizzare un meccanismo per i riferimenti incrociati e le citazioni fra documenti legali creati dagli stessi organi oppure fra organi di stati diversi [VZ07].

Una parte fondamentale delle specifiche tecniche AKOMA NTOSO è composta dalla definizione di un nuovo "dialetto XML per il markup di documenti legali in formato digitale". Questo dialetto è in grado di poter descrivere un'ampia gamma di documenti legali, che vanno dalla legislazione primaria ai documenti giudiziari, alle gazzette ufficiali, e così via, con la peculiarità di poter adattare il dialetto alle differenti necessità di specifici sistemi legislativi[VZ07]. I nomi FORMEX ed AKOMA NTOSO quindi indicano, in questo contesto, due dei tanti dialetti XML per il markup di documenti legali (FORMEX è stato creato dall'Ufficio delle Pubblicazioni dell'Unione Europea, e viene attualmente utilizzato per il markup di tutti i documenti legali dell'UE).

Per comprendere al meglio lo scopo di questo progetto di tesi, è opportuno soffermarsi ed analizzare quanto scritto tra doppi apici nel paragrafo precedente: dialetto XML per il markup di documenti legali in formato digitale.

Si parte dalla definizione di documento strutturato: un documento in formato digitale che racchiude, oltre l'effettivo contenuto del documento, delle informazioni addizionali; queste informazioni possono essere di tipo logico (riguardano il significato del contenuto del documento), di layout (descrivono la struttura della pagina dal punto di vista editoriale) e fisiche (riguardano lo specifico adattamento della struttura ad una forma fisica finale di rappresentazione) [Cer13].

Per poter integrare queste informazioni addizionali al documento è opportuno utilizzare degli appositi linguaggi detti di *markup* (termine inglese che in italiano si può tradurre con "marcatura"): XML è un linguaggio di markup, nato nel 1996 proprio dal Consorzio del World Wide Web (W3C). Uno dei principali aspetti positivi dell'utilizzo di XML sta nella sua natura di essere una tecnologia aperta e neutrale: col primo aggettivo, che è una semplice traduzione della locuzione inglese *open-source*, si indica una tecnologia con specifiche tecniche visionabili a tutto il pubblico, e soprattutto libera da qualsiasi concetto di *copyright* (chiunque è libero di utilizzare XML per qualsiasi scopo senza preoccuparsi della detenzione dei diritti d'uso da parte di un soggetto terzo); il secondo aggettivo (neutrale) è quasi una conseguenza del primo, in quanto la natura "open-source" si rispecchia anche sulla semplicità e assenza di vincoli nel creare applicazioni in grado di poter manipolare dei documenti che utilizzano il linguaggio XML, su qualsiasi tipo di piattaforma e dispositivo. È evidente che questi fattori giochino un ruolo importante nel contesto legale, in quanto un sistema legislativo volto alla libera fruizione dei documenti da parte del cittadino non deve essere legato ad una specifica applicazione e a dei vincoli d'uso per il normale svolgimento dei suoi compiti [PV12].

Un'altra peculiarità del linguaggio XML consiste nella possibilità di poter regolamentare il contenuto di un documento tramite la definizione ed adozione di apposite grammatiche che stabiliscono gli elementi che compongono il documento, aggiungendo dei vincoli riguardanti, ad esempio, il tipo di informazione contenuta in un certo elemento, oppure l'ordine di utilizzo degli

elementi in un documento, o i collegamenti fra i vari elementi da un punto di vista gerarchico, e così via. Queste grammatiche costituiscono il concetto di "dialetto XML", un concetto molto importante, tramite il quale è quindi possibile adattare il linguaggio XML a qualsiasi tipo di esigenza e contesto.

Data la diversità delle tradizioni giuridiche e dei sistemi parlamentari da paese a paese, spesso ogni nazione ha intrapreso l'iniziativa della creazione di un proprio dialetto XML per la rappresentazione dei propri documenti legali. Ad esempio, oltre FORMEX e AKOMA NTOSO, in questa dissertazione viene anche analizzato NormeInRete, il dialetto XML utilizzato per la rappresentazione dei documenti legali prodotti dagli organi legislativi italiani. Tuttavia, durante il processo della definizione di un dialetto, è opportuno seguire una serie di accorgimenti: realizzare un'architettura basata su modelli astratti, in quanto è possibile ottenere un maggior grado di flessibilità, applicando un insieme di regole su una certa classe di documenti ed altre regole su altre classi; definire una robusta nomenclatura, pienamente espressiva e invariante nel tempo e con lo sviluppo di nuove tecnologie; distinguere i diversi tipi di informazioni addizionali (logiche, di layout e fisiche) e la provenienza di tali informazioni (fonti ufficiali e non ufficiali) [PV12].

Il dialetto XML AKOMA NTOSO è stato progettato tenendo in considerazione tutti gli accorgimenti appena citati. È opportuno, quindi, promuovere questo dialetto affinché venga utilizzato dal maggior numero di sistemi parlamentari, in quanto la diffusione di uno standard comune rappresenta il più importante strumento di semplificazione nello scambio e nel riferimento di documenti fra diversi parlamenti e organi della stessa nazione.

Una volta individuata ed utilizzata la tecnologia opportuna per la rappresentazione dei documenti legali, è necessario discutere l'accesso e lo scambio di tali documenti. A tal fine vengono sviluppati portali per la distribuzione, ovvero siti web che siano in grado non solo di gestire un archivio di tali documenti, ma anche di catalogarli in base alle informazioni aggiuntive disponibili per ciascun documento, e di agevolare l'utente nel processo di ricerca.

Attualmente, la maggior parte degli stati si è adattata a questo tipo di

esigenza: il cittadino ha il diritto di poter visionare e studiare la legge vigente a cui è soggetto. Per il mio progetto di tesi ho analizzato nello specifico tre portali, ponendo l'attenzione su alcune lacune di usabilità: *Normattiva*, il portale ufficiale dello stato italiano, *Legislation.gov.uk*, il portale ufficiale del Regno Unito, e infine *EUR-Lex*, il portale ufficiale dell'Unione Europea. A partire dall'ultimo, ho progettato ed implementato un prototipo di portale che potesse supportare la distribuzione delle *direttive europee*, una delle fonti secondarie del diritto comunitario[1]

Le direttive europee, assieme a tutti i documenti legali prodotti dall'UE, sono archiviate sul sito EUR-Lex, utilizzando il dialetto XML FORMEX. EUR-Lex fornisce ai suoi utenti un'ampia gamma di funzionalità, in quanto, a differenza dei portali creati per ogni singolo stato, EUR-Lex deve poter mettere a disposizione tutta la sua documentazione nelle 23 lingue ufficiali parlate nell'UE. Sono presenti diversi strumenti di ricerca e diversi metodi per catalogare il vasto bacino di documenti a disposizione. Dal punto di vista dell'usabilità[2], il portale presenta alcune problematiche:

1. non è possibile recuperare dal sito un documento nel formato tecnico XML, ma solo in formati presentazionali finalizzati alla stampa;

2. non è possibile recuperare direttamente un singolo file tramite indirizzo Web, senza aver ricercato il documento ed essere arrivati alla pagina dei risultati;

3. la consultazione del sito tramite dispositivo mobile non è molto comoda, in quanto l'interfaccia non si adatta alla dimensione dello schermo del dispositivo dell'utente;

4. il sito presenta frequenti rallentamenti.

[1]Le altri fonti secondarie del diritto comunitario sono i regolamenti e le decisioni; i trattati rappresentano invece una fonte primaria.

[2]L'usabilità è definita come l'efficacia, l'efficienza e la soddisfazione con le quali determinati utentin raggiungono determinati obiettivi in determinati contesti

Per quanto riguarda il punto 1, è molto importante fornire l'accesso al formato tecnico, in quanto grazie ad esso anche un normale utente può comprendere al meglio non solo la materia ed il soggetto del documento, ma anche come questo è suddiviso e che tipo di valore e significato hanno i singoli elementi all'interno di esso; in merito al punto 2, qualsiasi archivio online dovrebbe fornire questo tipo di servizio (in informatica prende il nome di *API REST*), in quanto ci si è distaccati dal classico modello web in cui il sito rappresenta l'elemento centrale, e piuttosto bisogna puntare alla libera e agevole circolazione delle risorse [Fie00](anche un documento legale è una risorsa); per quanto riguarda il punto 3, al giorno d'oggi è importante progettare un sito tenendo in considerazione la sua utilizzabilità su un dispositivo mobile, dato che questi dispositivi rappresentano una enorme fetta del mercato tecnologico.

Nella realizzazione del prototipo di portale **Fmx2Akn** sono state prese in considerazione tutte le lacune citate precedentemente; vengono utilizzate tecnologie moderne e aperte per fornire un servizio rapido, efficiente e semplice da utilizzare. Il prototipo si occupa inizialmente di reperire le direttive europee in modo automatico dal sito EUR-Lex, recuperando tutti i formati di file disponibili per ognuna delle 23 lingue digitando semplicemente l'*id CELEX*, un identificatore univoco per i documenti legali prodotti dall'UE; il risultato della ricerca viene restituito all'utente in un'interfaccia pulita, non troppo complessa, e che si adatta alle dimensioni dello schermo del dispositivo dell'utente (questa pratica, in informatica, prende il nome di *responsive web design*); inoltre, il prototipo è in grado di colmare le altre lacune descritte in precedenza, in quanto è possibile sia recuperare una direttiva europea nel suo formato tecnico XML FORMEX, che poter reperire un singolo file oppure un insieme di file che soddisfano un certo requisito (di lingua o di formato) tramite una semplice digitazione nella barra dell'indirizzo che include l'identificativo CELEX e i parametri lingua e formato. Infine, per quanto riguarda la questione dei frequenti rallentamenti, ho utilizzato un web server moderno, veloce e aperto, chiamato *Node.js*, nato nel 2009, che presenta una serie di

vantaggi rispetto ai classici web server utilizzati per i siti web, il primo fra tutti la velocità nel poter soddisfare le richieste, nettamente superiore se paragonata alle tecnologie classiche più datate [LMT14].

Il prototipo **Fmx2Akn**, tornando alle funzionalità principali a cui dovrebbe assolvere, descritte nell'elenco numerato all'inizio di questa introduzione, è in grado di risolvere due dei tre punti: riesce a recuperare automaticamente tutte le istanze di file necessarie, incluso il formato tecnico FORMEX, quindi mette a disposizione il formato per la conversione; inoltre, il prototipo è già predisposto per la visualizzazione della conversione, poichè è in grado di mettere in parallelo le due istanze FORMEX e AKOMA NTOSO dello stesso documento, evidenziando fra due formati tecnici la corrispondenza degli elementi. Gli sviluppi futuri di questo progetto convoglieranno sicuramente nella creazione di una fase di conversione dal formato FORMEX ad AKOMA NTOSO.

Questa introduzione analizza brevemente tutte le tecnologie che vengono trattate in modo più approfondito nei prossimi capitoli di questa dissertazione, soffermandosi anche sul motivo della realizzazione del progetto di tesi. Procedendo con ordine, nel secondo capitolo vengono illustrati i documenti strutturati e il linguaggio XML, utilizzato per il markup in qualsiasi contesto (vengono analizzati tre esempi di utilizzo in contesto non giuridico e altri tre esempi in contesto giuridico); il terzo capitolo si occupa della distribuzione dei documenti legali, analizzando tre portali attualmente attivi e funzionanti, rispettivamente dell'Italia, del Regno Unito e dell'UE, accennando ai problemi di usabilità riscontrati nell'ultimo (EUR-Lex); nel quarto capitolo vengono proposte delle tecnologie per la risoluzione di tali problemi, che costituiscono lo scheletro prototipo di portale, la cui implementazione viene descritta nel dettaglio nel quinto capitolo.

Capitolo 2

I documenti strutturati nell'informatica giuridica

Questo primo capitolo è completamente incentrato sul linguaggio XML (*eXtensible Markup Language*). Inizio dando una chiara definizione del concetto di "Documento strutturato" nella prima sezione, parlando poi in dettaglio di XML (ottimo e noto linguaggio per il markup di documenti strutturati) ed illustrando alcuni dei suoi dialetti utilizzati per tale scopo, nella seconda sezione. Nella terza sezione, invece, entro nel dettaglio dell'ambito legale, per quanto riguarda i documenti strutturati, analizzando i principali obiettivi a cui dovrebbe assolvere un dialetto XML utilizzato per tale scopo, ed approfondendo il mio studio con l'analisi delle caratteristiche di tre dialetti XML nell'ambito legale: *FORMEX*, il dialetto attualmente in uso per il markup dei documenti contenuti nel sito ufficiale dell'Unione Europea; *NormeInRete*, il dialetto creato ed utilizzato appositamente per la marcatura delle leggi contenute nel database ufficiale dello Stato Italiano; *AKOMA NTOSO*, un dialetto sviluppato nell'ambito di un progetto delle Nazioni Unite, che può essere utilizzato per il markup di documenti legali provenienti dalle più differenti legislazioni globali. Concludo questo capitolo facendo una rapida introduzione ai portali ufficiali che contengono tali documenti legali strutturati.

2.1 Documenti strutturati

Un documento strutturato è un documento elettronico che ingloba, oltre al suo effettivo contenuto, delle informazioni addizionali definite da uno specifico schema[1]. Queste informazioni vengono aggiunte tramite l'utilizzo di uno specifico linguaggio di markup, ed hanno lo scopo di descriverne il contenuto.

Le informazioni aggiuntive servono a descrivere ulteriori caratteristiche del documento: le caratteristiche logiche, di layout e fisiche [Cer13]. Le caratteristiche logiche descrivono gli oggetti che compongono il documento e le relazioni fra loro; queste relazioni sono generalmente di natura gerarchica (ad esempio, un documento può includere un titolo, un autore, un sommario, una sequenza di capitoli; ma anche un capitolo può includere un titolo, un insieme di sezioni, e così via). Le caratteristiche di layout, come suggerisce la parola stessa, contengono informazioni riguardanti il layout del documento, ad esempio il formato della pagina, il testo specifico che costituisce un blocco, l'allineamento di tale testo nel blocco, l'enfasi da aggiungere al testo come la sottolineatura, il grassetto, eccetera. Le caratteristiche fisiche, infine, riguardano l'output finale del documento. Possono esserci più strutture fisiche del documento (poichè possono essere numerosi i tipi di output a cui il documento è destinato), ma c'è sempre e solo un'unica struttura sia logica che di layout [Mar92].

I linguaggi di markup utilizzati agli scopi di questo progetto di tesi sono linguaggi di markup utili a descrivere la struttura logica dei documenti. Uno dei linguaggi di markup utile alla descrizione della struttura logica dei documenti è il linguaggio di markup XML, che analizzo nella prossima sezione.

2.2 Il linguaggio di markup XML

XML (eXstensible Markup Language) è un linguaggio di markup nato nel 1996 da un working group del World Wide Web Consortium (W3C), derivato

[1] Pagina di Wikipedia "Structured Document"

dal più vecchio linguaggio SGML[2], e divenuto un "W3C Recommendation" a partire dal 10 febbraio 1998 [BPSM+08].

Gli obiettivi posti dai creatori di XML erano di realizzare un linguaggio di markup compatibile con SGML, leggibile dagli umani e da un elaboratore, facile da scrivere ed utilizzare e che permettesse l'interscambio di dati in internet [BPSM+08].

XML è una tecnologia aperta, "royalty-free", ovvero il suo sviluppo è documentato da documenti pubblici, le sue specifiche sono pubbliche, e chiunque, da un privato ad un ente commerciale o pubblico, è libero di utilizzarla senza pagare alcun tipo di licenza. Da questa tecnologia nasce anche l'omonimo formato file XML, aperto anch'esso, con estensione .xml .

Un documento XML inizia con una dichiarazione di tipo, ovvero una stringa che indica che stiamo utilizzando il linguaggio XML e ne specifica la versione utilizzata. Gli oggetti contenuti nel documento sono chiamati *elementi*; il costrutto che viene utilizzato per il markup è chiamato "tag". Per ogni elemento deve esserci un tag di apertura ed uno di chiusura; quello di chiusura differisce da quello di apertura per la presenza di una "slash" subito dopo la parentesi angolata aperta e prima del nome dell'elemento. L'elemento potrebbe anche essere vuoto, ovvero senza alcuna apertura e chiusura del tag. Gli elementi possono essere annidati fra loro, e l'elemento principale che racchiude tutti gli altri è chiamato "radice" (*root*). L'attributo è un altro costrutto di markup, viene utilizzato per specificare le proprietà degli elementi; esso è costituito da una coppia nome-valore; il valore deve essere scritto fra doppi apici, e non è possibile inserire più di una volta lo stesso attributo in un tag.

Listing 2.1: Esempio basilare di sintassi XML

```
<tag attributo="valore">contenuto</tag>
```

[2]Standard Generalized Markup Language è un metalinguaggio definito come standard ISO nel 1986 avente lo scopo di definire linguaggi da utilizzare per la creazione di documenti machine-readable.

Per regolamentare gli elementi contenuti in un documento XML vengono definite delle apposite grammatiche tramite *schema*, utilizzando specifici linguaggi (ad esempio DTD, XSD, Relax-NG, eccetera). Grazie a queste grammatiche è possibile[3]:

- definire un vocabolario contenente tutti gli elementi con i relativi attributi che possono essere utilizzati;
- associare dichiarazioni di tipo ad elementi ed attributi;
- specificare dove possono apparire elementi ed attributi nel documento e come sono collegati fra di loro in modo gerarchico (ad esempio come possono essere annidati gli elementi fra di loro).

Un documento XML, per essere definito "corretto" e con assenza di errori, passa due tipi di controlli. Il primo è un controllo sintattico, ovvero non ci devono essere irregolarità in base a quanto definito dalle specifiche stesse di XML (ad esempio, deve contenere solo caratteri unicode, i caratteri speciali di sintassi non devono apparire nel contenuto ma solo nel loro effettivo ruolo di markup, i tag devono essere correttamente annidati evitando l'overlapping[4], il nome del tag deve coincidere fra l'apertura e la chiusura dello stesso, e così via); un documento che supera questo primo controllo è definito come "ben formato". Il secondo controllo, invece, riguarda la grammatica a cui il documento è associato. Se il documento rispetta tutte le regole e i vincoli imposti dalla grammatica, allora può essere definito "valido".

In base a quanto specificato, possiamo definire XML come un metalinguaggio, ovvero un linguaggio grazie al quale possiamo definire nuovi "linguaggi marcatori", dei dialetti che possono essere utilizzati per il markup di documenti in specifici domini; inoltre, grazie a questa sua importantissima funzione, XML è stato adottato per la definizione di centinaia di formati di

[3]Estratto da www.w3.org/standards/xml/schema

[4]Nel procedimento di markup, l'overlapping avviene quando ci sono delle strutture che interagiscono fra di loro in modo non gerarchico, rendendo impossibile la rappresentazione del documento tramite una struttura ad albero

documento. Nelle prossime sezioni analizzerò alcuni esempi dell'utilizzo di questo linguaggio.

2.2.1 Esempio 1: GraphML

GraphML (Graph Markup Language) è un formato di file aperto dedicato all'archiviazione dei grafi[5]. È stato ideato e realizzato da una commissione del Graph Drawing (graphdrawing.org), con l'intento di creare un formato semplice da comprendere sia da un umano che da un computer, che sia in grado di rappresentare qualsiasi modello di grafo senza restrizioni (grafi gerarchici, ipergrafi, e così via), a cui sia semplice aggiungere informazioni addizionali (ad esempio per l'utilizzo in applicazioni specifiche) senza interferire con la struttura base del grafo o con altre informazioni addizionali [BELP13].

Descrivo brevemente gli elementi principali del documento GraphML. L'elemento radice di questo tipo di documento è chiamato `<graphml>`, che specifica la grammatica e il namespace, e può contenere un numero arbitrario di grafi, indicati col tag `<graph>`. Nodi ed archi vengono rispettivamente descritti con gli elementi `<node>` ed `<edge>`. Ogni grafo, nodo ed arco deve avere un id univoco. Per rappresentare un arco, vengono utilizzati gli attributi source e target nel tag `<edge>`. È possibile aggiungere informazioni addizionali agli elementi di un grafo (ad esempio il colore del nodo oppure il peso di un arco) tramite il meccanismo *data/key*: l'elemento `<key>` è dichiarato fuori dal grafo, possiede un id univoco, ed indica il nome dell'informazione addizionale, il tipo, l'elemento a cui è rivolta ed un valore di default; l'elemento `<data>`, invece, è l'effettiva rappresentazione dell'informazione, e viene inserito all'interno del tag a cui è rivolto[6].

Listing 2.2: Grafo con due nodi colorati ed un arco pesato.

```
<?xml version="1.0" encoding="UTF-8"?>                          1
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"          2
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"       3
```

[5] http://graphml.graphdrawing.org/

[6] Estratto da [BELP13]

```
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
   /1.0/graphml.xsd">
  <key id="d0" for="node" attr.name="color" attr.type="string
   ">
    <default>blue</default>
  </key>
  <key id="d1" for="edge" attr.name="weight" attr.type="
   double"/>
  <graph id="G" edgedefault="undirected">
    <node id="n0">
      <data key="d0">red</data>
    </node>
    <node id="n1"/>
    <edge id="e1" source="n0" target="n1">
      <data key="d1">2.5</data>
    </edge>
  </graph>
</graphml>
```

2.2.2 Esempio 2: RSS

RSS (Rich Site Summary)[7] è un formato molto popolare per la distribuzione di contenuti nel web, ed è un dialetto XML[8]. Viene principalmente utilizzato per la creazione di flussi che permettono di essere constantemente aggiornati sulla pubblicazione di nuovo contenuto da parte di un sito/blog senza avere la necessità di doverlo visitare manualmente. RSS definisce una struttura adatta ad aggregare un insieme di notizie, ciascuna contenente delle informazioni basilari, come l'autore, il titolo, il riassunto della notizia, e così via. Tramite software chiamati *RSS reader* è possibile iscriversi ad una serie di feed RSS: da quel momento, il software controllerà periodicamente la presenza di nuovi contenuti nei siti a cui ci si è iscritti, notificandoli all'utente [Mur07]. La prima versione di RSS risale al lontano marzo 1999.

[7] Inizialmente era l'acronimo di RDF Site Summary, spesso viene anche chiamato Really Simple Syndication (wikipedia)

[8] http://www.rssboard.org/rss-specification

Listing 2.3: Esempio di un documento RSS (tratto da Wikipedia)

```
<rss version="2.0">
<channel>
  <title>RSS Title</title>
  <description>This is an example of an RSS feed</description>
  <link>http://www.example.com/main.html</link>
  <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
  <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
  <ttl>1800</ttl>
  <item>
    <title>Example entry</title>
    <description>
      Here is some text containing an interesting description
    </description>
    <link>http://www.example.com/blog/post/1</link>
    <guid isPermaLink="true">7bd204c6-1655-4c27-aeee-53f933c5395f</guid>
    <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
  </item>
</channel>
</rss>
```

Analizzo brevemente la struttura base di un documento RSS. L'elemento radice è appunto il tag `<rss>`, con un attributo che ne specifica la versione utilizzata. Al suo interno ci deve essere almeno un elemento `<channel>`: è l'elemento che racchiude tutte le informazioni di uno specifico sito web. Le informazioni da inserire obbligatoriamente sono: `<title>`, `<link>` e `<description>` (titolo del sito, URL e breve descrizione). Il `<channel>` può contenere un numero arbitrario di elementi `<item>`: l'*item* può rappresentare, ad esempio, il post in un blog, una notizia in un sito di un giornale, e così via; è il contenuto creato dal sito. Tutti i campi dell'*item* sono opzionali, ma è richiesta almeno la presenza del titolo o della descrizione [9].

[9] Estratto da http://www.rssboard.org/rss-specification

2.2.3 Esempio 3: OpenDocument

OpenDocument (ODF - Open Document Format for Office Applications) è un formato di file per l'archiviazione di documenti della produttività d'ufficio (documenti di testo generici, diagrammi, presentazioni, fogli di calcolo). Sviluppato da una commissione tecnica dell'OASIS (Organization for the Advancement of Structured Information Standards), è un formato "royalty-free" basato su XML, divenuto sia uno standard OASIS (1 maggio 2005) che uno standard ISO/IEC (30 novembre 2006) [10].

OpenDocument è essenzialmente composto da un archivio JAR (ZIP) che contiene quattro file principali in formato XML, più eventuali file addizionali da aggiungere al documento (ad esempio immagini).

Come possiamo notare dal Listing 2.4, il grosso vantaggio di OpenDocument è l'applicazione concreta del principio della separazione degli interessi: infatti, guardando i nomi dei file all'interno dell'archivio, è facile intuire la separazione fra l'effettivo contenuto del documento rispetto allo stile da applicare, ai metadata (sull'autore e sul software con il quale è stato creato il documento) e alle specifiche impostazioni software da applicare al momento della visualizzazione del documento. Il tutto è stato compresso in un archivio per rimediare alla verbosità del linguaggio XML [Eis06].

Listing 2.4: Effettivo contenuto di un file ODF

```
vsv@STCZZ ~/Desktop $ unzip -l testfile.odt
Archive:  testfile.odt
  Length      Date    Time    Name
---------  ---------- -----   ----
       39  2016-10-13 16:26   mimetype
      417  2016-10-13 16:26   Thumbnails/thumbnail.png
     3563  2016-10-13 16:26   content.xml
    10320  2016-10-13 16:26   settings.xml
     1049  2016-10-13 16:26   meta.xml
    12079  2016-10-13 16:26   styles.xml
      899  2016-10-13 16:26   manifest.rdf
```

[10] Estratto da http://opendocumentformat.org/

```
        0  2016-10-13 16:26   Configurations2/images/Bitmaps/
        0  2016-10-13 16:26   Configurations2/toolpanel/
        0  2016-10-13 16:26   Configurations2/progressbar/
        0  2016-10-13 16:26   Configurations2/accelerator/
   current.xml
        0  2016-10-13 16:26   Configurations2/floater/
        0  2016-10-13 16:26   Configurations2/statusbar/
        0  2016-10-13 16:26   Configurations2/toolbar/
        0  2016-10-13 16:26   Configurations2/popupmenu/
        0  2016-10-13 16:26   Configurations2/menubar/
     1086  2016-10-13 16:26   META-INF/manifest.xml
---------                     -------
    29452                     17 files
```

2.3 XML per i documenti legali

Come abbiamo visto nelle sezioni precedenti, XML è largamente utilizzato per un vasto numero di scopi. Essendo un ottimo linguaggio di markup, quindi, è perfetto da utilizzare anche per il markup di documenti legali.

Siccome le tradizioni giuridiche sono diverse da paese a paese, anche la struttura ed i contenuti dei documenti legali possono cambiare significativamente. Per creare quindi un solido dialetto basato su XML che sia in grado di assolvere a pieno le funzionalità di markup per documenti legali, è necessario tener conto dei seguenti requisiti da dover rispettare [PV12]:

- il documento legale è il frutto di un particolare iter legislativo che vede coinvolti diversi enti ed istituzioni: per questo motivo i metadati messi a disposizione dal dialetto devono essere il più possibile astratti;
- il documento deve poter essere consultato facilmente e considerato legale anche col passare del tempo: questo significa che, quindi, non deve dipendere da alcuna tecnologia (soprattutto proprietaria);

XML è in grado di soddisfare tutti i requisiti precedenti: come abbiamo visto nelle sezioni precedenti, è un linguaggio "technology-independent",

quindi è possibile creare parser ed editor in qualsiasi linguaggio di programmazione e su qualsiasi piattaforma; è in grado di separare contenuto dai metadati; è capace di dare significati differenti a diverse porzioni dello stesso documento. Per sfruttare a pieno queste caratteristiche, però, il nuovo linguaggio da scrivere (ovvero la nuova grammatica da definire) deve essere appropriato; bisogna seguire quindi i seguenti accorgimenti [PV12]:

- l'architettura di tale linguaggio deve essere basata su modelli astratti, chiamati anche pattern; tramite modelli astratti abbiamo maggiore flessibilità, in quanto sono definite delle regole su una certa classe di documento e regole diverse su altre classi: queste classi saranno quindi facili da ampliare e da adattare a diverse esigenze;

- il linguaggio deve definire una robusta nomenclatura: è opportuno creare un modello pienamente espressivo, persistente e invariante anche col passare del tempo, in modo tale che un riferimento ad una risorsa possa restare intatto anche con lo sviluppo delle tecnologie;

- il linguaggio deve essere in grado di separare i metadati che provengono da fonti ufficiali dai metadati che provengono da altre fonti, ad esempio informazioni di sistema oppure informazioni di pubblicazione.

Negli ultimi anni sono stati sviluppati diversi standard XML per la descrizione e l'archiviazione dei documenti legali. Secondo studi tecnici [PCR09], possiamo dividere questi linguaggi in quattro categorie:

- i linguaggi di "prima generazione" effettuano una descrizione del contenuto e della sua struttura con un approccio molto simile a quello di un database relazionale: in questo modo, il documento viene completamente frammentato, perdendo inevitabilmente l'integrità della sua struttura base;

- i linguaggi di "seconda generazione" dedicano molta attenzione alla precisa struttura del documento, ai diversi significati delle diverse sezioni,

e ai metadati; la creazione di queste regole di markup, però, non è preceduta da un'astrazione delle classi di tipo di documento, e il risultato finale è una grammatica poco flessibile, con un eccessivo numero di regole e con il rischio di sovrapposizione fra metadati e altre informazioni specifiche sul testo;

- i linguaggi di "terza generazione" sono basati sui pattern: questa astrazione, come dicevo nel paragrafo precedente, permette maggiore flessibilità sul numero e la diversità di risorse che il linguaggio deve poter affrontare e descrivere, creando degli appositi strati in cui contenuto, metadati ed ontologie sono perfettamente separati; questa regole generiche, però, rendono il linguaggio privo di vincoli particolarmente restrittivi;

- i linguaggi di "quarta generazione", infine, utilizzano i pattern assieme a grammatiche vincolanti, in modo da risolvere il problema dei linguaggi di "terza generazione".

Le prossime sezioni sono dedicate all'analisi di tre diversi standard XML: *FORMEX*, il linguaggio di markup ufficiale dell'Unione Europea, appartenente alla "prima generazione"; *NormeInRete*, il linguaggio utilizzato nel sistema legislativo italiano, appartenente alla "seconda generazione"; *AKOMA NTOSO*, il linguaggio di "terza generazione" creato nell'ambito di un progetto delle Nazioni Unite.

2.3.1 FORMEX

FORMEX (Formalized Exchange of Electronic Publications) è un formato standard per il markup digitale dell'Official Journal (Gazzetta Ufficiale) dell'Unione Europea[11]. È stato sviluppato dall'Office for Official Publications a partire dal 1985: le prime versioni si basavano su SGML, ma nel 1999 è stata effettuata una migrazione verso XML; nell'ultima versione (v4, adottata

[11] http://formex.publications.europa.eu/index.html

definitivamente dal 2004) la grammatica è definita utilizzando XML Schema, e la codifica dei caratteri utilizzata è Unicode UTF-8[12]. Il file generato da questo formato (.fmx4) è in realtà un archivio ZIP contenente file XML (chiamati *istanze di Formex*) ed elementi addizionali (ad esempio grafici sotto forma di immagini TIFF). Esamino brevemente gli elementi di root previsti dalla grammatica di Formex[13]:

Listing 2.5: Struttura dell'elemento ACT nella grammatica di Formex

```
<xd:element name="ACT">
  <xd:complexType>
    <xd:sequence>
      <xd:element ref="BIB.INSTANCE"/>
      <xd:choice minOccurs="0" maxOccurs="unbounded">
        <xd:element ref="GR.ANNOTATION"/>
        <xd:element ref="TOC"/>
      </xd:choice>
      <xd:element ref="TITLE"/>
      <xd:choice minOccurs="0" maxOccurs="unbounded">
        <xd:element ref="GR.ANNOTATION"/>
        <xd:element ref="PROLOG"/>
        <xd:element ref="TOC"/>
      </xd:choice>
      <xd:element ref="PREAMBLE"/>
      <xd:element ref="ENACTING.TERMS"/>
      <xd:element ref="FINAL" minOccurs="0"/>
    </xd:sequence>
    <xd:attribute name="NNC" type="t_boolean" default="NO"/>
  </xd:complexType>
</xd:element>
```

ACT è utilizzato per contrassegnare diversi tipi di documenti legali (ad esempio direttive, decisioni, regolamenti), che possono come non possono avere una struttura regolare; questo elemento viene utilizzato sia

[12] http://formex.publications.europa.eu/formex-4/physspec/formex-4-introduction.htm

[13] http://formex.publications.europa.eu/formex-4/manual/manual.htm

per i testi adottati che per gli atti preparatori;

AGR è utilizzato per contrassegnare un accordo (agreement);

ANNEX è utilizzato per codificare un allegato;

CJT è utilizzato per il markup di documenti derivanti dalla Corte di Giustizia;

COMPETITION è usato per contrassegnare gli annunci relativi a concorsi;

CORR è usato per rappresentare i corrigenda;

DOC è utilizzato per la singola istanza di Formex che contiene la descrizione bibliografica e l'effettiva struttura di un documento;

FRAGMENT è usato come radice per una porzione (frammento) di un documento valido;

GENERAL è utilizzato per tutti i documenti non identificabili tramite gli altri elementi di root;

LSEU è utilizzato per il markup di sintesi legislative (legislation summary);

PUBLICATION è utilizzato per descrivere alcuni tipi di pubblicazioni.

FORMEX è un linguaggio appartenente alla "prima generazione" [PV12], e in quanto tale, effettua un markup in un modo che ricorda l'approccio dei database relazionali, frammentando il documento (basti pensare al fatto che un documento legale in FORMEX è in realtà diviso in più "istanze di FORMEX").

2.3.2 NormeInRete

NormeInRete (NIR) è il nome di un progetto iniziato nel 1999 da parte dell'AIPA[14], sotto la direzione del Ministero della Giustizia, con un duplice scopo [DGPF12]:

- realizzazione di un formato digitale aperto e di un solido dizionario di marcatura per la rappresentazione e l'identificazione di qualsiasi atto normativo previsto dal sistema legislativo italiano, da far adottare formalmente a tutte le istituzioni che possono generare tali documenti;
- realizzazione di un portale accessibile a tutti i cittadini per la pubblicazione, l'archiviazione e la consultazione di tali documenti, cercando di creare un unico punto di riferimento ufficiale sul web per la pluralità delle istituzioni coinvolte.

Gli obiettivi appena illustrati sono stati effettivamente messi in atto col passare degli anni, coinvolgendo diverse istituzioni pubbliche [DGPF12], dando vita a: lo standard URN[15] NIR, un meccanismo globale di assegnazione di nomi uniformi ai documenti giuridici (che quindi non dipende da fattori esterni al processo legislativo); lo standard aperto XML NIR per il markup digitale dei provvedimenti normativi; il portale online NIR, pubblicato nel 2000 (diventato poi "Normattiva") [MMSV02].

Data la diversità della tipologia degli attori coinvolti nel processo normativo e, di conseguenza, dei documenti generati, la grammatica definita da NIR è molto lunga, in quanto vengono definiti diversi "modelli di documento", ciascuno con delle specifiche regole da rispettare; questi modelli possono essere divisi in tre grosse categorie [LB03]:

- documenti con una struttura ben definita (ad esempio le leggi);
- documenti con una struttura parzialmente definita (ad esempio i decreti);

[14] Autorità per l'Informatica nella Pubblica Amministrazione, un organismo pubblico italiano istituito nel 1993 e soppresso nel 2003.

[15] URN è l'acronimo di Uniform Resource Name

- documenti generici.

La complessità della grammatica aumenta poiché la presenza o assenza di eventuali caratteristiche di elementi nel documento è gestita attraverso due diverse versioni di grammatica, ovvero una *"strict"*, con la presenza di più vincoli, e l'altra *"loose"*, ovvero più permissiva, e con compatibilità reciproca fra entrambe le versioni: la versione "strict" non descrive strutture differenti rispetto alla versione "loose", semplicemente aggiunge più vincoli [MMSV02]. Infine, la grammatica fornisce anche caratteristiche specifiche per elementi testuali, che hanno un valore più tipografico che semantico.

In dettaglio, la struttura base della grammatica di NIR è divisa in sei parti (documenti):

1. definizioni specifiche per la versione Strict (*strict.dtd*);
2. definizioni specifiche per la versione Loose (*loose.dtd*);
3. definizioni globali (*global.dtd*);
4. definizioni specifiche della struttura della norma (es. intestazione, preambolo, articolato) (*norm.dtd*);
5. definizione di strutture testuali per dare un valore tipografico (*text.dtd*);
6. definizioni dei metadati per gli enti del sistema normativo (*meta.dtd*).

La complessa struttura della grammatica NIR e il fatto che possa essere applicata ai soli documenti derivanti dalla normativa italiana evidenzia la sua appartenenza alla classe di linguaggi di "seconda generazione" [PV12].

Listing 2.6: Frammento di un documento legale con markup NIR

```
<?xml version="1.0" encoding="UTF-8"?>                                    1
<NIR xmlns="http://www.normeinrete.it/nir/2.2/"                           2
 xmlns:dsp="http://www.normeinrete.it/nir/disposizioni/2.2/"              3
 xmlns:h="http://www.w3.org/HTML/1998/html4"                              4
 xmlns:xlink="http://www.w3.org/1999/xlink" tipo="monovigente             5
   ">
```

```
<Legge>
  <meta>
    <descrittori>
      <pubblicazione norm="20160913" num="214" tipo="␣␣"/>
      <redazione id="16G00192" nome="" norm=""/>
      <urn valore="urn:"/>
    </descrittori>
  </meta>
  <intestazione>
    <tipoDoc>monovigente</tipoDoc>
    <dataDoc norm="20160826">26 agosto 2016</dataDoc>
    <numDoc>179</numDoc>
    <titoloDoc>Modifiche ed integrazioni al Codice dell'
  amministrazione
    digitale, di cui al decreto legislativo 7 marzo 2005, n
  . 82, ai
    sensi dell'articolo 1 della legge 7 agosto 2015, n.
  124, in materia
    di riorganizzazione delle amministrazioni pubbliche.
  (16G00192)</titoloDoc>
  </intestazione>
  <formulainiziale/>
  <articolato>
    <capo id="0"><num/><articolo id="1">
    <num>Art. 1.</num><comma id="art1-com1"><num>1</num>
    <corpo>
      <h:br/>
      <h:p h:style="text-align:␣center;">IL PRESIDENTE
  DELLA REPUBBLICA </h:p>
      <h:br/>
      <h:p h:style="padding-left:2px;">Visti gli articoli
  76, 87 e 117, secondo
      comma, lettera r), della Costituzione; </h:p>
...
```

2.3.3 AKOMA NTOSO

AKOMA NTOSO[16] (*Architecture for Knowledge-Oriented Management of African Normative Texts using Open Standards and Ontologies*) è un framework, un insieme di specifiche (la cui parte fondamentale è rappresentata da un dialetto XML) sviluppate all'interno di un progetto dell'UNDESA[17] nel 2005, con lo scopo di aiutare i parlamenti africani ad assolvere al meglio le loro funzioni democratiche [Cer13]. AKOMA NTOSO è stato ideato per sopperire alle seguenti necessità [VZ07]:

- sviluppare uno standard comune ed aperto per lo scambio di dati fra diversi parlamenti;
- definire una struttura di un document model di base sulla quale i sistemi parlamentari possono essere costruiti;
- realizzare un semplice meccanismo per le citazioni ed i riferimenti incrociati fra documenti legali creati dagli stessi organi/parlamenti oppure fra organi di stati diversi.

Il dialetto XML progettato per tali scopi è in grado di descrivere un'ampia gamma di documenti legali, che comprendono: l'insieme di tutti i documenti della legislazione primaria (inclusi quelli che rappresentano il ciclo di vita di una legge in fasi diverse del suo sviluppo), i dibattiti parlamentari, tutti gli emendamenti/correzioni legislative, documenti che riassumono briefing di specifici comitati, un'ampia gamma di documenti giudiziari e tutti i tipi di gazzette ufficiali [VZ07]. Questo dialetto è anche uno standard OASIS, con il namespace *LegalDocumentML* [18].

Il dialetto AKOMA NTOSO è in grado di scindere chiaramente il markup semantico da quello strutturale e presentazionale: il markup semantico è sicuramente quello più presente e corposo, in quanto dare un significato semantico

[16] In lingua akan (una lingua parlata nella parte ovest dell'Africa) significa "cuori legati"
[17] United Nations - Department for Economics and Social Affairs
[18] www.oasis-open.org/committees/tc_home.php?wg_abbrev=legaldocml

ad una porzione di testo implica il fatto che tale porzione abbia anche uno specifico aspetto e ruolo strutturale; il markup strutturale viene utilizzato per quelle porzioni di documento che non hanno uno specifico ruolo semantico; infine, il markup presentazionale è riservato per quelle parti di testo che hanno un particolare aspetto tipografico ma non un ruolo semantico/strutturale ben definito [VZ07].

La grammatica di AKOMA NTOSO è divisa in due parti (schemi):

- Un unico *Generic Schema*, poco restrittivo, con la definizione di tutti gli elementi possibili ma con pochissimi vincoli su di essi (ad esempio, permette tutti gli elementi in qualsiasi posizione);

- un insieme di diversi *Detailed Schema*, che aggiungono regole e vincoli agli elementi già definiti nel Generic Schema; rappresentano un sottoinsieme del Generic Schema.

Listing 2.7: Schema dell'elemento ACT nella grammatica AKOMA NTOSO

```
<xsd:element name="act" type="hierarchicalStructure">
  <xsd:annotation>
    <xsd:documentation>
      <type>Element</type>
      <name>act</name>
      <comment>Element act is used for describing the
   structure and content of an act</comment>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:unique name="eId-act">
    <xsd:selector xpath=".//*"/>
    <xsd:field xpath="@eId"/>
  </xsd:unique>
  <xsd:unique name="GUID-act">
    <xsd:selector xpath=".//*"/>
    <xsd:field xpath="@GUID"/>
  </xsd:unique>
</xsd:element>
```

Lo schema generico è utilizzato come controllo iniziale: può dichiarare validi anche dei documenti che presentano una struttura irregolare; questo allo scopo di poter validare anche documenti temporanei non completi che possono essere emessi dalle istituzioni durante l'iter legislativo. È infatti compito dei marcatori quello di assegnare tag ed informazioni a specifiche porzioni di testo: gli errori generati dalla validazione tramite lo schema generico evidenziano un markup realizzato in modo errato; gli errori generati dalla validazione tramite lo schema "specifico", invece, evidenziano il fatto che non siano state seguite le linee guida locali per la stesura del documento [VZ07].

Dietro alla realizzazione dello standard AKOMA NTOSO c'è un forte studio ed utilizzo degli XML Design Pattern[19] legati alla creazione di una nuova grammatica e alla definizione dei content model. I più importanti da citare per la realizzazione dello schema sono: *Universal Root* (un unico elemento root che contiene tutti gli elementi che descrivono i 5 tipi diversi di documento); *Consistent Element Set* (molti elementi condividono lo stesso content model, in modo da mantenere la semplicità generale dello schema); *Generic document and role attribute* (ci sono 5 elementi generici, che prendono il nome del rispettivo pattern, per il markup di parti di testo per cui non è stato fornito un elemento specifico; più in dettaglio nel prossimo paragrafo); *Reuse document types* (molti elementi hanno lo stesso significato degli omonimi elementi in XHTML).

I Design Pattern sono stati utilizzati anche per la progettazione dei content model. Ne sono previsti 5 tipi diversi, e questo comporta che tutti gli elementi complessi di AKN in realtà possono essere descritti ed utilizzati in base al pattern principale a cui fanno riferimento; essi sono:

hierarchy - serie di sezioni, anche annidate, con titolo e numerazione, che possono contenere altre sezioni o *block*, ma non del testo diretto;

block - container strutturato verticalmente che può contenere del testo o altre sottostrutture;

[19] Un Design Pattern è un modello logico da applicare per la prevenzione di un problema noto che può presentarsi nelle fasi di progettazione e sviluppo del software.

inline - elemento che evidenzia una porzione di testo per un particolare aspetto semantico o presentazionale;

marker - elemento vuoto per inserire metadati o riferimenti di nota;

container - sequenza di elementi specifici (anche opzionali).

La forte presenza dei pattern, la possibilità di poter descrivere uno svariato numero di documenti legali (anche molto differenti fra loro) e l'assenza di vincoli particolarmente restrittivi, identificano AKOMA NTOSO come un linguaggio di "terza generazione"[PV12].

2.4 Database di documenti legali

In questo capitolo, dopo aver dato una chiara definizione di "documento strutturato", ho esaminato in modo più approfondito il linguaggio XML, ottimo per il markup di documenti strutturati di qualsiasi tipo, citando tre esempi di utilizzo in contesti completamente differenti.

La sua caratteristica di essere "royalty-free", inoltre, lo rende particolarmente adatto ad ambiti molto delicati, in cui è impensabile dover dipendere da una licenza esterna restrittiva, quale, ad esempio, l'ambito legale. Numerosi sono stati i dialetti XML ideati per tale scopo: dopo averne evidenziate le principali caratteristiche e l'importante suddivisione nelle "4 generazioni", ho illustrato tre esempi di dialetti attualmente in uso.

Oltre all'ideazione di standard XML per il markup di tali documenti, sorge anche l'esigenza di creare dei portali online adatti all'archiviazione e alla fruizione di questi documenti, sia da parte di tecnici ed esperti del settore, ma soprattutto da parte dei comuni cittadini, che nei paesi democratici richiedono la massima trasparenza normativa.

Nel prossimo capitolo parlo di questi portali, in particolare di *EUR-Lex*, il portale ufficiale dell'Unione Europea, incentrando l'attenzione su un particolare tipo di documento legale, le direttive europee, l'elemento centrale del mio progetto di ricerca, evidenziandone la struttura e i metadati più importanti.

Capitolo 3

La distribuzione di documenti legali

Il secondo capitolo di questa dissertazione si occupa della distribuzione dei documenti legali nel web. Nella prima sezione introduco il concetto di *Portale di documenti legali*, specificando le funzionalità minime che dovrebbe avere; successivamente, analizzo in dettaglio due portali presi come esempi: *Normattiva - Il portale della legge vigente*, il sito web ufficiale dello Stato Italiano che mette a disposizione tutta la normativa statale in modalità multivigenza; *Legislation.gov.uk*, il portale ufficiale del Regno Unito che contiene quasi 1000 anni di legislazione primaria ed una parte della legislazione secondaria. Nella seconda sezione entro nel contesto della normativa europea: inizio col definire le *direttive europee*, un particolare tipo di documento legale di cui mi sono occupato nel mio progetto di tesi, per poi descrivere *CELLAR*, il sistema di archiviazione di tutti i documenti legali dell'UE. Nelle sezioni 3 e 4 parlo in maniera approfondita di *EUR-Lex*, il portale ufficiale dell'UE attraverso il quale è possibile accedere al database di documenti contenuti in CELLAR. Infine, nell'ultima sezione, analizzo brevemente i problemi riscontrati nella navigazione e nell'utilizzo di EUR-Lex.

3.1 Portali per la distribuzione di documenti legali

Nelle moderne democrazie, i prodotti del processo legislativo, ossia leggi e più in generale tutti i documenti utili a descrivere i passaggi dell'iter legislativo, devono essere pubblicati e liberamente accessibili ad ogni richiesta. Sia i cittadini che i professionisti devono avere un rapido accesso a tutte le più importanti fonti del diritto in modo da poter svolgere al meglio il loro lavoro. Nasce quindi l'esigenza di immagazzinare e catalogare tutta questa mole di informazioni in un posto di facile accessibilità: un portale online.

Un portale online di documenti legali è una struttura un po' più complessa di un semplice database. Un DBMS[1] è un sistema in grado di gestire informazioni strutturate. Nello specifico, i database relazionali, ovvero il tipo di database più utilizzato attualmente[2], contengono le informazioni in strutture tabellari, dove ogni riga della tabella rappresenta un *record*, ed ogni campo del record (cella della tabella) è di tipo predefinito (ad esempio un numero o una stringa) e di contenuto non arbitrario (ad esempio un numero definito in un range oppure un limite di caratteri). Un DBMS è in grado di rispondere ad una richiesta che contiene dei vincoli su un sottoinsieme di campi, listando tutti i record contenuti che soddisfano questi vincoli.

Un portale di documenti legali dovrebbe affidarsi non solo ad un puro DBMS, ma anche a soluzioni più moderne basate, ad esempio, su database *NoSQL* orientati al documento[3]; in questo tipo di basi di dati i "documenti" consistono in record che non hanno uno schema restrittivo, ciascuno di essi può avere dei campi diversi e generalmente non esistono campi vuoti; le codifiche più comuni per questi record sono XML, YAML, JSON e BSON

[1] acronimo di DataBase Management System

[2] http://pypl.github.io/DB.html

[3] NoSQL può indicare "non SQL" o "non relational" oppure "not only SQL", è un termine utilizzato per indicare un qualsiasi tipo di database che non memorizza le informazioni nel classico modello relazionale; i database orientati al documento sono un particolare tipo di database NoSQL (https://en.wikipedia.org/wiki/NoSQL)

(nel mio progetto di ricerca mi sono avvalso sia di *eXist-DB* che di *MongoDB*, due basi di dati orientate al documento; la prima utilizza lo standard XML mentre la seconda utilizza lo standard JSON).

Lancaster, nel 1978, coniava il termine di *information retrieval system*:

> "Il recupero delle informazioni è un processo di ricerca in alcune collezioni di documenti [...] in modo da identificare quelli che si occupano di un particolare soggetto. Qualsiasi sistema che sia in grado di facilitare questa ricerca può essere legittimamente chiamato *information retrieval system*"

La ricerca di informazioni in un database legale può richiedere molto tempo e non è sicuramente un'operazione alla portata di tutti: non solo può essere difficile selezionare le giuste parole per esprimere il proprio bisogno, è anche difficile combinare fra loro queste parole in una *query*; anche nel caso in cui un utente possa pensare che siano i termini giusti, non è detto che siano gli stessi termini usati dal legislatore [Mat99].

Ho citato Lancaster proprio perchè la sua definizione, seppur molto datata, identifica la funzionalità chiave di un portale per la distribuzione di documenti legali: un sistema che, al di là della semplice ricerca testuale, sia dotato di una buona classificazione di tali documenti in base ad una serie di metadati appropriati, per identificare l'interesse ad ogni specifico soggetto.

3.1.1 Normattiva

Normattiva è un portale di distribuzione per i documenti normativi italiani. È stato lanciato il 19 marzo 2010, realizzato e curato dall'Istituto Poligrafico e Zecca dello Stato, e al momento della presentazione conteneva tutti gli atti normativi pubblicati dal 1 gennaio 1981. I documenti contenuti in Normattiva sono archiviati seguendo le regole dello standard aperto XML *NormeInRete* (NIR, vedi sezione 2.3.2); è possibile accedere ai documenti seguendo il meccanismo globale di assegnazione di nomi uniformi ai documenti giuridici definito dallo standard URN *NormeInRete* [Ven10]. At-

tualmente, Normattiva contiene tutti gli atti normativi pubblicati a partire dal 1933; fra gli obiettivi futuri è prevista l'integrazione nella banca dati di tutti i rimanenti atti normativi pubblicati dal 1861 (e di quelli non normativi che hanno apportato modifiche ad atti normativi)[4].

Figure 3.1: Home page di normattiva.it

Normattiva è il risultato finale di un progetto istituito con la legge numero 388/2000 e regolamentato dal DPCM 24 gennaio 2003: il progetto aveva l'obiettivo di fornire ai cittadini l'accesso gratuito attraverso internet alla normativa aggiornata ed al legislatore strumenti di supporto per la produzione e semplificazione normativa. La prima versione del portale prendeva lo stesso nome degli standard sviluppati, quindi *NormeInRete* (nir.it): è stato mantenuto online dal 2000 al 2008 grazie al *CNIPA*[5], arrivando a federare oltre 50 amministrazioni e indicizzando circa 300.000 documenti; la sua chiusura è stata causata dall'attuazione del decreto-legge 22 dicembre 2008 n. 200, che stabiliva "la convergenza presso il Dipartimento degli affari giuridici e legislativi della Presidenza del Consiglio dei Ministri di tutti i progetti di

[4]http://www.normattiva.it/static/progetto.html

[5]Centro Nazionale per l'Informatica nella Pubblica Amministrazione, un ente di diritto pubblico italiano soppresso nel 2003, sostituito dal *DigitPA*

informatizzazione e di classificazione della normativa statale e regionale in corso di realizzazione da parte delle amministrazioni pubbliche" [DGPF12].

Uno dei punti di forza del portale Normattiva è la modalità *multivigenza*: è possibile ricercare e visualizzare un atto normativo non solo nella sua versione vigente (ovvero la versione legalmente valida alla data della ricerca), ma anche nella sua vigenza ad una data prescelta, o nella pura versione originale, o infine nella sua versione originale assieme agli aggiornamenti. Dal punto di vista dottrinale, la possibilità di poter visualizzare il completo percorso storico, in linea temporale, di un atto normativo, è un'ottima chiave per comprendere al meglio la legge stessa; si può comprendere in modo approfondito la volontà del legislatore, e capire se essa sia stata influenzata, ad esempio, da particolari eventi storici accaduti nel tempo. [Ant11] [PB02]

Figure 3.2: Interfaccia di ricerca avanzata di Normattiva

Normattiva mette a disposizione un'interfaccia di ricerca semplice ed una avanzata: la prima permette di visualizzare un atto nella sua attuale versione vigente, potendo specificare gli estremi dell'atto (numero atto, giorno mese ed anno di pubblicazione, numero articolo) e le parole chiave da ricercare, indistintamente, sia nel titolo che nel testo dell'atto; la ricerca avanzata, invece,

permette innanzitutto di specificare la preferenza di visualizzazione dell'atto fra "atto originario con aggiornamenti", "atto vigente ad una data" e "atto originario" (senza aggiornamenti), e successivamente offre altri strumenti per affinare la ricerca, ovvero la "denominazione atto" (un menù a tendina che contiene tutte le tipologie di atti, di cui se ne può selezionare una soltanto), una semplificazione dell'utilizzo degli operatori booleani per le parole da cercare sia nel titolo che nel testo (titolo e testo hanno dei campi specifici per inserire le parole chiave), e infine la possibilità di poter cercare non in una singola data ma in un range di date.

La schermata di visualizzazione di un atto è divisa in tre parti: la parte superiore, la parte di navigazione a sinistra e la parte principale con l'effettivo contenuto testuale dell'atto. Nella parte superiore, oltre alle informazioni essenziali dell'atto, sono presenti una serie di pulsanti che permettono alcune funzionalità:

versione stampabile per poter stampare la totalità o anche solo una o più parti dell'atto;

esporta per esportare l'intero atto o parti di esso nei formati HTML ed XML (seguendo lo standard NIR);

aggiornamenti all'atto per visualizzare un elenco completo di tutti gli atti (con i relativi link) che hanno modificato l'atto in questione, ordinati per data di pubblicazione in ordine crescente;

circolari per visualizzare le circolari emesse relative all'atto;

note all'atto per visualizzare tutti gli atti che hanno aggiunto note all'atto in questione;

lavori preparatori per visionare i lavori preparatori ed avere i link esterni alle successioni delle letture parlamentari;

relazione

aggiornamenti al titolo/struttura per individuare, se presenti, cambi di titolo o struttura dell'atto avvenuti nel corso del tempo;

rubriche articoli per l'elenco dei titoli dei singoli articoli dell'atto.

La parte di navigazione a sinistra presenta l'intero scheletro dell'atto, suddiviso in parti/titoli/capi/sezioni, con i link ai singoli articoli dell'atto e ad eventuali allegati anch'essi suddivisi (se è presente una suddivisione degli allegati). Infine, nella parte centrale è presente l'effettivo contenuto testuale dell'atto, con la possibilità di evidenziare i link di tutti gli atti citati all'interno del testo dell'atto in questione.

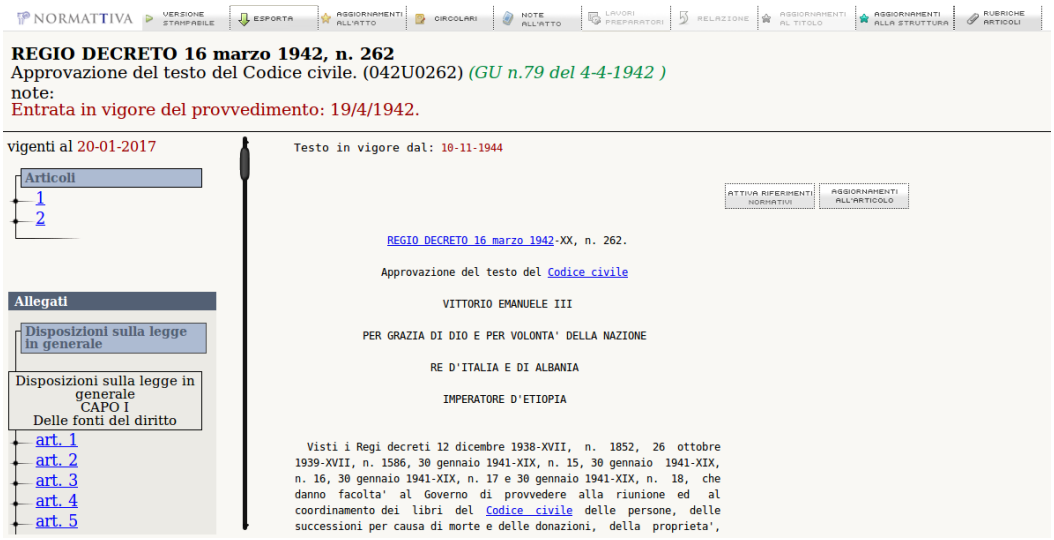

Figure 3.3: Schermata di visualizzazione atto in Normattiva

3.1.2 Legislation.gov.uk

Legislation.gov.uk è il sito web ufficiale per l'accesso agli atti normativi del Regno Unito, ed è gestito da *The National Archives* [6]. Il suo database include tutta la legislazione primaria in vigore al 1 febbraio 1991 (i documenti

[6]Gli "Archivi Nazionali" sono un dipartimento del Governo del Regno Unito e un'agenzia esecutiva del Segretario di Stato per la Giustizia

più longevi risalgono al 1267) più tutta la legislazione primaria e secondaria pubblicata dal 1991 in poi. [7]

Figure 3.4: Home page di Legislation.gov.uk

Le prime iniziative del Governo per la creazione di un database degli atti normativi sono state lanciate nel 1991, a seguito di un appalto concesso ad una azienda privata. Il sistema è stato terminato nel 1993 ma formalmente accettato dal Governo nel 1995; era stato concepito come sistema disponibile per uso interno agli organi statali, agli esperti del settore e alle librerie nazionali; il sistema era organizzato in un CD-ROM e comprendeva un editor WYSIWYG [8] in grado di visualizzare i documenti scritti in SGML. Nel 1995 è stata presentata una prima versione web disponibile solo per usi interni del Governo. Intanto, per quasi 10 anni non è cambiato nulla, fino all'annuncio ufficiale nel 2004 dello sviluppo di due diversi portali di accesso alla normativa, uno destinato al Governo e l'altro al pubblico. I sistemi sono stati lanciati nel maggio 2006, quello per il pubblico non era gratuito, ma a seguito delle numerose pressioni da parte dell'opinione pubblica (come ad esempio

[7] www.legislation.gov.uk/aboutus

[8] acronimo di "What You See Is What You Get"

la campagna "Free Our Data" del quotidiano *The Guardian*[9]) il Governo ha deciso di renderlo pubblico gratuitamente. I due sistemi si sono via via fusi a partire dal 2008, dando vita all'attuale portale [Zan15].

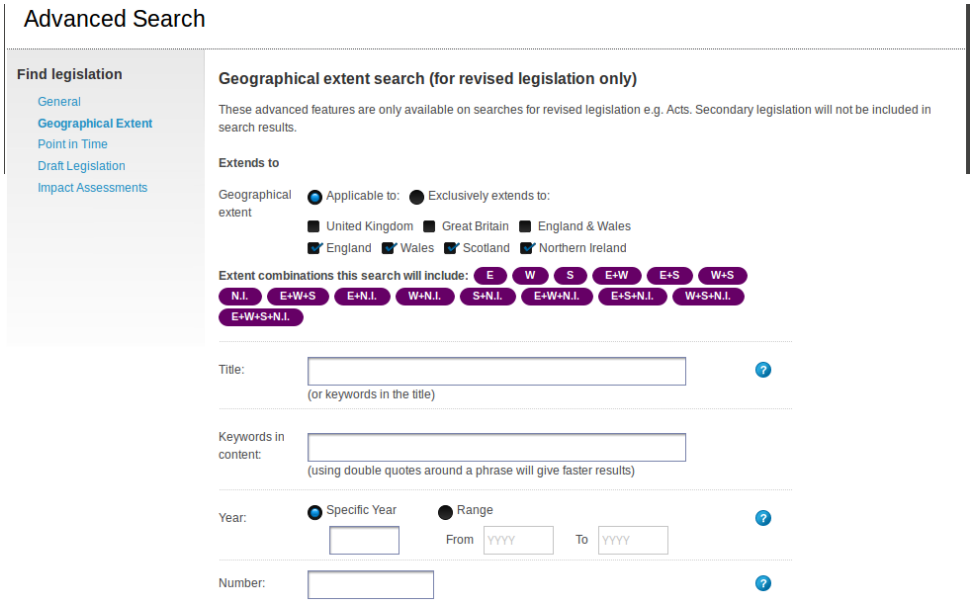

Figure 3.5: Tab "geographical extent" della ricerca avanzata

Legislation.gov.uk offre i metodi "semplice" ed "avanzato" di ricerca. La ricerca semplice prevede un form con i 4 campi titolo, anno, numero e tipo di documento, e tale form è presente in quasi tutte le pagine del sito, nella parte alta della pagina. Alla ricerca avanzata è invece dedicata una propria pagina, che include 5 tab:

general - i campi della ricerca semplice con l'aggiunta della ricerca per parole chiave, per un range di anni, e per visualizzare il risultato in inglese o in gallese;

geographical extent - un'estensione della tab *general*, disponibile solo per la legislazione primaria, in cui è possibile anche distinguere le varie regioni del Regno Unito in cui un certo atto è applicabile o esteso;

[9] www.freeourdata.org.uk/blog/

point in time - una ricerca semplice con l'aggiunta di una data da selezionare per la vigenza dell'atto;

draft legislation - per ricercare nei lavori preparatori degli atti;

impact assessments - una pagina esclusiva per ricercare fra le procedure finalizzate ad individuare, descrivere e valutare gli impatti degli atti normativi sulla società (ad esempio, l'impatto ambientale di un'opera oppure l'impatto sociale di una manovra economica).

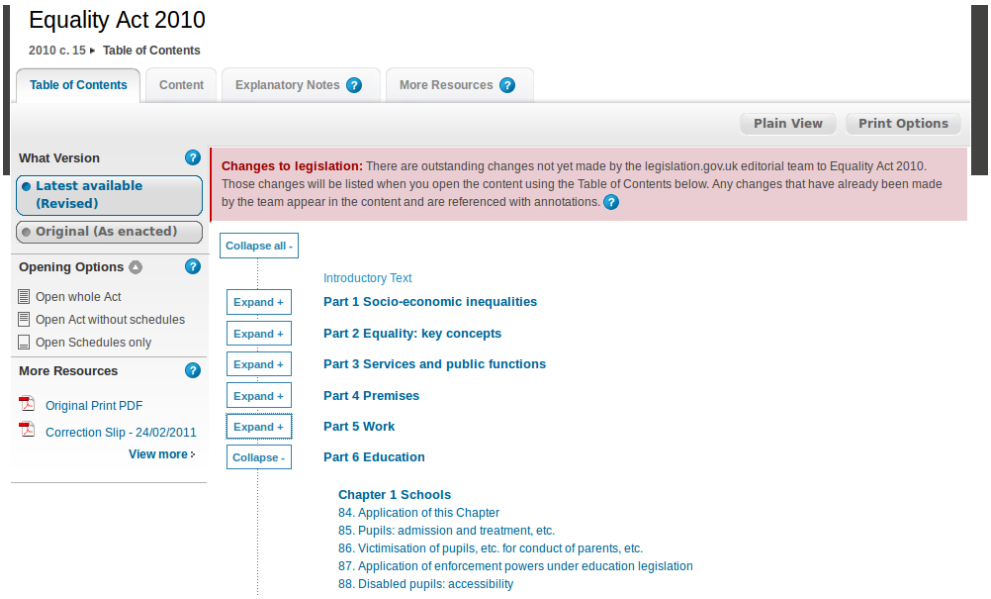

Figure 3.6: Schermata di visualizzazione atto in Legislation.gov.uk

La schermata di visualizzazione di un atto indirizza subito l'utente alla tabella dei contenuti del documento, divisa in parti, capitoli e sezioni. Nella parte superiore sono presenti alcune tab per accedere al contenuto dell'atto (la visualizzazione è divisa sezione per sezione, facendo navigare l'utente con i pulsanti "previous" e "next"), alle note esplicative dell'atto (non disponibile per tutti gli atti) e ad altre risorse collegate (ad esempio valutazioni degli impatti o emendamenti). Nella parte sinistra è presente un menù che permette di accedere alla versione attuale o originale dell'atto e agli allegati. E' possibile esportare l'atto nel formato PDF (l'atto intero, una porzione

dell'atto o solo la tabella dei contenuti), oppure visualizzarlo in una pagina HTML separata.

3.2 Distribuzione delle direttive europee

L'articolo 255 del Trattato di Amsterdam[10], paragrafo 1, dice:

> "1. Qualsiasi cittadino dell'Unione e qualsiasi persona fisica o giuridica che risiede o abbia la sede sociale in uno stato membro ha il diritto di accedere ai documenti del Parlamento Europeo, del Consiglio e della Commissione secondo i principi e alle condizioni da definire a norma dei paragrafi 2 e 3."

Il portale EUR-Lex (sezione 2.3) nasce anche con l'intenzione di garantire questo diritto al cittadino. Esso contiene tutta la documentazione ufficiale dell'UE; fra la documentazione ufficiale sono incluse le *direttive europee*, un particolare tipo di atto normativo, di cui mi sono pienamente occupato nel mio progetto di ricerca. Nella prossima sezione ne elenco le caratteristiche.

3.2.1 Le direttive europee (definizione)

Le direttive europee sono una delle fonti secondarie del diritto comunitario (le altre fonti secondarie sono i *regolamenti* e le *decisioni*). Sono atti dotati di efficacia vincolante, emanati congiuntamente dal Parlamento europeo e dal Consiglio dell'UE, al fine dell'assolvimento degli scopi previsti dai *Trattati* (fonte primaria del diritto comunitario). [11]

Le direttive possono avere portata individuale, nel senso che possono essere rivolte anche ad un singolo stato o a un sottoinsieme degli stati membri (quelle rivolte a tutti gli stati membri sono chiamate *direttive generali*).

[10] Il Trattato di Amsterdam è uno dei trattati fondamentali dell'UE, il primo tentativo di riformare le istituzioni europee in vista dell'allargamento (firmato nel 1997 ed entrato in vigore nel 1999)

[11] https://it.wikipedia.org/wiki/Diritto_dell'Unione_europea

La direttiva è un particolare tipo di atto vincolante, che obbliga lo stato a cui è rivolta a dover raggiungere uno specifico risultato, lasciando allo stato libertà circa la particolare normativa da emanare al fine di raggiungere l'obiettivo. Lo stato, in fase di recepimento, deve comunicare la forma e i mezzi attraverso i quali la direttiva è stata recepita, permettendo alla Corte di Giustizia dell'UE di valutare se i mezzi adottati corrispondono al principio di certezza del diritto.[12]

3.2.2 Organizzazione concettuale dei documenti legali nell'archivio CELLAR

Tutti i contenuti digitali dei documenti dell'UE, assieme ai relativi metadati, sono gestiti da un sistema chiamato *CELLAR*, mantenuto dall'Ufficio delle pubblicazioni dell'UE[13].

Il cuore del sistema CELLAR è il modello *FRBR*[14], uno schema concettuale realizzato tramite il modello entità-relazione allo scopo di fornire una rappresentazione formale alle informazioni bibliografiche.

Il modello FRBR è caratterizzato da tre gruppi di entità: il primo fornisce informazioni utili a catalogare una produzione intellettuale; il secondo si riferisce alle entità coinvolte alla creazione e distribuzione di tali opere; il terzo comprende informazioni relative ai soggetti di tali opere (il secondo e il terzo gruppo sono fuori dagli scopi di questa dissertazione). Il primo gruppo è chiamato con l'acronimo *WEMI* ed è composto dalle seguenti entità[15]:

Work - rappresenta l'effettiva produzione intellettuale;

Expression - identifica la "forma artistica/intellettuale" di tale opera;

Manifestation - è la materializzazione di una specifica *Expression*;

Item - rappresenta il singolo esemplare di una *Manifestation*.

[12] https://it.wikipedia.org/wiki/Direttiva_dell'Unione_europea

[13] https://joinup.ec.europa.eu/software/cellar/description

[14] FRBR è l'acronimo di *Functional Requirements for Bibliographic Records*

[15] en.wikipedia.org/wiki/Functional_Requirements_for_Bibliographic_Records

Per comprendere al meglio il gruppo di entità WEMI basta guardare il suo adattamento al contesto di CELLAR: il Work è il documento legale, a tutti gli effetti un'opera intellettuale; l'Expression è la realizzazione di un certo documento in una specifica lingua; la Manifestation è l'istanza di un documento nella lingua definita dall'Expression; infine l'Item (nel contesto CELLAR è chiamato *Content stream*) è l'entità che contiene fisicamente le informazioni della Manifestation (il file in uno specifico formato).[16]

Il gruppo WEMI è pienamente integrato nel sistema CELLAR; la comprensione di questo modello è stata fondamentale per individuare i campi in CELLAR che racchiudono le informazioni essenziali di una direttiva europea (indico fra parentesi l'entità a cui il campo si riferisce; il carattere "." invece indica un ulteriore livello di annidamento del campo, e in alcuni casi ho omesso livelli di annidamento successivi per semplicità):

WORK_HAS_RESOURCE-TYPE (work) per il tipo di documento;

WORK_DATE_DOCUMENT.YEAR (work) per l'anno di pubblicazione;

RESOURCE_LEGAL_NUMBER_NATURAL_CELEX (work) per l'id CELEX;

DATE_DOCUMENT (work) per la data (in formato esteso) di pubblicazione;

EXPRESSION_USES_LANGUAGE.IDENTIFIER (expression) per il codice ISO della lingua di una specifica Expression;

EXPRESSION_TITLE.VALUE (expression) per il titolo del documento tradotto nella specifica lingua;

EXPRESSION_MANIFESTED_BY_MANIFESTATION (expression) il vettore che contiene tutte le Manifestation di una specifica lingua;

[16] http://publications.europa.eu/documents/2050822/0/CEM-EEU-External+End+User+manual-v15+00.pdf

RESOURCE_LEGAL_CONSOLIDATED_BY_ACT_CONSOLIDATED (work) contiene le informazioni relative all'atto consolidato (in un sottoinsieme di campi);

MANIFESTATION.SAMEAS.URI.IDENTIFIER (manifestation) per il link esatto al file in uno specifico formato;

MANIFESTATION_HAS_ITEM (content stream) per recuperare il contenuto di un file .fmx4.

3.2.3 Altre funzionalità di CELLAR utili alla ricerca

Il numero di metadati contenuti in CELLAR, per quanto concerne le direttive europee, è davvero molto elevato; non sono stati oggetto del mio progetto di tesi, ma è comunque possibile recuperare anche le seguenti informazioni:

- le lingue facenti fede, per le vecchie direttive o per quelle rivolte non a tutti i paesi (sono comunque tradotte in tutte le lingue);
- tutte le date utili (di entrata in vigore, di recepimento, di fine validità, ecc.);
- tutte le classificazioni del documento (in base ai descrittori EuroVoc e agli argomenti);
- tutti i documenti collegati all'atto (oltre agli atti consolidati ci sono numerosi atti collegati, ad esempio il trattato che costituisce la base giuridica, le procedure legislative basate su tale documento, i documenti che lo citano e che vengono citati, gli emendamenti, ecc.);
- le misure nazionali di recepimento.

3.3 EUR-Lex

EUR-Lex è il nome del sito web dell'Unione Europea (UE) che permette l'accesso gratuito ad un vasto archivio di documenti legali dell'UE; è gestito dall'Ufficio delle pubblicazioni UE, è tradotto in tutte le lingue degli stati membri (sia l'interfaccia che gli effettivi documenti) ed offre metodi di ricerca sia semplici che avanzati (dedico la prossima sezione ai metodi di ricerca disponibili su EUR-Lex). La documentazione disponibile (il documento più longevo risale al 1951) comprende: la Gazzetta Ufficiale autentica, tutto il diritto dell'UE (trattati, direttive, regolamenti, ecc.), gli atti preparatori, la giurisprudenza dell'UE (sentenze, ordinanze, ecc.), gli accordi internazionali, i documenti dell'EFTA[17], altri documenti pubblici, la sintesi della legislazione europea e le procedure che portano all'adozione degli atti giuridici.[18]

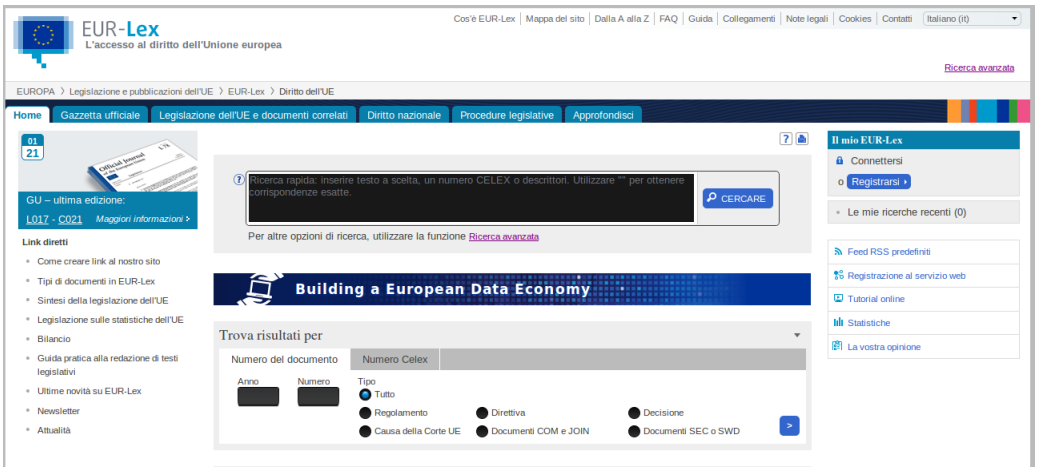

Figure 3.7: Home page italiana di EUR-Lex

EUR-Lex è nato nel 1998, e la sua attuale versione differisce non poco dalla sua versione originale: al di là delle tecnologie utilizzate per processare automaticamente i documenti e generare/memorizzare i metadati, lo scopo finale era diverso; EUR-Lex era nato per offrire gratuitamente la consultazione

[17] EFTA è l'acronimo di *European Free Trade Association* (Associazione Europea di libero scambio)

[18] fonte: http://eur-lex.europa.eu/content/welcome/about.html

della Gazzetta Ufficiale (fino a 20 giorni dalla pubblicazione) e poca altra documentazione ad un pubblico generico, utilizzando un'interfaccia di ricerca molto elementare, ed era stato pensato come servizio gratuito da affiancare al già affermato sistema a pagamento CELEX. CELEX (Communitatis Europae Lex) era un tool a pagamento sviluppato dall'UE che comprendeva un complesso database ricco di documenti risalenti anche al 1950, destinato ad un'utenza professionale (avvocati, giuristi, ecc.); nato nel lontano 1972, solo in lingua francese e per un utilizzo esclusivo all'interno degli organi dell'UE, è stato aperto al pubblico nel 1981, ed è diventato accessibile via interfaccia web nel 1997. È proprio nella fine degli anni 90 che l'UE ha iniziato a pensare all'idea di avere un unico *entry point* per tutta la legislazione europea; l'Ufficio delle pubblicazioni ha operato verso un'integrazione dei servizi di CELEX dentro il portale EUR-Lex; CELEX è diventato un servizio gratuito da luglio 2004 (sotto richiesta del Parlamento Europeo) ed è rimasto attivo fino a fine 2006, mentre da novembre 2004 la nuova versione di EUR-Lex era disponibile in 21 lingue ed aveva assorbito tutte le funzionalità presenti in CELEX [otEU06].

La schermata di visualizzazione di un documento in EUR-Lex è divisa in due tab: la tab "Testo" e la tab "Informazioni sul documento". Nella prima è presente il titolo del documento, il numero della Gazzetta Ufficiale che lo contiene, una tabella con i link per scaricare il documento nelle varie lingue degli stati membri (nei formati HTML, PDF, oppure visualizzazione tramite il sito nella Gazzetta Ufficiale che lo contiene) e l'effettivo testo del documento; è anche possibile una visualizzazione multilingua, fino a 3 lingue contemporaneamente: le versioni sono disposte una accanto all'altra in modo ordinato (articoli corrispondenti sono disposti alla stessa altezza di visualizzazione).

La tab "Informazioni sul documento" estende il contenuto della prima tab elencando:

- tutte le date riguardanti il documento (pubblicazione, entrata in vigore, scadenza, recepimento, fine validità);

Figure 3.8: Visualizzazione multilingua di un documento in EUR-Lex

- diversi tipi di classificazione assegnati al documento (secondo i descrittori *EuroVoc*[19], gli argomenti, ecc.);
- altre informazioni, quali l'organo autore, il tipo di documento, ecc.;
- il numero e il link alla procedura
- link a documenti correlati, sulla base di diversi tipi di correlazioni (documenti che hanno per base giuridica l'atto in questione, procedure legislative basate sull'atto, atti citati e che lo citano, ecc.).

EUR-Lex utilizza lo standard FORMEX (sezione 2.3.1) per l'archiviazione in formato XML dei propri documenti; tuttavia, tramite le due tab di visualizzazione non è possibile scaricare il documento in questo formato tecnico, ma solo in formati presentazionali.

3.4 Ricerca dei documenti su EUR-Lex

Il sito EUR-Lex propone diversi tipi di ricerca:

Ricerca rapida - un campo a testo libero in cui poter inserire parole chiave o numeri, presente nell'header del sito (quindi sempre disponibile).

[19]EuroVoc è un thesaurus multilingue, curato dall'Ufficio delle pubblicazioni, che comprende la terminologia dei settori di attività dell'UE - http://eurovoc.europa.eu

"Trova risultati per" - una sezione presente nella homepage con due campi "Anno" e "Numero" ed una scelta rapida per tipo di documento.

Ricerca avanzata - una pagina dedicata in cui poter fare una ricerca più dettagliata basata su un maggior numero di metadati.

Ricerca per utenti esperti - una pagina disponibile solo agli utenti registrati in cui poter fare ricerche complesse, utilizzando un vasto numero di operatori e caratteri speciali. [20]

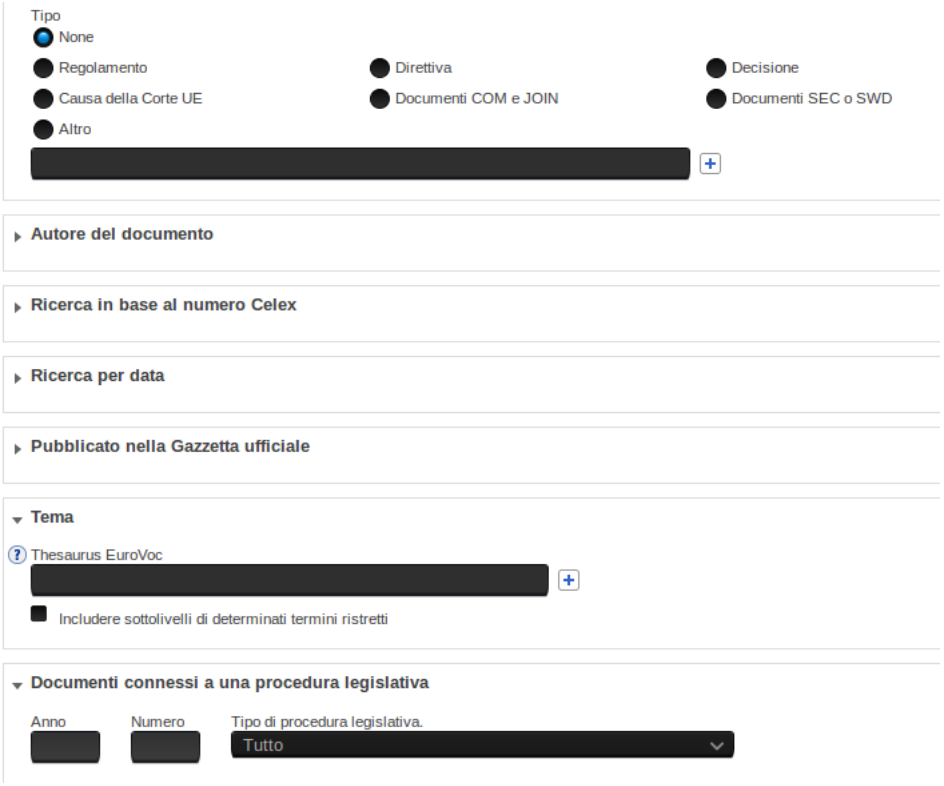

Figure 3.9: Ricerca avanzata (frammento) in EUR-Lex

La ricerca rapida effettua una ricerca testuale in un numero non troppo limitato di campi, ovvero: titolo, testo, numero CELEX (nel prossimo paragrafo), autore, argomento, descrittori EuroVoc (solo una parte), titolo della

[20] http://eur-lex.europa.eu/content/help/search/intro.html

proposta per le procedure legislative, repertori, e infine codice di procedura interistituzionale per le procedure legislative. La ricerca avanzata permette una maggiore precisione nella ricerca, mettendo a disposizione dei singoli campi di input per ognuno dei campi della ricerca semplice, aggiungendo l'utilizzo degli operatori booleani, la possibilità di specificare un range di date, il tipo di documento, il numero della Gazzetta Ufficiale, il tema in base al thesaurus EuroVoc. Registrando un account sul sito di EUR-Lex è possibile accedere alla modalità di ricerca per utenti esperti, ad altre piccole personalizzazioni dell'interfaccia del sito ed ai feed RSS per essere sempre aggiornati sulla nuova documentazione disponibile: la registrazione dell'account è gratuita.

Un elemento caratteristico di EUR-Lex per quanto riguarda la ricerca è il *numero CELEX*: è una stringa che identifica in maniera univoca ciascun documento indicizzato, indipendentemente dalla versione linguistica, ed è comprensibile sia da un umano che da un elaboratore. E' composta da quattro parti:

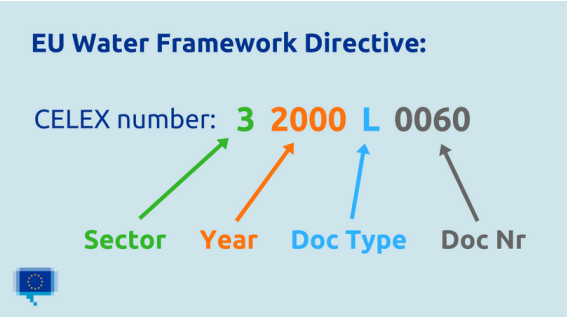

Figure 3.10: Composizione del numero CELEX (fonte: wikipedia)

settore - 1 carattere per indicare l'area di appartenenza del documento (ad esempio trattati, accordi internazionali, diritto derivato ecc., per un totale di 12 settori);

anno - 4 numeri per indicare l'anno (in generale di emanazione) dell'atto;

descrittore - 1 o 2 caratteri per indicare il tipo di documento (ad esempio,

nel settore 3 del diritto derivato troviamo i regolamenti, le direttive, le decisioni, ecc.);

numero - in genere 4 cifre per indicare il numero del documento, seguite eventualmente da suffissi per specificare, ad esempio, una rettifica

CELEX	Scomposizione CELEX
22016D0121	2=Settore "Accordi internazionali"; 2016=ANNO; D=Descrittore "Atti degli enti istituiti da accordi internazionali"; 0121=N. DOCUMENTO
32016R2282	3=Settore "Legislazione"; 2016=ANNO; R=Descrittore "Regolamenti"; 2282=N. DOCUMENTO
32016H0022	3=Settore "Legislazione"; 2016=ANNO; H=Descrittore "Raccomandazioni"; 0022=N. DOCUMENTO

Table 3.1: 3 esempi di numeri CELEX scomposti.

3.5 Semplificare la ricerca (il recupero) delle direttive europee

In questo capitolo mi sono occupato dei portali web per la distribuzione dei documenti legali. Ho analizzato *Normattiva*, il portale ufficiale della legislazione italiana, e *Legislation.gov.uk*, il portale che contiene la normativa del Regno Unito.

Successivamente ho illustrato le diverse funzionalità di ricerca e di visualizzazione dei documenti in *EUR-Lex*, il portale ufficiale di tutti i documenti dell'UE, ed ho brevemente parlato del sistema di archiviazione *CELLAR*.

EUR-Lex svolge bene il lavoro di distribuzione e organizzazione di documenti, tuttavia ci sono degli aspetti che potrebbero essere migliorati, e delle funzionalità importanti che attualmente non sono presenti:

- tramite interfaccia web non è possibile recuperare il documento nel formato tecnico XML (FORMEX);
- non è presente un API REST per poter recuperare direttamente un singolo file in un formato specifico;
- la consultazione del portale tramite dispositivo mobile non è molto comoda in quanto l'interfaccia di EUR-Lex non è *responsive*.

Nel prossimo capitolo analizzo queste problematiche, parlando delle tecnologie che ho utilizzato per implementare un prototipo di portale con lo scopo di migliorare EUR-Lex.

Capitolo 4

Tecnologie per la distribuzione dei documenti legali

Nel terzo capitolo di questa dissertazione illustro, non troppo in dettaglio, le tecnologie che ho utilizzato per la creazione del mio prototipo di portale di distribuzione di documenti legali, evidenziandone per ciascuna le caratteristiche, i benefici e le motivazioni che mi hanno spinto a scegliere tali tecnologie. Le prime due sezioni di questo capitolo si occupano puramente del back-end del portale: nella prima parlo di *Node.js*, il web-server moderno e veloce che ho utilizzato; nella seconda parlo invece delle API REST e dell'importanza che ha un'architettura di un web server in grado di poter supportare questo tipo di modello client-server. La terza sezione è dedicata all'interfaccia: parlo dell'importanza del *Responsive Web Design*, di quanto sia doveroso pensare ad una corretta visualizzazione di un sito web sui dispositivi mobili data la larga fetta di mercato che occupano, illustrando il framework che mi ha semplificato il raggiungimento di tale obiettivo: *Semantic-Ui*. Nella quarta sezione espongo l'importanza delle richieste *AJAX* ed evidenzio la reattività che aggiungono alla user-experience. Infine, nelle ultime due sezioni riassumo e collego tutte queste tecnologie.

4.1 Un web-server moderno e veloce: Node.js

Il panorama delle tecnologie a disposizione per la creazione di un web server è molto ampio. Numerosi, infatti, sono i linguaggi disponibili: per citarne alcuni, fra i più famosi abbiamo PHP, C, Python, Java, Javascript, Ruby, Perl e ASP.NET. Attualmente, PHP è linguaggio più utilizzato fra i web server di cui l'implementazione è nota [1]. Anche il numero dei web server a disposizione è elevato: possiamo citarne alcuni come Apache, Nginx, Microsoft-IIS; il più utilizzato attualmente è Apache [2].

Per il mio progetto di tesi, la mia scelta è ricaduta sul linguaggio Javascript e sull'utilizzo di *Node.js*. Node.js è una piattaforma open-source per lo sviluppo di web server e applicazioni in Javascript; utilizza la *Javascript Engine V8* di Google per compilare Javascript in linguaggio macchina nativo (anzichè interpretarlo in tempo reale); è un web-server *single-thread*, si basa sull'architettura *event-driven* con I/O asincrono, ed è multipiattaforma [3] [4].

Il fatto di essere single-thread è una notevole semplificazione alla scrittura delle web-app, e per avere le performance di una app multi-thread basta semplicemente avviare più istanze di Node.js contemporaneamente; in questo modo il parallelismo viene spostato dal livello applicativo al livello server [Bro14].

Le funzioni in Node.js sono *non bloccanti*: nei tradizionali ambienti di sviluppo, un comando viene eseguito solo nel momento in cui il comando precedente viene completato; in Node.js, invece, i comandi vengono eseguiti in parallelo, usando il meccanismo della *callback* per segnalare il completamento di una certa elaborazione (non bloccanti perchè appunto non bloccano lo svolgimento delle altre funzioni), facendo rimanere il server in modalità *sleep* fino all'arrivo di tale callback [5]. Questo particolare tipo di architettura

[1] https://w3techs.com/technologies/overview/programming_language/all
[2] https://w3techs.com/technologies/overview/web_server/all
[3] https://nodejs.org/en/
[4] https://en.wikipedia.org/wiki/Node.js
[5] https://it.wikipedia.org/wiki/Node.js

rende un web server in Node.js particolarmente performante in casi di alto traffico e di elevato numero di richieste I/O (questi studi [LMT14] dimostrano che il numero di richieste al secondo servite da un web server in Node.js può essere fino a 10 volte superiore, nei casi difficili di alto traffico, rispetto a web-server scritti in PHP e Python).

Un altro grosso vantaggio dell'utilizzo di Node.js sta nel non dover più effettuare il *context-switch* fra diversi linguaggi di programmazione: nasce quindi la figura del *Full Stack Developer* in Javascript, in grado di curare sia il front-end che il back-end di una web-app.

Tutti questi fattori positivi rendono Node.js il candidato ideale per la creazione di un web server per la fruizione di documenti legali: non si avranno più problemi di lentezza del servizio (che purtroppo alcune volte possiamo incontrare nell'utilizzo di EUR-Lex).

4.2 L'importanza delle API REST

In informatica, il termine *API* (acronimo di *Application Programming Interface* indica un insieme di funzioni disponibili (generalmente al programmatore) raggruppate in modo da formare un'interfaccia per lo svolgimento di un determinato compito in un programma [6]. Il termine *REST* (acronimo di Representational State Transfer) indica invece un particolare tipo di architettura software per lo scambio di dati su internet: non si parla di un protocollo o di uno standard *de facto*, ma di un metodo per sviluppare servizi web favorevole dal punto di vista delle performance, della scalabilità e della semplicità d'uso [7].

Col termine API REST, quindi, si identifica un'interfaccia che raggruppa una serie di funzioni che permettono lo scambio e la modifica di risorse, rispettando i vincoli architetturali REST che sono [Fie00]:

Client-Server - ci deve essere una netta separazione in stile architetturale

[6] https://en.wikipedia.org/wiki/Application_programming_interface
[7] https://en.wikipedia.org/wiki/Representational_state_transfer

client-server (ad esempio, il client non si deve occupare della memorizzazione dei dati e il server non si deve occupare dell'interfaccia utente).

Stateless - ogni richiesta dal client contiene tutto il necessario per richiedere una specifica risorsa, e non ha nessuna relazione con le richieste precedenti o successive; lo stato della sessione è contenuto sul client, e sul server non viene salvato nulla riguardo al contesto del client.

Cacheable - il client può avere la possibilità di fare caching delle risposte del server (la risposta deve essere esplicitamente dichiarata chacheable in modo da evitare che i client possano utilizzare dati non aggiornati).

Layered System - il client non può sapere se è connesso direttamente ad un server di livello basso o intermedio (si possono aggiungere server intermedi per migliorare la scalabilità o, per esempio, offrire politiche di sicurezza).

Uniform Interface - l'interfaccia che collega le varie componenti deve essere uniforme, trasferendo i dati in un formato standard piuttosto che in un formato adatto ad uno specifico client (nella sottosezione seguente specifico quali sono i vincoli per ottenere un'interfaccia uniforme).

4.2.1 La chiave dell'architettura REST: la risorsa

La risorsa, in un'architettura REST, è il concetto chiave di astrazione dell'informazione. Qualsiasi fonte di informazione, qualsiasi oggetto a cui un ipertesto può fare riferimento, può essere una risorsa; un documento, un'immagine, una persona, un servizio, ecc. : tutto ciò può essere identificato come risorsa. È opportuno precisare che una risorsa non identifica la rappresentazione fisica/virtuale di uno specifico oggetto (in quanto ci possono essere risorse che possono variare nel tempo), ed è proprio per questo che si parla di astrazione dell'informazione [Fie00].

Nel quinto vincolo REST, *Uniform Interface*, entrano in gioco le risorse, e sono stati definiti quattro criteri [RAAR13]:

Identification of resources - ogni risorsa è identificata tramite uno specifico URI univoco (la risorsa, non la specifica rappresentazione).

Manipulation of resources through representation - il client, se autorizzato, quando possiede la rappresentazione di una risorsa, deve avere tutte le informazioni necessarie per modificarla o cancellarla.

Self-descriptive messages - ogni messaggio inviato dal server deve contenere dei metadati che descrivono come deve essere processato.

Hypermedia as the engine of application state - le risposte del server possono includere degli URI ad altre risorse (ad esempio, parlando di un libro, informazioni come autore ed editore sono specificati tramite URI) e il client deve essere in grado di interpretarli correttamente.

Generalmente, per effettuare una specifica operazione su una risorsa, il client fa richiesta sempre allo stesso URI (URL), cambiando semplicemente il metodo HTTP della richiesta:

- GET, utilizzato per recuperare informazioni su una risorsa o collezione, è considerato un metodo "sicuro" in quanto non altera nulla nella base di dati;

- PUT e DELETE, usati rispettivamente per modificare e cancellare una risorsa, sono considerati "idempotenti", in quanto alterano la base di dati una sola volta, anche con successive richieste identiche.

- POST viene solitamente usato solo in riferimento ad una collezione, per ottenere il link alla nuova risorsa creata da modificare.

Un portale di documenti legali, in quanto fruitore di informazioni, dovrebbe essere implementato seguendo l'architettura REST.

4.3 Semplificare e migliorare l'interfaccia con Semantic-UI

Fra le problematiche riscontrate nell'utilizzo di EUR-Lex, ho constatato il fatto che l'interfaccia del portale non è stata progettata in modo *responsive*. La locuzione *"Responsive Web Design"* è stata coniata nel 2010 da Ethan Marcotte, in un articolo sulla rivista *A List Apart* [8]: consiste nella tecnica di web design di realizzazione di un sito con contenuto che si adatta automaticamente (e graficamente) alle specifiche tecniche del dispositivo sul quale viene visualizzato; di conseguenza, gli utenti che navigano con diversi browser e periferiche hanno accesso allo stesso file sorgente, che però gestisce la visualizzazione dei contenuti in modo da renderli sempre di facile consultazione, evitando operazioni come lo scorrimento orizzontale e il ridimensionamento [Nat13].

Per attuare le caratteristiche appena citate vengono utilizzate le seguenti tecniche:

griglia flessibile - tutti gli elementi del sito vengono dimensionati utilizzando unità relative (ad esempio percentuali, em e rem[9] piuttosto che unità assolute (come pixel, pollici e punti).

immagini flessibili - la dimensione delle immagini deve poter cambiare, in modo da adattarsi all'impaginazione evitando sovrapposizioni con altri elementi.

media query - è un modulo CSS3 che permette l'adozione di diversi stili CSS in base ad alcune specifiche tecniche del dispositivo (ad esempio la larghezza dello schermo in pixel, la densità e l'*aspect ratio*).

La tecnica del Responsive Web Design ha assunto sempre più importanza col passare degli anni; ricerche e statistiche di vendita dimostrano che, a

[8] http://alistapart.com/article/responsive-web-design

[9] l'unità di misura *em* fa riferimento alla dimensione del font dell'elemento attuale, mentre l'unità *rem* fa riferimento alla dimensione del font dell'elemento radice

partire dal 2012, il numero di dispositivi mobili venduti è superiore rispetto al numero di notebook e desktop pc [Nat13]. Nella tabella, invece, mostro le statistiche riguardo alle risoluzioni di schermo più diffuse in Europa nell'ultimo anno: le risoluzioni desktop standard sono affiancate ad alcune delle numerose risoluzioni mobile[10].

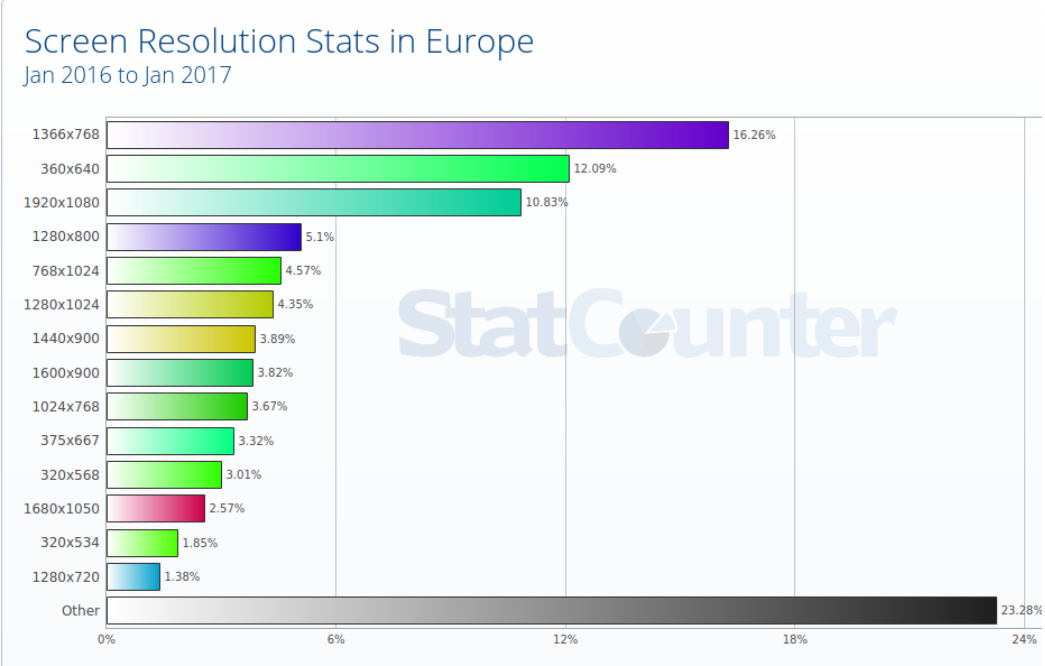

Figure 4.1: Risoluzioni video più comuni in Europa nel 2016

Al giorno d'oggi, data la quantità di persone che naviga in internet utilizzando un dispositivo mobile, è impensabile la creazione dell'interfaccia di un sito senza l'adozione di tecniche di Responsive Web Design. Per fortuna, a partire dal 2011 sono iniziati a nascere una serie di *framework CSS*, ovvero delle librerie che possono essere utilizzate per semplificare il web design favorendo l'adozione di tecniche per il responsive; in genere sono composte da [Jai15]:

- codice CSS per la creazione della griglia, per poi posizionare ciascun elemento all'interno di essa;

[10]fonte: gs.statcounter.com

- definizioni tipografiche per gli elementi HTML;
- codice alternativo per risolvere problemi di incompatibilità nei browser;
- classi CSS predefinite che possono essere utilizzate per stilare gli elementi.

I vantaggi di utilizzare un framework CSS sono diversi: si imparano tecniche efficaci per la soluzione di problemi di web-design, non bisogna preoccuparsi di problemi di compatibilità dei browser, si velocizza il processo di mockup, si tiene un codice pulito ed ordinato, ed è più facile il coordinamento dei membri di un team per quanto riguarda la creazione degli elementi e dei loro relativi stili [Jai15].

Per la creazione del mio prototipo di portale di documenti legali ho scelto di affidarmi a *Semantic-UI*, un framework CSS abbastanza famoso (oltre 30000 stelle su GitHub) "basato sui principi del linguaggio naturale" [11], che contiene tutte le caratteristiche classiche di un framework che ho citato nel paragrafo precedente, con l'aggiunta di una serie di elementi "esclusivi" già predefiniti e da poter includere facoltativamente nel modulo finale. Dispone inoltre di un'ottima documentazione.

4.4 Velocizzare il retrieval con richieste AJAX

Il termine *AJAX*, in informatica, rappresenta l'abbreviazione di *Asynchronous Javascript and XML*; è stato coniato nel 2005 da Jesse James Garrett, ispirato dalle tecniche utilizzate da Google per i servizi Gmail e Google Maps: indica una serie di tecnologie che vengono adoperate in modo appropriato per permettere che l'interazione fra l'utente e il sito avvenga in modo asincrono, indipendentemente dalla comunicazione col server[12].

Il modello classico di sito web, fino alla fine degli anni 90, era il seguente: ogni azione effettuata dall'utente comportava la generazione di una richiesta

[11] http://www.semantic-ui.com/

[12] en.wikipedia.org/wiki/Ajax_(programming)

HTTP al server; il server elaborava la richiesta e rispondeva con una pagina HTML contenente le nuove informazioni. Questo approccio era ottimo per la fruizione di contenuti statici, ma poco adatto per i siti web dinamici: il server veniva sfruttato per reinviare l'intera pagina HTML, anche se a subire le modifiche fosse una piccola parte di essa (basti pensare anche al semplice layout del sito che non cambia e viene inviato più volte), con conseguente spreco di banda; inoltre, anche la *user experience* risentiva di questo approccio, in quanto l'utente vedeva scomparire tutto il contenuto per poi riapparire con le modifiche. [G+05]

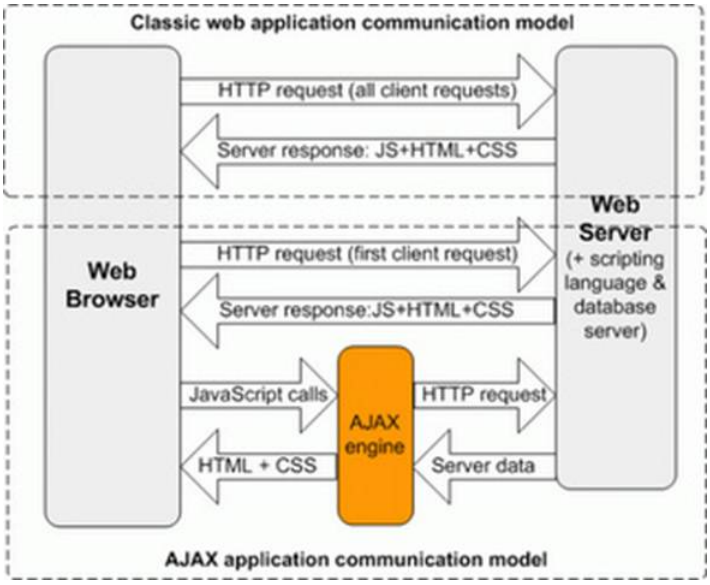

Figure 4.2: Modello classico (in alto) e modello AJAX (in basso) - fonte: bhartisoftland.com

Le tecnologie raggruppate dall'architettura AJAX sono le seguenti [G+05]:

- rappresentazione dell'informazione tramite HTML e CSS;
- manipolazione del *Document Object Model* (DOM);
- interscambio di dati in un formato standard (ad esempio XML o JSON);
- recupero asincrono delle informazioni tramite l'oggetto *XMLHttpRequest* (XHR);

- il linguaggio Javascript per unire tutte queste tecnologie.

Il cuore dell'architettura è proprio l'oggetto *XMLHttpRequest* che contiene delle API che permettono di scambiare le informazioni fra client e server in modo asincrono: è stato introdotto a partire dal 2000 ed è stata la prima soluzione "elegante" a questo tipo di problema[13]. I suoi metodi più importanti sono [LWZZ08]:

open per costruire una nuova richiesta;

send per inviare la richiesta al server;

abort per cancellare la richiesta corrente;

responseText che contiene la risposta del server;

readyState è la proprietà che esprime lo stato attuale della richiesta (0 per la richiesta non inizializzata, 1 per la richiesta inizializzata, 2 per la richiesta inviata, 3 per la richiesta in fase di elaborazione e 4 per richiesta terminata).

L'architettura AJAX introduce quindi la *Ajax Engine* che funge da intermediario, inviando le richieste al server e gestendo le risposte ricevute tramite il meccanismo delle *callback*. Ho utilizzato AJAX nel mio progetto di tesi per creare un'interfaccia di ricerca stile Google, fornendo all'utente dei suggerimenti di ricerca in base a quanto digitato; ho implementato AJAX tramite *jQuery*, una libreria in Javascript che semplifica una serie di operazioni importanti e ricorrenti nella programmazione client-side (come appunto l'Ajax Engine e la manipolazione del DOM).

4.5 L'unione di tutte le tecnologie

In questa sezione illustro a grandi linee tutte le tecnologie che ho utilizzato per la realizzazione del mio prototipo di portale per la distribuzione di documenti legali.

[13] dal blog di Aaron Swartz: www.aaronsw.com/weblog/ajaxhistory

Per quanto riguarda il web server ho deciso di utilizzare Node.js: nella sezione 4.1 ho illustrato tutti i vantaggi di questa moderna tecnologia, che la rendono ideale per la realizzazione di una applicazione web *I/O-intensive*; mi sono servito anche di *Express*, un framework per Node.js che facilita la creazione di una web-application, favorendone la manutenibilità e la modularizzazione del codice [Bro14].

Per quanto riguarda l'interfaccia, ho utilizzato sia un *template engine* che un *front-end framework*. Il template engine è l'anello di congiunzione fra presentazione e informazioni: è in grado di riempire dei template con dei dati recuperati dinamicamente, favorendo il principio di "separazione degli interessi": il programmatore si occupa di mettere a disposizione i dati grazie a dei metodi, mentre il web designer si occupa dell'interfaccia, inserendo delle variabili particolari al posto delle effettive informazioni [Bro14]; come template engine ho utilizzato *Handlebars*. Nella sezione 4.3 ho illustrato l'utilità dei front-end framework; per il mio prototipo ho utilizzato Semantic-UI (sezione 4.3).

Per quanto riguarda la memorizzazione dei dati mi sono servito di due soluzioni: *eXist-DB* e *MongoDB*. Entrambi sono due DBMS *NoSQL* orientati al documento: come ho accennato nella sezione 3.1, in questo tipo di basi di dati le informazioni non vengono memorizzate utilizzando il classico modello relazionale, bensì i documenti consistono in record che non hanno necessariamente uno schema restrittivo. eXist-DB ha la peculiarità di essere anche un database XML nativo, permettendo infatti l'utilizzo di XQuery e XSLT per l'interrogazione nei propri documenti: l'ho utilizzato perchè sarà molto utile per operare sulle future rappresentazioni XML dei documenti legali. Ho utilizzato MongoDB, invece, per la memorizzazione dei metadati di ciascun documento: studi dimostrano che, confrontato con un DBMS relazionale, "MongoDB è in grado di operare facilmente su uno schema dinamico come un sistema di gestione di documenti con diversi campi dinamici e pochi campi su cui effettuare la ricerca", ottenendo ottime prestazioni anche in schemi più rigidi con query semplici e basate sulle chiavi [PPV13].

4.6 L'implementazione del portale

In questo capitolo ho analizzato, una ad una, tutte le tecnologie che faranno parte del prototipo di portale per la distribuzione di documenti legali. Sono partito dal web server *Node.js*, dimostrandone le peculiarità ed i pregi rispetto ad altri web server classici ampiamente utilizzati nel web. Successivamente ho introdotto il concetto di API REST ed evidenziato l'importanza della *risorsa*, la chiave di questo tipo di architettura. Nella terza sezione ho parlato dell'interfaccia, introducendo la locuzione *front-end framework*, ed illustrando i pregi dell'utilizzo di *Semantic-UI*. Nella quarta sezione ho parlato delle chiamate AJAX, una pratica già ampiamente diffusa sul web per permettere che l'interazione fra l'utente e il sito avvenga in modo asincrono, indipendentemente dalla comunicazione col server. Infine, nella quinta sezione, ho riassunto tutte le tecnologie che compongono il portale, di cui nel prossimo capitolo ne spiegherò l'effettiva implementazione.

Capitolo 5

Fmx2Akn, l'implementazione del portale

Il quinto capitolo di questa dissertazione si occupa dell'implementazione del prototipo di portale **Fmx2Akn**, ed è arricchito con porzioni di codice e screenshot delle schermate disponibili. La prima sezione fornisce una panoramica dell'architettura del portale, specificando la funzione di ogni singolo file creato, la struttura delle cartelle, le *route* disponibili ed illustrando il codice dell'istanza di web server, il punto di partenza del portale. Nella seconda sezione viene illustrata l'interfaccia, elencando tutti gli elementi esclusivi del front-end framework che ho utilizzato e gli helper personalizzati che ho creato, specialmente per la visualizzazione della tabella dei risultati. La terza sezione descrive il file *eurlex.js*, una libreria da me creata ricca di funzioni in grado di poter manipolare i metadati contenuti in CELLAR. La quarta sezione fornisce un'analisi dettagliata dell'utilizzo del portale, e di cosa succede sia nel front-end che nel back-end al momento della visualizzazione della home page, della digitazione dei caratteri, dell'avvio della ricerca e della visualizzazione del risultato finale. Infine, nella quinta sezione, vengono illustrate altre funzionalità importanti a cui però non è possibile accedere tramite interfaccia.

5.1 L'architettura del portale

In questa sezione spiego come ho strutturato il prototipo di portale, illustrando i file creati, il loro percorso e il loro scopo. Li elenco:

fmx2akn.js - è il cuore del web server, il file da associare all'istanza di Node.js;

db.js - contiene le funzioni relative alla connessione, visualizzazione ed aggiornamento dati al database MongoDB (in cui sono contenuti metadati relativi alle direttive europee);

hbshelpers.js - questo file contiene gli *helper*, delle funzioni che vengono date in pasto al template engine permettendogli di popolare i layout dell'interfaccia con i dati effettivi;

***.json** - sono file che contengono parametri di configurazione;

public/ - in questa cartella sono contenuti tutti i file statici da inviare al client (immagini, fogli di stile, javascript lato client e file relativi all'interfaccia con Semantic-UI);

routes/ - ciascun file .js in questa cartella rappresenta una specifica *route* disponibile sul web server;

views/ - file relativi alle *view* e ai *layout*, ovvero gli scheletri di interfaccia che verranno riempiti grazie al template engine;

node_modules/ - qui sono contenute tutte le librerie aggiuntive del web server, non scritte da me, recuperate grazie al gestore di pacchetti *npm*;

data/ - in questa cartella verranno salvati tutti i file relativi ai processi di download delle direttive europee, recuperati da EUR-Lex;

utils/ - cartella che contiene sia la libreria per interfacciare Node.js con eXist-DB (recuperato sul web) che il file *eurlex.js*, in cui ho disposto

una serie di funzioni per estrapolare dati importanti riguardo ad una direttiva contenuta in CELLAR.

Il punto di partenza, *fmx2akn.js*, è costituito da un file molto piccolo, in quanto ho deciso di inserirci il minimo indispensabile solo per tirare su un'istanza di web server: inclusioni di alcuni moduli e librerie essenziali, file di configurazione, definizione delle varie *route* da seguire, del template engine, e infine connessione col database MongoDB (la creazione dell'istanza del server è vincolata dalla buona riuscita della connessione al database).

Listing 5.1: Il file *fmx2akn.js*, il punto di partenza del web server

```
var fs = require('fs');
var express = require('express');
var exphbs  = require('express-handlebars');
var XML = require('pixl-xml');
var db = require(__dirname+'/db.js');
var cfg = JSON.parse(fs.readFileSync(__dirname + '/config.
   json'));
var homeRoute = require(__dirname+'/routes/home.js');
var serveRoute = require(__dirname+'/routes/serve.js');
var downloadRoute = require(__dirname+'/routes/download.js');
var proxyRoute = require(__dirname+'/routes/proxy.js');
var suggestRoute = require(__dirname+'/routes/suggest.js');
var app = express() ;
app.set('port', process.env.PORT || cfg.expr_port);
app.engine('.hbs', //SETTING THE TEMPLATE ENGINE
  exphbs({
    defaultLayout: 'main',
    extname: '.hbs',
    helpers: require(__dirname+'/hbshelpers.js')
  }));
app.set('view␣engine', '.hbs');
app.use(express.static('public')); //STATIC RESOURCES
app.get('/' , homeRoute);
app.get('/serve/:celex/:opt?', serveRoute);
app.get('/download/:celex/:par1?/:par2?', downloadRoute);
app.get('/proxy/:id/:dt/:num/:ln/:f', proxyRoute);
```

```
app.get('/suggest/:input', suggestRoute);
app.get('/dlfeed/:celex', function(req, res) {
  var clx = req.params.celex;
  db.feedDl(clx, function(err, dbObj) {
    res.json(dbObj);
  })
})
db.connect(cfg.mongo_url, function(err) {
  if (err) { //UNABLE TO CONNECT TO MONGO
    process.exit(1)
  } else {
    app.listen(app.get('port'), function(){
      console.log( 'Fmx2Akn! Express started on http://
   localhost:' +
      app.get('port') + '; press Ctrl-C to terminate.' );
    });
  }
});
```

Spiego brevemente le route che ho inizializzato e a cosa servono:

/ - rappresenta la home del portale, con la barra di ricerca al centro della pagina ed una porzione in basso che indica gli ultimi documenti aggiunti al database;

/serve/ - è la route che entra in gioco nel momento in cui effettuiamo una ricerca, producendo come risultato finale una tabella con l'elenco di tutti i file relativi al documento; prende in input un ID CELEX e un altro parametro opzionale (per restituire, ad esempio, solo il *Tree Notice* del documento);

/download/ - questa route rappresenta una scorciatoia per il download di un singolo documento (specificando lingua e formato) oppure di un archivio contenente più file che soddisfano un certo requisito (ad esempio tutti i formati per una specifica lingua);

/proxy/ - è la route per scaricare un file specifico, utilizzata come col-

legamento per il download dei vari file nella tabella del risultato della ricerca;

/suggest/ - è la route che accoglie la richiesta AJAX per fornire un suggerimento di risultato durante la digitazione del CELEX;

/dlfeed/ - route che viene invocata per offrire un feed all'utente riguardo allo stato attuale del recupero dei dati da EUR-Lex (dopo aver digitato il celex).

5.2 Come è strutturata l'interfaccia

Ho realizzato l'interfaccia del mio prototipo di portale grazie all'utilizzo di *Semantic-UI*, un ottimo *front-end framework* che presenta tutte le caratteristiche citate nella sezione 4.3 di questa dissertazione. Ho ottenuto un'interfaccia fluida, responsive e chiara grazie ad una serie di elementi previsti dal framework, che ora illustro.

Sono partito dal *container*, un elemento progettato per contenere gli elementi della pagina entro una larghezza massima ragionevole in base alla dimensione dello schermo dell'utente. È molto utile per limitare la dimensione di qualsiasi elemento, come menù e griglie, ma anche del semplice testo (nel caso del testo in una singola colonna, la larghezza massima del container, soprattutto in schermi più grandi è ulteriormente ridotta). Tutte le pagine del portale sono racchiuse nell'elemento container.

Un altro elemento importante che ho utilizzato è il *grid* (griglia), creato per permettere di far scorrere il contenuto in un modo più naturale. La griglia divide lo spazio orizzontale in unità invisibili dette "colonne". Semantic-UI divide di default lo spazio in 16 colonne; a ciascuna colonna è assegnata la larghezza come proporzione dello spazio disponibile nella "riga". La riga raggruppa un insieme di colonne, e se non dichiarata, viene creata una nuova riga quando non c'è più spazio disponibile. La griglia gioca un ruolo fondamentale sul posizionamento degli elementi nella pagina, permettendo di affiancare gli

elementi oppure di posizionarli verticalmente in base alla dimensione dello schermo dell'utente.

Oltre ai principali elementi griglia e container, ho utilizzato diversi altri elementi, progettati per un'ottima esperienza visiva ed una completa personalizzazione:

menu - utilizzato per il menù in alto nell'header della pagina e per mostrare il contenuto dell'archivio Formex nella tabella dei risultati;

input - utilizzato per il campo di ricerca

dimmer/loader - presenti nell'input di ricerca: ho utilizzato il *dimmer* per oscurare il campo di ricerca, dando enfasi al *loader*, che consiste in un'immagine che mostra il caricamento di una risorsa;

segment - per raggruppare gli ultimi documenti aggiunti al database in un "segmento" nella home page;

divider - per dividere gli elementi di una pagina tramite una sottile barra, aggiunta nella pagina dei risultati per separare il titolo del documento, i metadati e la tabella con l'elenco dei file;

table - per ottenere facilmente una tabella ben formattata;

label - utilizzato nella pagina dei risultati per evidenziare in alto i principali metadati del documento;

icon - tramite un porting di fontawesome.io sono disponibili numerose icone scalabili vettoriali, tutte in un unico font;

flags - ho utilizzato le bandiere degli stati membri dell'UE nella tabella dei risultati per indicare la lingua dei vari file; le bandiere sono tutte inserite in un'unica immagine, riducendo il numero di richieste al server e salvando banda[1]

[1] Questa tecnica prende il nome di *image sprite*.

Tutto il contenuto del portale è servito all'utente tramite *Handlebars*, un template engine che rappresenta l'anello di congiunzione fra presentazione ed informazioni (accennato nella sezione 4.5). Handlebars, per renderizzare un template, ha bisogno sia del template stesso che di un oggetto JSON chiamato "contesto": nel contesto sono presenti tutte le informazioni necessarie da inserire nella pagina. Nel template, che consiste in un normale HTML, per inserire un elemento dinamico dal contesto basta richiamare il nome della specifica chiave nell'oggetto del contesto: la chiave si scrive fra doppie parentesi graffe nel documento HTML (3 parentesi se l'oggetto contiene codice HTML). È possibile, inoltre, eseguire delle operazioni con i dati presi dal contesto, per visualizzare del contenuto specifico: gli *helper* sono delle funzioni speciali utilizzate dal template engine in fase di renderizzazione che servono a tale scopo; ci sono degli helper forniti già da Handlebars, ad esempio per costrutti basilari come if/else e forEach, ma è possibile crearne anche di nuovi.

Handlebars divide il template in due parti: *view* e *layout*. La view rappresenta una singola pagina del sito (o ad esempio una porzione di pagina gestita da chiamate AJAX); il layout è una view particolare. Il layout serve ad inserire tutte quelle informazioni che sono utilizzare in più di una view (ad esempio tutte le inclusioni fatte nell'elemento `<head>` ma anche parti della presentazione come l'header e il footer della pagina).

Per il mio progetto ho creato 1 layout, 3 view e 7 helper personalizzati. Mi è bastato un unico layout in quanto le schermate fondamentali sono quella di ricerca e quella di visualizzazione risultato, ed entrambe hanno lo stesso contenuto nel tag `<head>` e presentano lo stesso header. Le view invece sono:

home - rappresenta la pagina iniziale, e contiene il nome del portale seguito da una grossa barra di ricerca (stile Google), il segmento che indica gli ultimi documenti aggiunti, e la porzione di pagina adibita a mostrare i suggerimenti di ricerca (non visibile prima della digitazione);

table - costituisce la pagina del risultato, e contiene il titolo del documento, i metadati importanti e la tabella con l'elenco dei file divisi per formato

e lingua;

error - una piccola view che viene renderizzata in caso di errori nella ricerca.

Listing 5.2: Il file di layout *main.hbs*

```
<!doctype html><html>
<head>
  <title>Fmx2Akn</title>
  <link rel="stylesheet" href="{{setInclPath}}/semantic/dist/semantic.min.css">
  <link rel="stylesheet" href="{{setInclPath}}/css/main.css">
</head>
<body>
  <div class="ui fixed inverted blue menu">
    <div class="ui container">
      <a id="brand" class="item" href=""><b>Fmx2Akn</b></a>
      <div id="navsrch" class="ui search item"></div>
      <div class="ui right dropdown item">
        <img src="{{setInclPath}}/img/menu_icon.png"></img>
        <div class="ui inverted blue menu">
          <a class="item" href="">Home</a>
          <a class="item" href="http://akomantoso.org/"
          target="_blank">Akoma Ntoso</a>
        </div>
      </div>
    </div>
  </div>
  <div class="ui grid main container">
    {{{body}}}
  </div>
  <script src="{{setInclPath}}/js/jquery-2.2.0.min.js"></script>
  <script src="{{setInclPath}}/semantic/dist/semantic.min.js"></script>
  <script src="{{setInclPath}}/js/main.js"></script>
</body>
</html>
```

Gli helper che ho creato sono 7:

setInclPath - utilizzato nelle inclusioni di fogli di stile e file JS, per impostare la cartella corretta come sorgente dei file;

engTitle - per visualizzare in alto, nella pagina dei risultati, il titolo in inglese del documento;

truncPath - utilizzato per troncare parti inutili dell'URL Akoma Ntoso;

consOrNot - per stabilire se il documento in questione è in realtà un consolidato oppure no;

setIcon - per impostare tutte le icone nella tabella dei risultati, con la giusta immagine in base al formato e il link corretto per il download;

consDropdown - per far apparire, nei metadati del documento, un menù che contiene l'elenco dei consolidati di tale documento oppure la dicitura del documento originale nel caso di un consolidato;

setFlag - per impostare tutte le bandiere, nella tabella dei risultati, per indicare la lingua dei file.

5.3 La libreria eurlex.js

Il file *eurlex.js* è molto importante, in quanto racchiude tutte le funzioni in grado di estrapolare le informazioni dal *Tree notice* in CELLAR, scaricare tutti i file sempre da CELLAR e preparare tutti i metadati del documento da inserire in MongoDB.

Prima di spiegare le varie funzioni che ho creato voglio descrivere brevemente un modulo che ho utilizzato più volte in questo file: *async*. Questo modulo è stato progettato per semplificare l'organizzazione di operazioni parallele che richiedono del tempo (*CPU bound* e *I/O bound*) in armonia con la natura asincrona di Javascript. Ho utilizzato questo modulo per semplificare il processo di download concorrenziale di più risorse da EUR-Lex, tramite

il metodo *mapLimit* che funziona in questo modo: `async.mapLimit(array, limit, iterator, done)`, dove `array` è il vettore che contiene tutte le informazioni sui file da scaricare (più campi di controllo), `limit` è il numero massimo di richieste (download) attivi, e `iterator` è la funzione che cura il singolo download e viene quindi "iterata", chiamando una propria callback ogni qual volta l'operazione termina; quando tutte le istanze di `iterator` terminano chiamando la propria callback, allora si passa all'esecuzione della funzione `done`. Il modulo prevede che se anche una sola istanza di `iterator` termina con errore, allora l'intero processo dei download si blocca: ho modificato questo comportamento, in quanto spesso possono accadere errori di timeout da EUR-Lex, chiamando quindi le callback comunque con esito positivo ma aggiungendo un campo all'oggetto del download che segnala tale errore, e un controllo per tale campo nella funzione `done`, in modo da poter successivamente scaricare solo i file con errori.

Dopo aver introdotto il modulo *async*, ora procedo con la spiegazione delle funzioni che ho creato, descrivendone lo scopo, l'input e l'output.

validClx - serve per dichiarare se un id CELEX possa essere valido prima di chiedere tale CELEX ad EUR-Lex, avvalendosi delle funzioni **validYear** e **validMainClx** (viene fatto un controllo sulla struttura dell'id, non sulla sua effettiva esistenza); restituisce un booleano.

getTreeNotice - si occupa di recuperare il Tree Notice da CELLAR, utilizzando il modulo esterno *request* per semplificare le chiamate HTTP e seguire eventuali reindirizzamenti; in caso di errore la funzione viene chiamata ricorsivamente (alcune volte ho riscontrato lentezza nella risposta da EUR-Lex, facendo andare le richieste in timeout); converte il Tree Notice in JSON e al temine passa il controllo alla callback.

getWorkInfo - estrapola i metadati essenziali del documento dal Tree Notice: tipo documento, anno, numero documento e data; restituisce questi metadati in un oggetto.

getFmxContent - è in grado di reperire il contenuto di un file Formex di

una specifica lingua (specificata in input) dal Tree Notice; restituisce un vettore che rappresenta l'elenco dei file dell'archivio (questa funzione viene invocata una volta per ogni lingua, dato che esiste un unico Formex per lingua).

prepDl - la funzione che prende in input il CELEX, il Tree Notice, l'oggetto contenente i metadati essenziali ed un parametro addizionale (per indicare se l'operazione di download deriva dalla ricerca dell'utente tramite interfaccia oppure dalla scorciatoia API REST tramite URL, per salvare i file in cartelle diverse); restituisce due vettori: il primo contiene un oggetto per ogni file da scaricare con tutte le informazioni necessarie al download; il secondo, invece, contiene informazioni da salvare in MongoDB (ogni oggetto rappresenta una Expression).

getConsArray - recupera tutti i consolidati disponibili per un certo documento, restituendoli in un vettore.

getConsTrees - si occupa di scaricare il Tree Notice di tutti i consolidati passati in input tramite un vettore di oggetti, grazie all'ausilio della async.mapLimit; la funzione iteratrice si chiama **consIterator**.

initConsArray - inizializza tutti i consolidati in questione in MongoDB, utilizzando sempre il modulo async, con la funzione iteratrice che si chiama **initConsIter**.

downloadFiles - procede al download di tutti i file in tutte le lingue e formati per un certo documento, le cui informazioni sono contenute in un array dato in input; il download avviene sempre grazie al modulo async, e la funzione iteratrice prende il nome di **fileIterator**.

createObj - crea l'oggetto che verrà salvato in MongoDB e che contiene tutte le informazioni post-download relative al documento, ottenute grazie all'esecuzione di buona parte delle funzioni precedenti.

checkParam - data una certa sigla, questa funzione riconosce se tale sigla rappresenta un formato di file, una lingua o nessuna delle due cose; questa funzione viene utilizzata nella route che prevede il recupero diretto di uno o più file tramite URL.

5.4 Utilizzo del portale

La schermata iniziale del portale contiene:

- l'header, sempre presente, con il link alla home ed un piccolo menù;
- la sigla che identifica il portale, al centro della pagina;
- la barra per effettuare la ricerca, posizionata subito al di sotto della sigla;
- una porzione che indica gli ultimi documenti aggiunti al database, con id CELEX cliccabile ed il titolo in inglese.

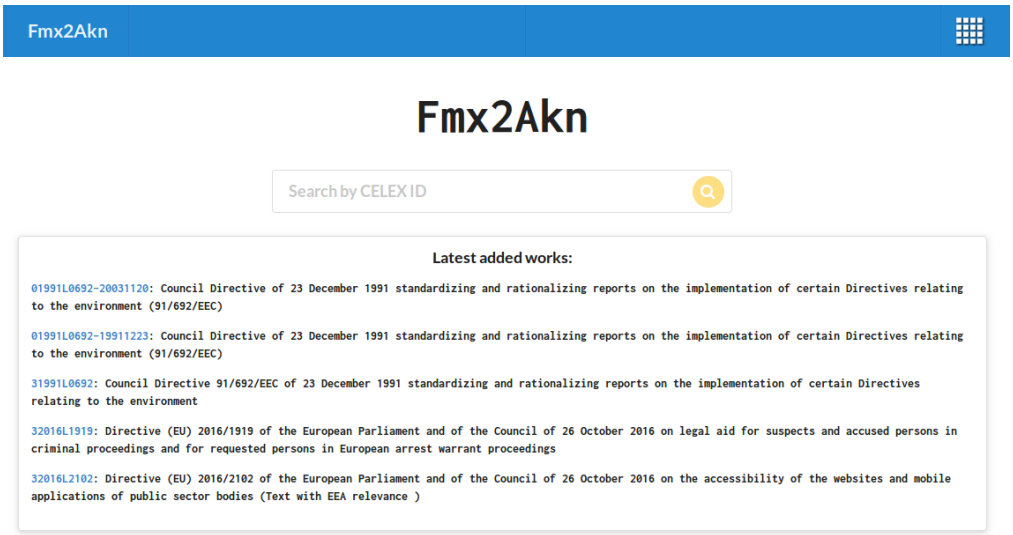

Figure 5.1: Home page del portale

Per quanto riguarda il back-end, il lavoro da fare per servire la home page è davvero poco, in quanto l'unico contenuto dinamico da caricare è appunto

l'elenco degli ultimi documenti aggiunti: basta semplicemente chiedere tale informazione al database MongoDB.

5.4.1 Cosa succede quando si digita nel campo di ricerca

Nel momento in cui l'utente inizia a digitare qualcosa nella barra di ricerca, l'interfaccia subisce una serie di trasformazioni (per queste trasformazioni mi sono ispirato a *Google*):

- nel caso di ricerca già effettuata, il div che contiene la tabella dei risultati viene rimosso dal DOM;
- il div che contiene la sigla del portale (Fmx2Akn) viene nascosto dal DOM;
- la porzione di pagina che contiene gli ultimi documenti aggiunti al database viene rimossa dal DOM;
- il div che contiene la barra di ricerca viene spostato nella barra del menù, contenuto nell'header della pagina;
- una volta spostata la barra di ricerca, è necessario far ritornare il *focus* su di essa.

Nel frattempo, dal client parte una richiesta AJAX con i caratteri digitati verso la route `/suggest/`, per recuperare i suggerimenti di risultato. Nel back-end, la route `/suggest/` prevede una semplice invocazione della funzione `searchSugg`, contenuta nel file *db.js*, che si occupa di effettuare la ricerca dei caratteri digitati in ogni documento in MongoDB, nel campo CELEX, restituendo i risultati in formato JSON.

Anche per la formattazione dei suggerimenti mi sono ispirato a *Google*: per ogni risultato compare l'id CELEX scritto in blu e cliccabile, subito dopo l'URI Akoma Ntoso e infine il titolo del documento in lingua inglese.

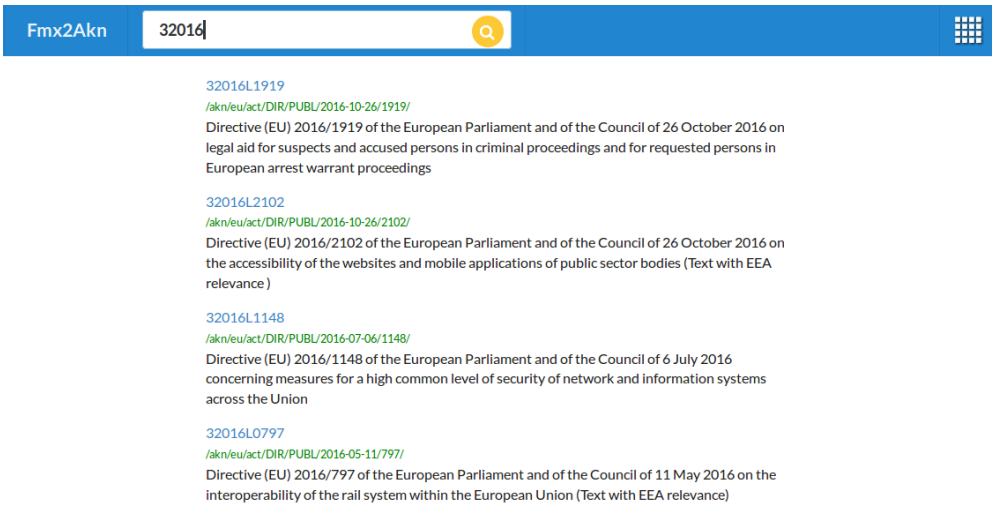

Figure 5.2: Visualizzazione dei suggerimenti durante la digitazione

5.4.2 Cosa succede quando si avvia la ricerca

Quando l'utente digita esattamente l'intero id CELEX di un documento, si possono presentare due situazioni:

1. il documento non è presente nel database poichè nessuno lo ha mai ricercato, (l'interfaccia suggerisce di effettuare la ricerca per aggiungere il documento al database);

2. il documento in questione è già presente nel database perchè qualcuno lo ha già ricercato almeno una volta, e in questo caso comparirà l'unico suggerimento per tale documento (unico in quanto l'id CELEX è un identificatore univoco).

In entrambi i casi la route che gestisce la richiesta è `/serve/`. Se è il documento non è presente nel database, allora entrano in gioco tutte le funzioni contenute nel file *eurlex.js*, in quanto è necessario recuperare tutte le informazioni utili dal *Tree Notice* per poi procedere col download di tutti i singoli file per ogni lingua e formato.

La struttura del file *serve.js* sfrutta molto il concetto di *middleware* di Express: il file è diviso in 9 parti, facendo eseguire a ciascun middleware

uno specifico punto del processo di scraping dell'intero documento; ciascuna parte deposita il risultato della propria computazione nell'oggetto `req`, e cede il controllo alla parte successiva tramite la funzione `next()`:

- la prima parte si occupa di validare l'input digitato, avvisando l'utente di eventuali errori di "costruzione" dell'id CELEX, e in caso positivo ricerca l'id nel database, inserendo il risultato nell'oggetto req;
- nel caso di risultato vuoto, la seconda parte inizializza l'id CELEX nel database, altrimenti passa al template engine il risultato trovato, che lo serve nella view *table* (questa sarebbe la seconda situazione, e quindi a questo punto il percorso della route finisce qui);
- la terza route controlla che il Tree Notice del documento non sia già presente nel file system, e in tal caso provvede a recuperarlo e a salvarlo (sia nel file system che nell'oggetto req);
- la quarta route serve il Tree Notice nel caso in cui sia richiesto questo servizio (è possibile interrogare tramite url: `/serve/CELEX/treenotice` per ottenere direttamente il Tree Notice in formato JSON piuttosto che il risultato con i file nella tabella), altrimenti provvede a recuperare i metadati principali e ad inserirli nell'oggetto req;
- la quinta, sesta e settima parte si occupano di recuperare tutte le informazioni e l'elenco dei file per i consolidati, se presenti;
- l'ottava parte avvia il download di tutti i file, e al termine memorizza l'oggetto completo del documento nel database, e in caso di assenza di consolidati, il percorso della route termina qui;
- l'ultima parte sistema i singoli oggetti relativi ai consolidati nel database, servendo infine la view col documento principale (e con un menù a tendina tramite il quale è possibile accedere facilmente ai consolidati).

Bisogna specificare che, se ci troviamo nella prima situazione, ovvero tutti i file relativi al documento devono ancora essere scaricati, il portale potrebbe

servire la richiesta in tempi molto lunghi (superiori al minuto), in quanto si tratta comunque di scaricare 3-4 formati di file diversi per ognuna delle 24 lingue di EUR-Lex: in tal caso la route `dlfeed`, richiamata sempre dal client con richiesta AJAX, si occupa di informare l'utente riguardo alla lunga attesa.

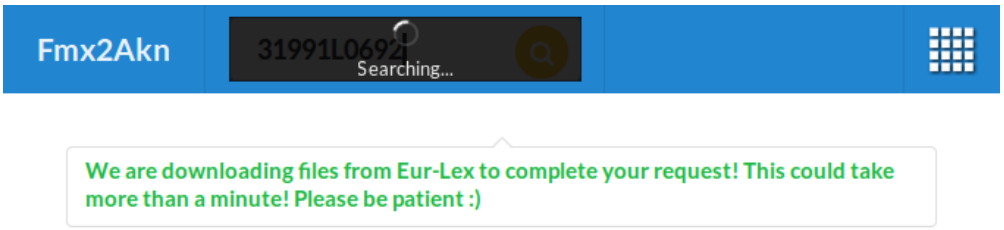

Figure 5.3: Il messaggio che notifica l'utente per l'attesa del download dei file.

Come si può notare dalla figura, nel momento in cui parte la ricerca entrano in gioco altri due elementi dell'interfaccia: il *dimmer* e il *loader*. Il dimmer permette di aggiungere uno strato "opaco" sulla barra di ricerca per mettere in evidenza il loader, ovvero l'immagine di caricamento, assieme alla scritta "Searching...". Inoltre, con l'attivazione di questi due elementi, la barra di ricerca si disattiva, rendendo impossibile il focus o la digitazione di altri caratteri.

5.4.3 Il risultato finale della ricerca

Nel momento in cui la ricerca termina, assieme all'eventuale download di tutti i file collegati, il risultato finale viene servito dal template engine tramite la view *table*, passando come contesto l'intero oggetto riguardo ad un singolo documento che viene memorizzato nel database MongoDB.

L'interfaccia della schermata del risultato finale è divisa in 3 parti:

- nella prima parte è presente il titolo in inglese del documento, inserito in un *container* testo per limitarne la larghezza massima;
- nella seconda parte sono presenti altre informazioni principali, ovvero l'id CELEX, la data del documento, il suo URI Akoma Ntoso e gli

eventuali consolidati; ciascuna di queste informazioni è inserita nell'elemento di interfaccia *label*, per poterle mettere in risalto, mentre per quanto riguarda i consolidati, se presenti, sono elencati in un *drop-down menù*, attivabile con un click sulla specifica label;

- la terza parte è costituita da una grossa tabella; ogni riga rappresenta una specifica lingua e contiene: la sigla della lingua, una bandierina per un riferimento visivo più istantaneo, l'uri AKN di ciascuna *expression* e le icone per ogni formato di download disponibile (XHTML, HTML, PDF, Formex, RDF) oppure una croce rossa nel caso in cui il file in quella specifica lingua e formato non sia disponibile.

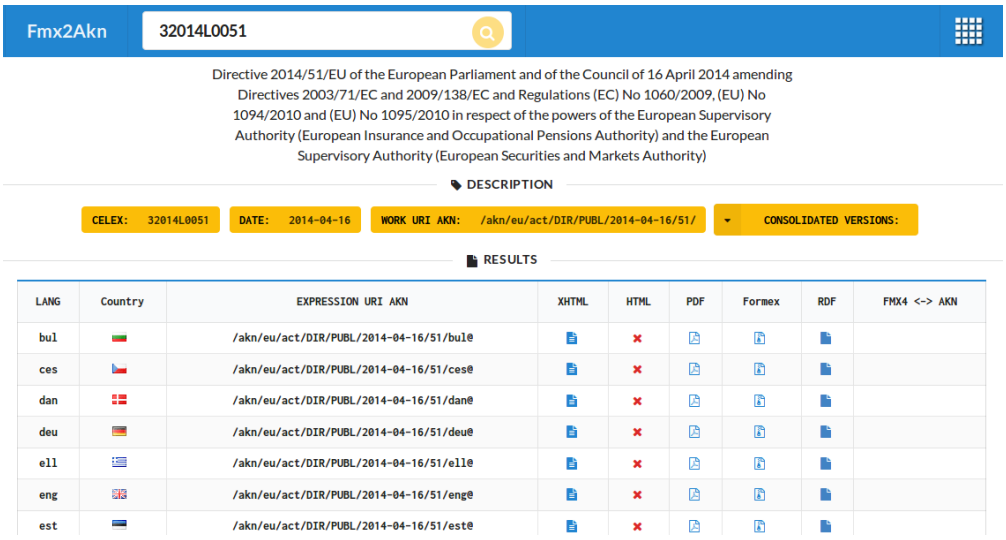

Figure 5.4: La schermata del risultato finale di ricerca.

Le tre porzioni di pagina sono suddivise dall'elemento *divider*, che aggiunge una sottile linea orizzontale di separazione assieme ad un'indicazione centrale relativa al contenuto. Lo sfondo delle righe della tabella è di due diverse sfumature in modo alternato, per una migliore leggibilità; le righe sono ordinate per la colonna della sigla della lingua, in ordine alfabetico. L'ultima colonna della tabella è riservata alla visualizzazione *side-by-side*, attualmente non disponibile, e quindi vuota (ne parlerò nella sezione 5.5.3).

I link dei singoli file nella tabella non indicano l'effettiva posizione in cui tali file sono salvati nel server, bensì indirizzano la richiesta alla route `/proxy/` che, conoscendo tutti i parametri del database eXist-DB, è in grado di ricostruire il link effettivo e servire i file in formato XHTML, HTML e PDF in una nuova scheda del browser, mentre per i file in formato Formex e RDF appare un prompt per il download.

Listing 5.3: Modulo esportato dal file *proxy.js*

```
var proxyRoute = [
  function (req, res) { //PARAMS CONTAINED IN REQUEST URL
    var clx = req.params.id;
    var date = req.params.dt;
    var num = req.params.num;
    var lan = req.params.ln;
    var file = req.params.f;
    var realurl = ('http://'+cfg.edb_opt.host+':'+cfg.edb_opt
   .port
      +cfg.edb_opt.rest+cfg.edb_dbpath
      +date+'/'+num+'/'+lan+'/'+file); //THE REAL URL
    var fileext = file.split('.')[1];
    console.log('fileext␣=␣'+fileext);
    switch (fileext) {
      case 'rdf':
        res.set("Content-Type" , "application/octet-stream");
        res.set("Content-Disposition", "attachment;filename="
   +clx+"-"+file);
        break;
      case 'fmx4':
        res.set("Content-Type" , "application/octet-stream");
        res.set("Content-Disposition", "attachment;filename="
   +clx+"-"+file+".zip");
        break;
      default: break;
    }
    proxy.web(req, res, { 'target': realurl });
  }
];
```

5.5 Altre funzionalità

Le prossime sottosezioni illustrano delle funzionalità del portale che non sono accedibili tramite interfaccia: il recupero di uno o più file tramite l'invocazione della route `/download/` senza il bisogno di ricercare il documento tramite interfaccia e di attendere la generazione della tabella dei risultati; l'aggiunta automatica al database di tutte le direttive emanate in un certo anno specificato (questa funzionalità è disposta in un web server separato); un prototipo di visualizzazione affiancata delle istanze FORMEX ed AKOMA NTOSO dello stesso documento, per evidenziare la traduzione da un formato all'altro dei vari elementi che lo compongono.

5.5.1 Recuperare direttamente uno o più file tramite URL

Questa importante funzionalità permette il download di un singolo file o di un insieme di file serviti in un archivio che soddisfano i parametri in input selezionati dall'utente, tramite digitazione nella barra degli indirizzi, senza quindi passare per l'interfaccia, ovvero senza effettuare la ricerca del documento ed attendere la generazione della tabella dei risultati; tale funzionalità è gestita dalla route `/download/`.

Questa route, oltre ad importare il file *eurlex.js* ed utilizzare le funzioni presenti al suo interno, definisce altre tre funzioni:

prepFiles - questa funzione, prendendo in input un intero oggetto memorizzato in MongoDB riguardante un singolo documento, e i parametri lingua, formato e nome file, è in grado di filtrare il vettore di *expression* presente nell'oggetto e di restituire un vettore di indicazioni per il download dei file che soddisfano i parametri della richiesta;

dlExist - questa funzione si occupa di recuperare i file che sono già disponibili sul database eXist, tramite l'ausilio della *async.map*; la funzione iteratrice si chiama **dlEIter**;

fileFilter - questa funzione è simile alla prima, però procede col filtraggio dei file prendendo in input non l'oggetto memorizzato su MongoDB, ma l'oggetto intermedio generato dalla funzione definita in *eurlex.js* per il download dei file.

La tabella seguente illustra, tramite esempi, che tipo di richieste è in grado di gestire questa route e cosa viene restituito per ogni esempio di richiesta.

Esempio URL	Risultato
/download/32016L0097	Archivio con i file di tutte le lingue e in tutti i formati
/download/32016L0097/ita	Archivio con i file in lingua italiana e in tutti i formati disponibili
/download/32016L0097/pdf	Archivio con i file in formato PDF in tutte le lingue
/download/32016L0097/eng/fmx	Singolo file FORMEX in lingua inglese
/download/32016L0097/fmx/eng	Singolo file FORMEX in lingua inglese

Table 5.1: Esempi di richieste gestite dalla route /download/.

Anche la struttura di questo file sfrutta il concetto di *middleware* di Express, dividendolo in 6 porzioni; ciascuna esegue una parte della computazione e cede il controllo alla porzione successiva tramite la funzione *next()*:

- la prima parte si occupa della validazione del CELEX e dei parametri passati, stabilendo se si tratti effettivamente della sigla di una lingua o di un formato, dando errore se non c'è corrispondenza per la sigla oppure se vengono specificate due sigle dello stesso tipo (due lingue o due formati);
- nella seconda parte si effettua la ricerca in MongoDB, e in caso di risultato vuoto, si inizializza un nuovo oggetto;

- la terza parte procede col filtraggio del vettore contenuto in MongoDB per ottenere solo i file di interesse, e successivamente recupera tali file dal database eXist;

- da qui in poi si ha una semplificazione delle operazioni svolte anche nella route `/serve/`: si recupera il Tree notice (se non presente), e si procede al download dei file da EUR-Lex;

- l'ultima parte si occupa di servire il risultato finale direttamente (se si tratta di un singolo file) oppure di inserire i file di interesse in un archivio .zip e di restituirlo all'utente.

5.5.2 Aggiungere al database tutte le direttive di uno specifico anno

Questa funzionalità è stata inserita in un web server separato, il cui nome è *webscr.js*: questo file si occupa di recuperare tutte le direttive europee emanate in uno specifico anno. Questa istanza è dipendente dall'istanza principale *fmx2akn.js*, in quanto l'effettivo download delle direttive avviene richiamando la route del portale che si occupa del download di ogni singolo file da EUR-Lex.

La funzionalità aggiuntiva di questo file consiste nella possibilità di fare uno *scrape* del codice HTML della pagina dei risultati di EUR-Lex, per poter estrapolare l'id CELEX di ciascuna direttiva, ovvero l'unica informazione essenziale per procedere con il download. A tale pagina dei risultati si può accedere direttamente dalla home page di EUR-Lex, in cui è possibile fare una ricerca specificando solo l'anno ed il tipo di documento.

Il modulo aggiuntivo utilizzato solo in questa porzione di codice si chiama *cheerio*, e serve appunto a semplificare il processo di estrapolazione delle informazioni da una pagina HTML. Le funzioni create sono due:

scrapeEurLex - si occupa di scansionare tutte le pagine di risultati per trovare gli id CELEX di tutte le direttive emanate in un certo anno e

restituisce un vettore contenente tali id;

clxDownloader - procede con le richieste al prototipo di portale per scaricare ogni direttiva, grazie all'ausilio della funzione async.mapLimit; la funzione iteratrice prende il nome di **clxIterator**.

Listing 5.4: La funzione scrapeEurLex nel file *webscr.js*

```
function scrapeEurLex(year, pag, pages, array, callback){
  var totpages = pages;
  var url='http://eur-lex.europa.eu/search.html?DTA='+year+'&
   DB_TYPE_OF_ACT=directive&CASE_LAW_SUMMARY=false&DTS_DOM=
   ALL&excConsLeg=true&typeOfActStatus=DIRECTIVE&type=
   advanced&SUBDOM_INIT=ALL_ALL&DTS_SUBDOM=ALL_ALL&page='+pag
   ;
  request(url, function(err, res, html) {
    if (err) { //ERROR FROM EUR-LEX
    } else {
      var $ = cheerio.load(html);
      $('.documentTable').filter(function(){
        $('.leftMetadata').each(function(i , elem){
          var iddoc = $(this).find('.directTextAccess').next
   ().text().split(': ')[1];
          if (!iddoc || iddoc.length > 10) { //INVALID CELEX
          } else { array.push(iddoc); }
        });
      });
      if (pag == 1) { //NUMBER OF PAGE-RESULTS
        var ress = $('.resultNumber').first().text();
        totres=parseInt(ress.toString().split('of')[1]);
        totpages = Math.ceil(totres/10);
      }
      if(pag==totpages||totpages==1){callback(null, array);}
      else{scrapeEurLex(year,pag+1,totpages,array,callback);
      }
    }
  });
};
```

5.5.3 La visualizzazione side-by-side

Quest'ultima funzionalità consiste nella visualizzazione affiancata delle istanze FORMEX ed AKOMA NTOSO dello stesso documento legale, permettendo di mettere in risalto gli elementi che hanno lo stesso significato fra i due diversi dialetti.

L'elemento principale di questa funzionalità si chiama *CodeMirror*, ed è un modulo lato client che permette di visualizzare un editor di testo nella finestra del browser ricco di funzionalità, che sono: visualizzazione del numero di riga con una piccola freccetta grazie alla quale è possibile effettuare il *code folding*[2], highlight colorato del codice XML (i commenti in grigio, i tag in verde, gli attributi in blu e i valori degli attributi in rosso), possibilità di evidenziare l'apertura e la chiusura di uno specifico tag cliccando su di esso.

La soluzione che ho ideato per mettere in risalto la mappatura dei file, ovvero la corrispondenza fra gli elementi che hanno lo stesso significato nei due dialetti, consiste nell'aggiungere un nuovo attributo ad ogni tag che deve essere mappato, in entrambi i file: il valore dell'attributo sarà un id che metterà in relazione i due tag interessati, e tale id verrà nascosto nella visualizzazione su browser.

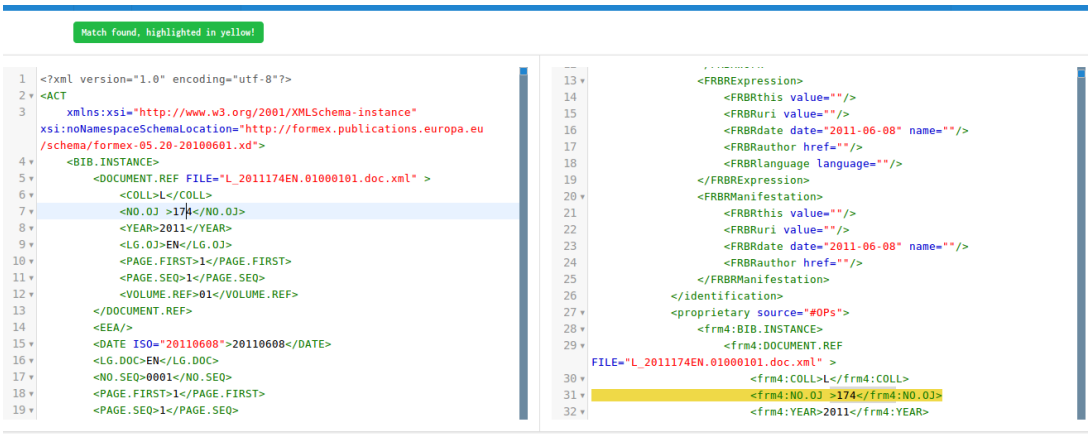

Figure 5.5: La visualizzazione side-by-side (prototipo).

[2]Con la dicitura "code folding" si intende il raggruppamento, in un'unica riga, di un tag che si estende in più righe.

Per poter supportare la funzione visiva di corrispondenza della mappatura è stato necessario modificare alcuni file sorgente del modulo *CodeMirror*, in particolare quelli che si occupano dell'addon di ricerca: nel momento in cui l'utente clicca su un tag che è presente nella mappatura, l'istanza di CodeMirror non fa altro che recuperare il valore dell'attributo destinato alla mappatura e ricercarlo nell'altra istanza di CodeMirror.

Questa funzionalità è in realtà solo un prototipo, in quanto nel portale non è presente un modulo che permette la conversione da FORMEX ad AKOMA NTOSO; la visualizzazione è infatti disponibile solo per un documento, di cui mi è stata fornita, oltre l'istanza FORMEX, la corrispondente traduzione in AKOMA NTOSO; il principio funziona e può essere utilizzato per sviluppi futuri di tale funzionalità.

5.6 Conclusione del progetto

Questo capitolo illustra tutto ciò che riguarda l'implementazione del prototipo di portale **Fmx2Akn** ed è arricchito dall'inclusione di alcuni frammenti di codice, da alcuni screenshot e dalla spiegazione di numerose funzioni create nel portale. Nella prima sezione ho esposto in generale l'architettura del portale, descrivendo la suddivisione in cartelle e la funzione a cui è adibito ogni singolo file; la seconda sezione è dedicata all'interfaccia, ed analizza tutti gli elementi esclusivi di Semantic-Ui che ho utilizzato, le view, il layout e gli helper che ho creato; nella terza sezione si scende nel dettaglio del file *eurlex.js*, libreria da me creata che contiene tutte le funzioni necessarie per interagire con i contenuti di CELLAR, il database di EUR-Lex; la quarta sezione è dedicata all'utilizzo del portale, e descrive approfonditamente cosa succede al momento del caricamento della home page, al momento della digitazione nel campo di ricerca, all'avvio della ricerca e al momento della visualizzazione finale del risultato; la quinta sezione, infine, analizza le funzionalità del portale che non sono accedibili tramite interfaccia.

Capitolo 6

Conclusioni

Questa dissertazione ha trattato come argomento centrale la distribuzione dei documenti legali, e si conclude con la realizzazione di un prototipo di portale adibito a tale scopo. Questo prototipo, chiamato **Fmx2Akn**, si posiziona in un contesto di promozione dello standard AKOMA NTOSO, di cui ho ampiamente parlato: il dialetto XML AKOMA NTOSO è in grado di poter descrivere un'ampia gamma di documenti legali, grazie alla sua architettura basata su modelli astratti, e si può adattare alle differenti necessità di specifici sistemi legislativi; è opportuno promuoverlo non solo fra gli stati africani (è nato con lo scopo di aiutare i parlamenti africani ad assolvere al meglio le loro funzioni democratiche tramite l'utilizzo di tecnologie aperte), in quanto la diffusione di uno standard comune rappresenta il più importante strumento di semplificazione nello scambio e nel riferimento di documenti fra diversi parlamenti e organi della stessa nazione.

La realizzazione del prototipo di portale non è stata semplice ed ha richiesto diverse settimane di lavoro intenso: le parti che hanno richiesto più tempo e in cui ho riscontrato maggiori difficoltà sono state:

- la piena comprensione del *Tree notice*, il documento XML contenuto in CELLAR che contiene tutte le informazioni relative ad uno specifico documento, per trovare esattamente i campi giusti ed universali per poter estrapolare i metadati corretti (il Tree notice di un documento,

formattato in modo leggibile da un essere umano, può consistere anche in un file di oltre 30000 righe);

- l'utilizzo della tecnologia *Node.js*, in quanto non è estremamente intuitivo programmare un web server seguendo il modello asincrono di Javascript (è stata la prima volta che ho utilizzato questa tecnologia per tale scopo); per ben due volte, durante la fase di implementazione, ho deciso di riscrivere il codice seguendo delle *best-practice* che ho imparato poco a poco con l'esperienza;

- la realizzazione del prototipo di interfaccia *side-by-side* che affianca le due istanze FORMEX ed AKOMA NTOSO dello stesso documento per evidenziare la mappatura dei tag, in quanto è stato necessario effettuare numerose modifiche al codice sorgente di *CodeMirror*, un modulo lato client che permette la visualizzazione di un editor di testo avanzato nel browser.

L'idea per questo prototipo è di realizzare un sistema che sia in grado di reperire automaticamente i documenti legali rappresentati col dialetto FORMEX, convertirli nel dialetto AKOMA NTOSO e restituire all'utente il risultato di tale conversione, tenendo in considerazione una serie di lacune di usabilità che ho riscontrato nell'utilizzo di EUR-Lex che sono: impossibilità di reperire un documento nel formato tecnico FORMEX, impossibilità di recuperare i file relativi ad un certo documento tramite interrogazione ad una API REST, e impossibilità di poter adattare la schermata del sito alle dimensioni dello schermo dell'utente.

Il prototipo che ho realizzato è in grado di sopperire alle lacune appena citate, e riesce ad effettuare due delle tre task importanti: recupero automatico dei file FORMEX e visualizzazione del risultato della conversione.

Gli sviluppi futuri di questo progetto di tesi comprendono l'implementazione del processo di conversione dei documenti dal dialetto FORMEX ad AKOMA NTOSO; per quanto riguarda la visualizzazione di tale conversione, la mia idea per evidenziare la mappatura degli elementi si concentra sull'ag-

giunta di un nuovo attributo, affiancato ad ogni tag mappato, che riesce a collegare i tag equivalenti grazie all'utilizzo di un identificatore univoco; nulla toglie che possa esistere un approccio migliore e più veloce per la risoluzione di questo problema.

Ulteriori miglioramenti al prototipo di portale potrebbero essere i seguenti:

- gestione di un numero più elevato di metadati per ciascun documento, per includere, ad esempio, i descrittori *EuroVoc* (EuroVoc è il thesaurus multilingue che comprende la terminologia dei settori di attività dell'UE);
- implementazione di un'interfaccia multilingua;
- possibilità di poter manovrare non solo le direttive europee ma qualsiasi tipo di documento legale prodotto dall'UE.
- possibilità di ampliare lo strumento di ricerca, utilizzando non solo l'identificativo CELEX ma anche altri metadati.

Bibliografia

[Ant11] Lucia Antonelli. Normattiva: sviluppo ed evoluzione. *Biblioteche oggi: Mensile di informazione aggiornamento dibattito*, 29(10):53–54, 2011.

[BELP13] Ulrik Brandes, Markus Eiglsperger, Jürgen Lerner, and Christian Pich. Graph markup language (graphml). *Handbook of graph drawing and visualization*, 20007:517–541, 2013.

[BPSM+08] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible markup language (xml) 1.0, 2008.

[Bro14] Ethan Brown. *Web Development with Node and Express: Leveraging the JavaScript Stack*. " O'Reilly Media, Inc.", 2014.

[Cer13] Luca Cervone. *Parametric editors for structured documents*. PhD thesis, 2013.

[DGPF12] DigitPA., Maria Pia Giovannini, Monica Palmirani, and Enrico Francesconi. *Linee guida per la marcatura dei documenti normativi secondo gli standard Normeinrete*. European Press Academic Publishing, 2012.

[Eis06] J David Eisenberg. *OASIS OpenDocument Essentials*. LuLu. com, 2006.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[G+05] Jesse James Garrett et al. Ajax: A new approach to web applications. 2005.

[Jai15] Nilesh Jain. Review of different responsive css front-end frameworks. *Journal of Global Research in Computer Science*, 5(11):5–10, 2015.

[LB03] Caterina Lupo and Carlo Batini. A federative approach to laws access by citizens: The "normeinrete" system. In *International Conference on Electronic Government*, pages 413–416. Springer, 2003.

[LMT14] Kai Lei, Yining Ma, and Zhi Tan. Performance comparison and evaluation of web development technologies in php, python, and node. js. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 661–668. IEEE, 2014.

[LWZZ08] Zhijie Lin, Jiyi Wu, Qifei Zhang, and Hong Zhou. Research on web applications using ajax new technologies. In *MultiMedia and Information Technology, 2008. MMIT'08. International Conference on*, pages 139–142. IEEE, 2008.

[Mar92] Nenad Marovac. Document recognition: concepts and implementations. *ACM SIGOIS Bulletin*, 13(3):28–38, 1992.

[Mat99] Luuk Matthijssen. Interfacing between lawyers and computers. *An Architecture for Knowledge based Interfaces to Legal Databases*, 1999.

[MMSV02] Andrea Marchetti, Fabrizio Megale, Enrico Seta, and Fabio Vitali. Using xml as a means to access legislative documents: Italian and foreign experiences. *ACM SIGAPP Applied Computing Review*, 10(1):54–62, 2002.

[Mur07] San Murugesan. Understanding web 2.0. *IT professional*, 9(4), 2007.

[Nat13] Kailashkumar V Natda. Responsive web design. *Eduvantage*, 1(1), 2013.

[otEU06] Publications Office of the European Union. *25 years of European law online (deluxe edition)*. 2006.

[PB02] Monica Palmirani and Raffaella Brighi. Norma-system: A legal document system for managing consolidated acts. In *International Conference on Database and Expert Systems Applications*, pages 310–320. Springer, 2002.

[PCR09] Monica Palmirani, Giuseppe Contissa, and Rossella Rubino. Fill the gap in the legal knowledge modelling. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 305–314. Springer, 2009.

[PPV13] Zachary Parker, Scott Poe, and Susan V Vrbsky. Comparing nosql mongodb to an sql db. In *Proceedings of the 51st ACM Southeast Conference*, page 5. ACM, 2013.

[PV12] Monica Palmirani and Fabio Vitali. Legislative xml: Principles and technical tools. Technical report, Inter-American Development Bank, 2012.

[RAAR13] Leonard Richardson, Michael Amundsen, Mike Amundsen, and Sam Ruby. *RESTful Web APIs*. " O'Reilly Media, Inc.", 2013.

[Ven10] Fernando Venturini. La banca dati normattiva. *Le Carte e la Storia*, 16(1):37–39, 2010.

[VZ07] Fabio Vitali and Flavio Zeni. Towards a country-independent data format: the akoma ntoso experience. In *Proceedings of the V legislative XML workshop*, pages 67–86. Florence, Italy: European Press Academic Publishing, 2007.

[Zan15] Michael Zander. *The law-making process*. Bloomsbury Publishing, 2015.

Elenco delle figure

Elenco del codice

Elenco delle tabelle

Ringraziamenti

Ed eccomi qui, a scrivere i ringraziamenti della tesi, sto ancora realizzando il fatto di aver raggiunto questo grande traguardo! E' stato un percorso molto lungo, complicato e ricco di esperienze che mi hanno fatto crescere. Non posso lontanamente immaginare il numero di persone che ho conosciuto qui a Bologna, e che, nel bene o nel male, mi hanno lasciato qualcosa!

Vorrei innanzitutto ringraziare la mia famiglia. Papà, mamma e mia sorella hanno sempre creduto in me. Mi hanno sempre sostenuto e non mi hanno mai fatto sentir solo, nonostante i 600km che ci separano non siano pochi, e di conseguenza le occasioni per vederci non siano tante. La distanza ha giocato un ruolo fondamentale nel comprendere quanto mi abbiano voluto bene in tutto questo tempo, pur non essendo sempre stati ricambiati, e senza di loro non sarei riuscito ad arrivare fin qui!

E' vero, ho un bel caratterino, e quindi mi sento di ringraziare in particolare tutti i coinquilini che in questi anni mi hanno sopportato e con i quali ho stretto un bel rapporto di amicizia! Dalla casa in via Schiassi, per poi Montebello e adesso in Mura Galliera, ho condiviso momenti di quotidianeità, gioie e dolori con davvero tante persone! Siete in troppi, vi ringrazio tutti, ma in particolare voglio salutare Paolo, Alessandro, Barbara, Vincenzo, Domenico, Enrico, Francesca, Ylenia, che mi hanno sempre dimostrato il loro affetto e la loro fiducia, condividendo tutte le gioie ed aiutandomi ad affrontare tutti i problemi. (e Carlo Alberto???). Poi c'è il grande Omar, considerato coinquilino acquisito, sempre presente fin dal primo anno di università, che si laurea il mio stesso giorno: sono troppo felice anche per te! Da non dimenti-

care anche la squadra in via Musolesi (Gabriele, Davide e Michele, mi sono sempre sentito a casa da voi!), e ancora altri colleghi come Fulvio, Alessio, Francesco, Milos, Pietro e Claudio, eccetera.

Un grazie un po' a tutti quelli che mi hanno voluto bene e mi hanno fatto passare dei bei momenti, non riuscirei nemmeno a ricordarmi di tutti voi per quanti ne siete! In particolare un abbraccio a Valentina, le "Giulie", Carla, Bianca, le vicine di mura galliera (Erica mi mancate!), le mie care amichette siciliane (Viola ti rivoglio a Bologna!), Andrea, Sofia e così via. Un saluto anche ai ragazzi di Bandolero Movement e Plasma FX, in particolare un abbraccio ad Andrea e Roberto.

Un abbraccio speciale va ai miei amici a Foggia, che sono sempre pronti ad accogliermi calorosamente nonostante ci vediamo e sentiamo poco, in particolare Piero, Alessandro e Ausilia (e il piccolo Gabriele), Valentina, Paolo, Agostino, Gabriele, Roberta, Giuseppe, Raffaella, eccetera.

Per terminare, vorrei innanzitutto ringraziare Luca per avermi guidato nella realizzazione del progetto di tesi e nella redazione di questa dissertazione, e poi i docenti che mi hanno seguito, dalle elementari fino all'università, e in particolare ai prof. Vittorio Ghini e Renzo Davoli che mi hanno dato mille spunti a livello informatico e mi hanno introdotto al mondo dell'open source, e infine alle prof. Luisa Cavaliere, Teresa Pellicano e Maria Teresa Caroppi, che mi hanno dato molto di più del semplice insegnamento e mi hanno voluto bene.

Non mi basterebbero altre dieci pagine per potervi ringraziare tutti! Grazie a tutti voi che siete passati nella mia vita ed avete lasciato un segno!