Alma Mater Studiorum - Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in
Informatica per il Management

Analisi di riferimenti bibliografici in documenti XML

Tesi di Laurea in
Ingegneria del Software

Relatore:
Chiar.mo Prof. Paolo Ciancarini
Correlatore:
Dr. Francesco Poggi

Presentata da:
Ariel Demian

Sessione III
Anno Accademico 2012/2013

Indice

Introduzione

Con il seguente elaborato propongo di presentare il lavoro svolto sui documenti XML che ci sono stati forniti. Più nello specifico, il lavoro è incentrato sui riferimenti bibliografici presenti in ogni documento e ha come fine l'elaborazione delle informazioni estrapolate al fine di poterle esportare nel formato RDF (Resource Description Framework). I documenti XML (eXtensible Markup Language) fornitimi provengono dalla casa editrice Elsevier[1], una delle più grandi case editrici di articoli scientifici organizzati in riviste specializzate (journal).

Il formato RDF è una parte importante del più ampio intento di rendere il World Wide Web più facilmente accessibile sia alle macchine, sia agli utenti, denominato Semantic Web. Il primo capitolo sarà un'introduzione al Semantic Web e alla necessità di utilizzare questo formato per esportare i dati ottenuti dall'elaborazione dei riferimenti bibliografici.

Il secondo capitolo, dopo una breve introduzione al formato XML e al linguaggio di definizione dei documenti XML, il DTD (Document Type Definition), verranno presentate e spiegate le difficoltà incontrate nel comprendere e utilizzare i documenti DTD forniti dalla stessa Elsevier, oltre ai documenti XML.

Nel terzo capitolo saranno introdotti tutti gli strumenti utilizzati a supporto dello sviluppo. Saranno spiegate le parti più importanti e i meccanismi di grande importanza per il raggiungimento dell'obiettivo preposto del lavoro. Oltre all'ambiente di sviluppo Java, sono stati utilizzati diversi strumenti durante le varie fasi dello sviluppo per meglio comprendere e individuare i pattern presenti nei documenti e successivamente elaborarli.

Tali strumenti sono:

- **PowerGREP**, uno strumento software molto potente che utilizza le espressioni regolari per analizzare i documenti forniti in input. Reputo tale strumento "potente" in quanto è ottimizzato per il compito svolto ed è assolutamente necessario nell'ambito dell'analisi di un certo numero di documenti, dell'ordine di migliaia o centinaia di migliaia
- **Jena**, una libreria Java per la creazione, l'elaborazione e l'esportazione automatica in diversi formati di statement RDF ma che ha le capacità per adempiere molti altri compiti legati al Semantic Web
- **Liquid XML Studio**, una suite completa per la manipolazione di documenti XML

Il sistema software prodotto sarà presentato nel quarto capitolo con tutte le considerazioni fatte durante lo sviluppo.

Nell'ultimo capitolo, le conclusioni riguarderanno i risultati raggiunti con il software prodotto e una considerazione sulle tecniche apprese personalmente inerenti alle attività di data mining.

[1] http://www.elsevier.com/

Capitolo 1

Semantic Web

1.1 Le origini del *World Wide Web*

Internet nasce nel 1989 con l'idea di un giovane ricercatore presso il CERN (l'Organizzazione Europea per la ricerca nucleare) per creare un sistema per la gestione delle informazioni relative agli esperimenti che si svolgevano presso il centro. Questo giovane ricercatore, Tim Berners-Lee, insieme a Robert Cailliau hanno realizzato gli elementi costitutivi che avrebbero determinato la nascita e lo sviluppo del *World Wide Web*. Tali elementi erano e sono tuttora sostanzialmente tre:

1. **URI** (*Uniform Resource Identifier*): uno schema univoco e globale per l'individuazione e indirizzamento di risorse mediante una stringa di caratteri

2. **HTTP** (*Hyper Text Transfer Protocol*): un protocollo di trasporto dati di tipo *client-server*, privo di memoria delle interazioni precedenti (*stateless*) e indipendente dal tipo di dati trasportati

3. **HTML** (*Hyper Text Markup Language*): un linguaggio di markup embedded (con la funzione di aggiungere informazioni e significato al testo presentato) basato sullo più generico *SGML*, con la capacità di aggiunta di collegamenti ipertestuali e in grado di fornire strumenti presentazionali (inserimento di immagini, tabelle, modifica dell'aspetto dei caratteri, ecc.); la prima versione di HTML era molto essenziale mentre oggi il linguaggio permette elaborazioni presentazionali molto più elaborate

Il protocollo HTTP, contrariamente alla sua denominazione (*Text Transfer*) che lascia intendere una specializzazione al trasferimento di ipertesti, è invece un protocollo di trasporto dati di qualunque tipo (risorse) e questo è anche uno dei motivi principali del suo successo negli anni. [3] Un esempio di URL è il seguente:

`http://www.example.com/my_page.html`

Inserendo questa stringa in un browser noi richiediamo al server

`www.example.com`

attraverso il protocollo `http://` la pagina `my_page.html`. La spiegazione del funzionamento non è ancora del tutto completa perché le macchine non comprendono il significato della stringa `www.example.com`. Questa stringa viene prima tradotta da un server DNS (*Domain Name System*) in un indirizzo IP e successivamente inoltra la richiesta al server che risponde a quel indirizzo IP, il quale ci restituisce la risorsa richiesta. Da questo breve riassunto del funzionamento possiamo capire come il funzionamento del *World Wide Web* si basi su una moltitudine di tecnologie, le quali interagiscono tra di loro per fornirci la risorsa richiesta.

Alle origini del *WWW*, le pagine erano del tutto statiche: dei semplici contenitori di testo, immagini, tabelle. Adempivano egregiamente al loro scopo originario: rappresentare le informazioni relative agli esperimenti scientifici. Infatti questa parte della storia di Internet è arrivata a chiamarsi *Web 1.0* o anche *Web statico*.

Con il proseguire degli anni, lo sviluppo delle tecnologie, la produzione di massa di PC e soprattutto con il successo di Internet, si sentiva il bisogno di arricchire le pagine con contenuti dinamici, contenuti capaci di rispondere alle richieste dell'utente e di eseguire compiti complessi come ad esempio: visualizzare video, gestire sessioni di utenti, caricare contenuti dinamicamente, abbellire le pagine con stili diversi, ecc. Sono state sviluppate diverse tecnologie per consentire questi compiti addizionali richiesti e che sarebbero stati impossibili da soddisfare soltanto dai tre semplici componenti iniziali del web.

Queste tecnologie addizionali sono, dalla parte del client:

- **CSS** (*Cascading Style Sheets*): linguaggio che permette di definire dei fogli di stile per la formattazione di documenti HTML e XML, ovvero definire l'aspetto con cui i contenuti di tali documenti sono mostrati all'utente
- **JavaScript**: linguaggio di programmazione che permette l'inserimento di codice all'interno di una pagina web per renderla dinamica e rispondere all'input dell'utente; questo linguaggio di programmazione è completo e permette un range molto ampio di comandi per manipolare la pagina web
- **XML** oppure **JSON**: il primo è un linguaggio di markup (*eXtensible Markup Language*) simile all'HTML ma più rigido nella sintassi e il secondo (*JavaScript Object Notation*) è un linguaggio di notazione per gli oggetti derivante da JavaScript; entrambi i linguaggi permettono di incapsulare dati in formato testuale e sono utili per il trasferimento e archiviazione di questi dati
- **RSS** (*RDF Site Summary*): è un formato popolare basato su XML per la distribuzione di contenuti, soprattutto per pubblicazione di news
- **jQuery** oppure **Ext JS**: framework JavaScript molto utili per valicare i diversi modi in cui ogni browser interpreta il codice Javascript e per creare pagine web ancora più interattive fornendo dei tool che sarebbero difficili da imitare con del semplice codice JavaScript
- **Ajax** (*Asynchronous JavaScript and XML*): un insieme di tecniche di programmazione delle pagine web per creare applicazioni web asincrone; la parte più importante di questa tecnica è l'oggetto XMLHttpRequest che in JavaScript permette di inviare richieste HTTP ai server e ricevere le risposte per visualizzare nuovi contenuti all'utente dinamicamente
- **XSLT e XPath**: il primo è un linguaggio di programmazione dichiarativo per la trasformazione di documenti XML, il secondo è un linguaggio di interrogazione di documenti XML maggiormente utilizzato all'interno di XSLT

Le tecnologie dalla parte server sono i linguaggi quali PHP, Ruby, Perl, Python, Enterprise Java (J2EE), Microsoft.NET Framework e Node.js per permettere agli sviluppatori sia di fornire ai client risposte elaborate alle richieste statiche e dinamiche, sia di creare canali di comunicazione tra un server e un altro.

Oltre a queste nuove tecnologie si devono menzionare le nuove capacità di trasmissione delle reti in fatto di velocità e flessibilità: reti Wi-Fi, fibra ottica, reti 3G e 4G per il mobile e crescita esponenziale del pool di utenti Internet in pochi anni.

Questo insieme di tecnologie accoppiato alle tecniche di web development utilizzate in tutto il mondo hanno decretato il cosiddetto *Web 2.0* ovvero il *Web dinamico*, in contrasto con il Web statico di pochi

anni prima. Il punto fondamentale di questo nuovo insieme di tecnologie è stato di dare la possibilità a tutti gli utenti di diventare autori di contenuti, non soltanto semplici fruitori. Si è cominciato ad usare il termine Web 2.0 per indicare la nuova situazione nel 1999, con il termine coniato da Darcy DiNucci nel suo articolo *Fragmented Future*[1] ma che venne popularizzato soltanto nel 2004 da Tim O'Reilly[2].

Durante tutti questi anni di sviluppo, è stati il *W3C* (World Wide Web Consortium) a stabilire e a detenere gli standard del World Wide Web, fondato e attualmente amministrato da Tim Berners-Lee.

1.2 Che cosa è il Web semantico?

Il passo successivo dovrà essere il Web 3.0, detto anche Web Semantico, che sfrutta tutte le tecnologie appena menzionate con l'aggiunta di altre per creare un World Wide Web in cui le risorse non sono semplici risorse di dati ma acquistano un significato che potrà essere facilmente consultato dagli utenti e sfruttato dalle macchine per fornire servizi che con le tecniche e le tecnologie precedenti sarebbero quasi impossibili da offrire. Le informazioni, che prima erano sparse su vari documenti e risorse sulla rete saranno disponibili in modo integrato e il Web Semantico decreterà il passaggio dal *web of documents* al *web of data*. Con questo sottocapitolo si vuole soltanto fornire una breve presentazione delle basi del Web Semantico e delle sue prospettive future.

È stato scelto di pubblicare le informazioni contenute nei riferimenti bibliografici dei documenti utilizzando le tecnologie del Web Semantico per mettere a disposizione in un formato aperto, ricco e riutilizzabile queste informazioni. Tutte le caratteristiche che rendono adatti gli strumenti del Web Semantico verranno spiegate in dettaglio nei prossimi sottocapitoli.

Quello del Web semantico è ancora un *work in progress* anche se numerosi aspetti sono ormai chiaramente delineati e resi standard. Bisogna però menzionare che vi è una leggera differenza di significato nei due termini in quanto Tim Berners-Lee ha descritto il Web Semantico come una componente del Web 3.0:

> «People keep asking what Web 3.0 is», Berners-Lee said. «I think maybe when you've got an overlay of scalable vector graphics - everything rippling and folding and looking misty - on Web 2.0 and access to a semantic Web integrated across a huge space of data, you'll have access to an unbelievable data resource.»[3]

Che cosa è il Web Semantico e quali sono le tecnologie che aiuteranno a dare significato alle risorse presenti sul Web?

Per rappresentare le componenti del Web Semantico si può facilmente utilizzare il modello di struttura a strati fornito dallo stesso W3C in figura 1.1.

Il modello ha come base gli URI/IRI (*Uniform Resource Identifier* e *Internationalized Resource Identifier*), di cui gli URL, menzionati all'inizio del capitolo, sono una specializzazione. La differenza più marcata fra URI e IRI è che questi ultimi permettono anche l'inserimento di caratteri non ASCII grazie alla codifica UTF-8[4]. Essi servono per identificare univocamente una risorsa. L'insieme degli URI è composto dagli URL e dagli URN, come indicato in figura 1.2. Gli URN (*Uniform Resource Name*) sono degli identificatori univoci, cioè degli URI, per una certa risorsa che utilizzano un namespace identifier diverso dagli URL. È molto più facile comprendere cosa siano da alcuni esempi di URN: i codici ISBN

[1] http://darcyd.com/fragmented_future.pdf
[2] http://oreilly.com/web2/archive/what-is-web-20.html
[3] http://www.nytimes.com/2006/05/23/technology/23iht-web.html
[4] http://www.w3.org/International/iri-edit/draft-duerst-iri-03.txt

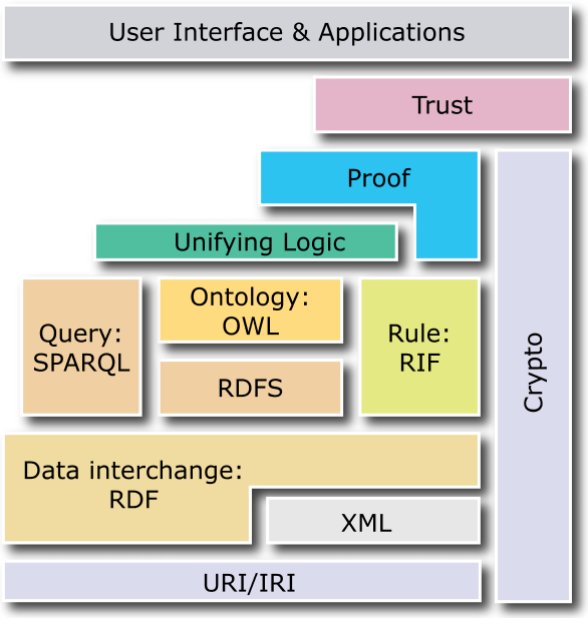

Figura 1.1: Modello di struttura a strati del Web Semantico `http://www.w3.org/2007/03/layerCake.png`

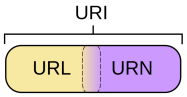

Figura 1.2: Fonte: `http://en.wikipedia.org/wiki/Uniform_resource_locator`

dei libri, i codici DOI (*Digital Object Identifier*)[5], i codici a barre ISSN dei prodotti: tutti questi sono degli URN perché identificano univocamente una risorsa all'interno del proprio namespace.

Gli URI sono la base per il Web Semantico, infatti vengono riportati in basso come primo gradino nella figura 1.1. Il gradino immediatamente superiore è l'XML, che abbiamo menzionato a pag. 2 e che verrà spiegato in dettaglio nel prossimo capitolo.

1.3 Resource Description Framework

Il tassello più importante per dare significato alle risorse sul Web e di interconnettere questi significati tra loro è l'RDF. L'elemento base di questo formato è lo statement. Un documento RDF è formato da numerose asserzioni (*statement*). Il singolo statement è formato da una tripla della forma:

Soggetto Predicato Oggetto . (da notare il punto alla fine, il quale è sempre necessario)

RDF è un modello astratto in cui le asserzioni vengono espresse attraverso delle triple, mentre per esprimere le triple abbiamo diverse notazioni o formati.

[5]Il gruppo responsabile del DOI ha deciso di non registrare il DOI come un namespace URN in quanto l'hanno reputata una cosa non necessaria e ridondante, nonostante il DOI soddisfi tutti i requisiti funzionali. La differenza sostanziale è avere invece della stringa 'urn:doi:10.1000/1' la stringa 'doi:10.1000/1'.

Figura 1.3: Soggetto Predicato Oggetto.

Un esempio molto semplice di statement RDF è presentato in figura 1.3.

Lo stesso statement in notazione N3 è il seguente:

```
<http://it.wikipedia.org/wiki/Paolo_Borsellino> <http://xmlns.com/foaf/0.1/familyName>
    "Borsellino" .
```

Nel nostro esempio, il soggetto è la risorsa che denota il magistrato Paolo Borsellino, il predicato è una risorsa che ci indica che l'oggetto sarà il nome e l'oggetto è una stringa. Semplicemente, il nostro statement ci comunica che la risorsa indicata dall'URI ha come nome di famiglia "Borsellino". In RDF è assolutamente necessario che il soggetto e il predicato siano degli URI mentre l'oggetto può essere sia un URI sia un letterale (stringa, numero intero, float, boolean, ecc.).

Ci sono diverse notazioni per l'RDF. In questo elaborato verranno utilizzate soltanto le prime due di queste notazioni, la prima per la sua facilità di utilizzo, la seconda per il suo esteso utilizzo da parte delle macchine:

- **Turtle** (*Terse RDF Triple Language*)
- **RDF/XML**
- **N3** (*Notation3*)
- **N-Triples**

Come si può evincere dal nome, la sintassi RDF/XML si basa sulla sintassi XML per codificare gli statement RDF. Di seguito troviamo lo stesso statement precedente con la sintassi RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ns="http://xmlns.com/foaf/0.1/">
        <rdf:Description rdf:about="http://it.wikipedia.org/wiki/Paolo_Borsellino">
                <ns:familyName>Borsellino</ns:familyName>
        </rdf:Description>
</rdf:RDF>
```

La notazione Turtle è invece più naturale da leggere per un essere umano. Lo statement precedente potrà essere codificato in Turtle nel modo seguente:

```
@prefix itwiki: <http://it.wikipedia.org/wiki/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

itwiki:Paolo_Borsellino foaf:familyName "Borsellino" .
```

Possiamo osservare l'introduzione di un nuovo componente `@prefix`. Questo componente permette di risparmiare molto spazio nella pagina, dichiarando una parte dell'URI di una risorsa come un prefisso e incrementando la leggibilità del testo.

In RDF ci sono le *classi* (definite grazie a RDF-S, presentato più avanti) e le *proprietà* per permettere di descrivere la natura di una risorsa. Negli statement ordinari, le classi possono essere soggetto oppure oggetto ma non possono svolgere il ruolo di predicato. Le proprietà sono generalmente utilizzate come predicati. Per brevità qui menzioniamo soltanto la proprietà più largamente utilizzata: `rdf:type`, che nella notazione Turtle viene brevemente indicata con `a`. Un esempio di utilizzo di questa proprietà è il seguente:

```
@prefix itwiki: <http://it.wikipedia.org/wiki/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

itwiki:Paolo_Borsellino rdf:type foaf:Person .
itwiki:Paolo_Borsellino a foaf:Person .
```

I due statement precedenti sono del tutto equivalenti, quindi nessuno dei due aggiunge alcuna informazione in più rispetto all'altro. Entrambi affermano che la risorsa `itwiki:Paolo_Borsellino` è una `foaf:Person`..

Questi semplici statement sono la base del Web Semantico ma in RDF manca un meccanismo generale per generalizzare affermazioni come "Paolo Borsellino è un magistrato." e "Ogni magistrato è una persona.", in modo da permettere a RDF di dedurre automaticamente che "Paolo Borsellino è una persona." Per superare questa e altre limitazioni in RDF per quanto concerne la generalizzazione degli statement è stato introdotto **RDF-S** (*RDF Schema*).

RDF fornisce già alcuni termini predefiniti per classificare e descrivere i suoi elementi:

- rdf:type – il primo e il più importante termine RDF che permette di inserire come oggetto la tipologia della risorsa indicata come soggetto
- rdf:Property – è la classe dei predicati RDF
- rdf:Statement – è la classe degli statement
- rdf:subject – è un'istanza di rdf:Property usata per descrivere il soggetto di uno statement
- rdf:predicate – è un'istanza di rdf:Property usata per descrivere il predicato di uno statement
- rdf:object – è un'istanza di rdf:Property usata per descrivere l'oggetto di uno statement

Altri termini predefiniti sono stati introdotti in RDF per descrivere contenitori di oggetti come ad esempio le liste.

RDF-S[6] è un'estensione di RDF che fornisce un insieme di strumenti per poter descrivere meglio le risorse utilizzate. Per brevità qui enumeriamo soltanto le risorse principali di RDF-S.

Classi RDF-S (`@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .`):

- rdfs:Resource – tutte le cose descritte in RDF sono risorse
- rdfs:Class – la risorsa che descrive una classe
- rdfs:Literal – la classe dei literali come ad esempio le stringhe e i numeri interi
- rdfs:Datatype – la classe delle tipologie di dato

Proprietà RDF-S:

- rdfs:range – indica quali sono le classi degli oggetti ammessi come oggetto per una data proprietà (codominio)
- rdfs:domain – indica quali sono le classi degli oggetti ammessi come soggetto per una data proprietà (dominio)
- rdfs:subClassOf – indica che una classe è sottoclasse (estende) di un'altra
- rdfs:subPropertyOf – indica che una proprietà è sottoproprietà (estende) di un'altra
- rdfs:label – l'oggetto di questa proprietà indica versione leggibile della risorsa soggetto
- rdfs:comment – indica un commento per la risorsa soggetto

[6] http://www.w3.org/TR/rdf-schema/

1.4 Ontologie e dataset per il Web Semantico

Nella vita di tutti i giorni, tra di noi riusciamo a comunicare a voce, per iscritto o a segni, perché utilizziamo un linguaggio comune. Nel Web Semantico vi è la necessità di avere delle classi e delle proprietà comuni per poter comunicare e per poter codificare il mondo esterno e portarlo nel mondo del Web. Questa cosa viene resa possibile grazie alle ontologie.

Che cosa è un'ontologia?

Nel dizionario della lingua italiana Hoepli, alla voce "ontologia" leggiamo: "Parte della filosofia che ha come oggetto di studio i caratteri universali dell'essere in quanto tale, a prescindere dalle sue qualità particolari o fenomeniche"[7]. Nel Web Semantico, questa parola assume un significato molto diverso e denota una "concettualizzazione di un dominio di interesse"[8]. Precedentemente abbiamo utilizzato un certo URI a pagina 5: `foaf:familyName`. Il FOAF è una delle tante ontologie, detti anche vocabolari del Web Semantico.

Grazie alle ontologie, si viene a creare una netta distinzione fra due tipologie di statement:

- textbfTbox sono componenti terminologici (*terminological component*) che descrivono una concettualizzazione attraverso statement
- textbfAbox sono componenti assertivi (*assertion component*) che descrivono oggetti utilizzando le classi e le proprietà di una concettualizzazione descritta attraverso Tbox, sempre attraverso statement

Un'ontologia può essere considerata come un insieme di statement Tbox per concettualizzare un dominio di interesse, mentre tutti gli statement che utilizzano quell'ontologia sono degli Abox.

Le due ontologie che brevemente presenterò in questo elaborato a titolo di esempio sono FOAF e Dublin Core. Il progetto FOAF (*Friend Of A Friend*) ha come scopo la definizione di un'ontologia per esprimere i metadati relativi a persone, ai loro interessi, relazioni e attività. Il progetto Dublin Core invece cerca di definire un vocabolario comune per la descrizione semantica di tutte le risorse presenti sul Web. Il DC non potrà essere in grado di contenere o mantenere aggiornate tutte le descrizioni delle risorse introdotte sul Web ma il principio su cui si basa è quello di svolgere il ruolo di blocco base: descrizioni aggiuntive saranno prodotte combinando metadati con tale vocabolario per produrre descrizioni più ricche e complete.

Ritengo più semplice e immediato riportare un esempio di utilizzo del FOAF e del DC per mostrarne le potenzialità:

```
@prefix itwiki: <http://it.wikipedia.org/wiki/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

itwiki:Paolo_Borsellino a foaf:Person .
itwiki:Paolo_Borsellino foaf:gender "male" .
itwiki:Paolo_Borsellino foaf:givenName "Paolo" .
itwiki:Paolo_Borsellino foaf:familyName "Borsellino" .
itwiki:Paolo_Borsellino foaf:birthday "01-19" .
itwiki:Giovanni_Falcone a foaf:Person .
itwiki:Paolo_Borsellino foaf:knows itwiki:Giovanni_Falcone .
itwiki:Giovanni_Falcone foaf:knows itwiki:Paolo_Borsellino .
itwiki:Matrix a http://purl.org/dc/dcmitype/MovingImage .
itwiki:Matrix http://purl.org/dc/terms/title "Matrix"@it .
itwiki:Matrix http://purl.org/dc/terms/title "The Matrix"@en .
```

[7] http://www.grandidizionari.it/Dizionario_Italiano/parola/O/ontologia.aspx?query=ontologia
[8] http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

RDF permette anche di indicare la lingua nella quale è espresso il letterale stringa. In questo esempio abbiamo il titolo “Matrix” in italiano che è diverso dal titolo “The Matrix” in inglese della stessa risorsa.

La facilità con cui abbiamo scritto questi statement è evidente ma ci sono delle preoccupazioni a cui bisogna rispondere per quanto riguarda la loro attendibilità. Nei prossimi sottocapitoli vedremo come gestire gli statement, come pubblicarli e come analizzare la loro attendibilità.

1.4.1 OWL

Il linguaggio OWL *Web Ontology Language* verrà brevemente presentato in questo paragrafo. Esso fornisce più espressività rispetto a RDF e RDF-S per la definizione di ontologie. La versione presentata qui è la OWL 2.

Sono state definite cinque sintassi diverse per OWL 2:

- Turtle
- RDF/XML
- OWL/XML
- sintassi funzionale
- Manchester Syntax

In pratica un’ontologia viene definita utilizzando tutti gli strumenti a disposizione dello sviluppatore quindi OWL, RDF e RDF-S vengono utilizzati congiuntamente per la definizione di classi, proprietà e regole per d’impiego per entrambe.

Le ontologie utilizzate per lo sviluppo del sistema software sono state scritte in OWL quindi una minima conoscenza di questo linguaggio è risultata utile a comprendere ad es. il ruolo svolto da una certa classe e quindi come utilizzarla correttamente. Di seguito abbiamo a titolo di esempio la dichiarazione della classe `Expression` nell’ontologia FRBR (in formato RDF/XML) che verrà presentata più avanti.

```
<rdf:Description rdf:about="http://purl.org/vocab/frbr/core#Expression">
  <owl:disjointWith rdf:resource="http://purl.org/vocab/frbr/core#Manifestation"/>
  <skos:changeNote rdf:nodeID="A12"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  <skos:changeNote rdf:nodeID="A13"/>
  <skos:changeNote rdf:nodeID="A14"/>
  <owl:disjointWith rdf:resource="http://purl.org/vocab/frbr/core#Work"/>
  <skos:definition xml:lang="en">A realization of a single work usually in a physical
      form.</skos:definition>
  <rdfs:comment xml:lang="en">This class corresponds to the FRBR group one entity
      'Expression'.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://purl.org/vocab/frbr/core#Endeavour"/>
  <owl:disjointWith rdf:resource="http://purl.org/vocab/frbr/core#Item"/>
  <rdfs:isDefinedBy rdf:resource="http://purl.org/vocab/frbr/core"/>
  <rdfs:label xml:lang="en">expression</rdfs:label>
  <dct:issued>2005-07-15</dct:issued>
</rdf:Description>
```

Come possiamo osservare, OWL sfrutta sia il linguaggio RDF-S sia i propri strumenti. In questo esempio abbiamo `owl:disjointWith` che sta ad indicare che la classe `Expression` è disgiunta dalla classe `Manifestation` e dalla classe `Work`. Questa dichiarazione ci dice chiaramente che un `Expression` non potrà essere né `Work` né `Manifestation`.

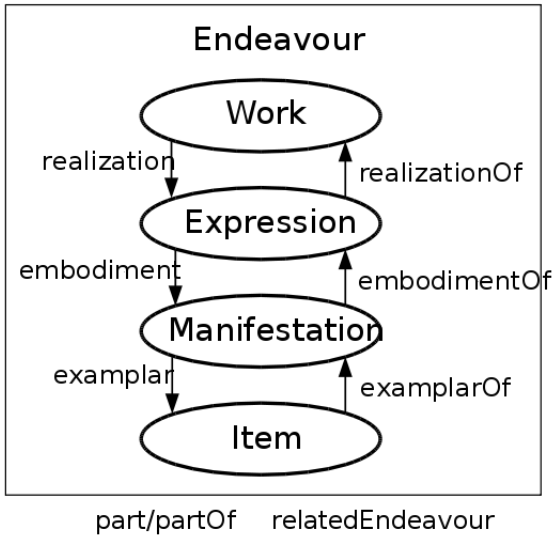

Figura 1.4: Diagramma FRBR che mostra i quattro stadi di un Endeavour `http://it.wikipedia.org/wiki/File:FRBR-Group-1-entities-and-basic-relations.svg`

Reasoner

Lo strumento che è capace di verificare l'attendibilità delle dichiarazioni di un'ontologia e degli statement che utilizzano classi e proprietà di quest'ultima viene denominato *Semantic Reasoner*. Un reasoner è un software capace di inferire conseguenze logiche sulla base di regole formalizzate, nel nostro caso un'ontologia, e di produrre nuove affermazioni derivanti dalle regole della logica formale. Questo strumento è capace di individuare errori di logica negli statement analizzati e anche nelle ontologie.

Un reasoner svolge essenzialmente due compiti:

1. verificare la correttezza degli statement in base alle ontologie utilizzate e agli statement a monte
2. produrre nuovi statement attraverso le regole della logica formale

È il reasoner che permette di derivare dalle affermazioni "Paolo Borsellino è un magistrato." e "Ogni magistrato è una persona." che "Paolo Borsellino è una persona.", dall'esempio esposto a pagina 6.

Le varie implementazioni di reasoner[9] non verranno descritte in questo elaborato ma si informa il lettore che questo ambito è ancora oggetto di studio e di sviluppo.

1.4.2 FRBR

FRBR (Functional Requirements for Bibliographic Records) è uno schema concettuale sviluppato da *International Federation of Library Associations and Institutions*[10] (IFLA) allo scopo di fornire uno strumento per la rappresentazione delle informazioni bibliografiche. La parte più importante e significativa per questo elaborato e per il sistema software prodotto riguarda i diversi livelli di una cosiddetta impresa intellettuale (*Endeavour*).

Nella figura 1.4 possiamo osservare come i quattro stadi di una impresa intellettuale siano distinti. Per spiegare brevemente questo concetto di divisione: un'opera (*Work*) ha come realizzazione la sua espressione (*Expression*). A titolo d'esempio l'opera "I Promessi Sposi" ha avuto durante gli anni diverse

[9] http://jena.apache.org/documentation/inference/
[10] http://www.ifla.org/

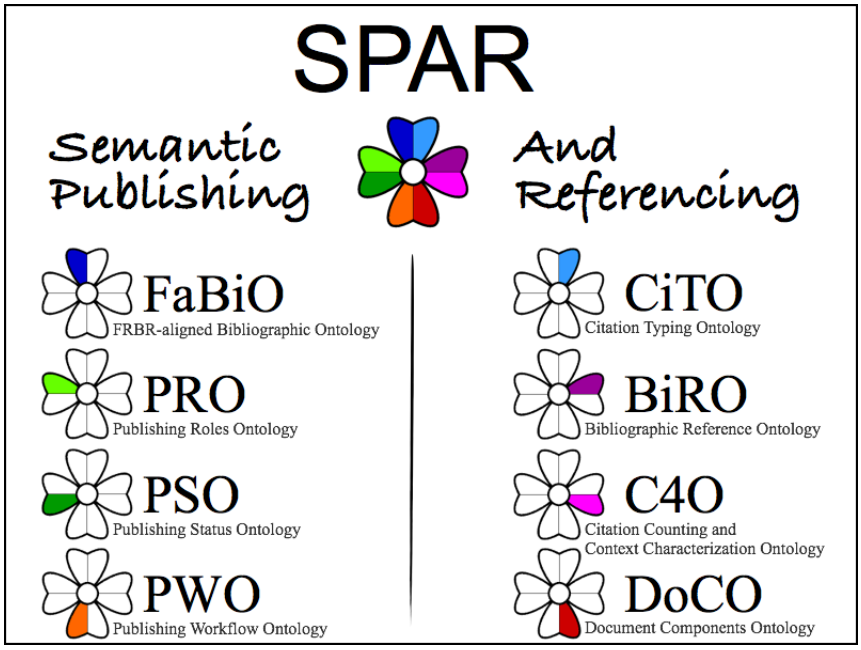

Figura 1.5: Diagramma a fiore che mostra il logo di SPAR `https://svn.code.sf.net/p/sempublishing/code/SPAR/SPAR-ontos-new.png`

revisioni che potranno essere assimilate a diverse espressioni secondo questo modello. Anche la traduzione in una lingua diversa rappresenta una diversa espressione della stessa opera. A sua volta, ogni espressione può avere diverse manifestazioni (*Manifestation*) che potranno essere le diverse edizioni di un libro che comprende l'opera. Una manifestazione viene stampata quindi si avranno diversi esemplari materiali (*Item*) dell'opera.

Ad ogni livello del modello FRBR della stessa opera si potranno associare diverse informazioni, come ad es. il titolo che verrà associato a livello d'espressione perché la stessa opera avrà diversi titoli in lingue diverse. Nel suo sviluppo, questo modello non è stato pensato per il Web Semantico ma ha assunto un ruolo cardine per il SPAR, una famiglia di ontologie che verranno presentate nel prossimo paragrafo.

È stata sviluppata anche un'ontologia che implementa i concetti descritti nel FRBR [2] e che viene utilizzata dal sistema software. Di seguito vengono elencate le classi principali di questa ontologia:

- Endeavour: `http://purl.org/vocab/frbr/core#Endeavour`
- Work: `http://purl.org/vocab/frbr/core#Work`
- Expression: `http://purl.org/vocab/frbr/core#Expression`
- Manifestation: `http://purl.org/vocab/frbr/core#Manifestation`
- Item: `http://purl.org/vocab/frbr/core#Item`

1.4.3 SPAR

SPAR[11] (*Semantic Publishing and Referencing Ontologies*) è una famiglia di ontologie per creare dati in RDF comprensibili dagli utenti umani e dalle macchine per la pubblicazione semantica[12]. Per pubblicazione semantica di un documento si intende la pubblicazione online di tutte le informazioni relative a quel documento utilizzando gli strumenti forniti dal Web Semantico per la codifica dei metadati.

[11] http://sempublishing.sourceforge.net/
[12] http://en.wikipedia.org/wiki/Semantic_publishing

Questa famiglia di ontologie è stata definita utilizzando OWL 2. Le informazioni pubblicate grazie a SPAR dipendono dai singoli componenti:

- FaBiO (*FRBR-aligned Bibliographic Ontology*): ontologia allineata con il FRBR per la descrizione di entità che sono state pubblicate o potranno esserlo
- PRO (*Publishing Roles Ontology*: ontologia che caratterizza i ruoli di agenti (persone e corporazioni) che vengono adempiuti nel processo di pubblicazione di qualunque tipo di materiale editoriale
- PSO (*Publishing Status Ontology*): ontologia per la descrizione lo stato di pubblicazione di un documento o di altre tipologie di pubblicazioni a diversi stadi del processo di pubblicazione
- PWO (*Publishing Workflow Ontology*): ontologia per la descrizione del flusso di lavoro associato con la pubblicazione di un documento o di altre tipologie di pubblicazioni
- CiTO (*Citation Typing Ontology*): ontologia che caratterizza la natura delle varie tipologie di citazioni che si possono trovare in un documento riguardo ad altri tipologie di pubblicazioni o documenti
- BiRO (*Bibliographic Reference Ontology*): ontologia allineata con il FRBR per la definizione di record e riferimenti bibliografici
- C4O (*Citation Counting and Context Characterization Ontology*): ontologia che permette la caratterizzazione di citazioni bibliografiche riguardante il loro numero e il loro contesto
- DoCO (*Document Components Ontology*): ontologia per la caratterizzazione delle varie parti, sia strutturali (ad es. blocco, linea, paragrafo, sezione, capitolo) che retoriche (ad es. introduzione, discussione, riconoscimenti, lista dei riferimenti, appendice) che compongono le varie tipologie di documenti

Alcune di queste ontologie sono state utilizzate per codificare le informazioni presenti nei riferimenti bibliografici dei documenti forniti, in particolare FaBiO, PRO e BiRO.

FaBiO

FaBiO [1] è un'ontologia per la descrizione di entità pubblicate o da pubblicare. Nell'ambito del sistema software è stata utilizzata per la descrizione delle opere a cui i riferimenti bibliografici si riferiscono. Questa ontologia è composta dalle dichiarazioni delle classi e delle proprietà. Le classi di questa ontologia sono numerose e comprendono le varie forme di entità pubblicabili, dai blog alle tesi e dagli articoli alle poesie.

Le classi e le proprietà utilizzate nel software prodotto sono:

```
@prefix fabio: <http://purl.org/spar/fabio/> .
```

- `fabio:JournalArticle` articolo solitamente di un giornale scientifico che viene raggruppato assieme ad altri articoli in una pubblicazione (*issue*)
- `fabio:JournalIssue` pubblicazione che raccoglie più articoli
- `fabio:JournalVolume` un volume di un periodico che raccoglie più pubblicazioni (*issue*)
- `fabio:Journal` un periodico che raccoglie contenuti su un dato argomento
- `fabio:PrintObject` classe che esprime un oggetto adatto ad essere stampato

- `fabio:Book` classe che esprime un libro
- `fabio:BookSeries` classe che esprime una serie di libri
- `fabio:WebContent` classe che esprime del contenuto pubblicato sul Web, a livello Expression del modello FRBR
- `fabio:WebManifestation` classe che esprime la manifestazione del contenuto pubblicato sul web secondo il modello FRBR
- `fabio:hasSequenceIdentifier` proprietà che descrive un identificatore di un oggetto all'interno di una sequenza di pubblicazioni

PRO

PRO [5] è un'ontologia per la descrizione dei vari ruoli nel processo di pubblicazione. La classe `pro:PublishingRole` contiene vari elementi che nell'ambito delle ontologie vengono chiamati *named individuals*. Nell'ambito del software prodotto gli elementi utilizzati principalmente sono: `pro:author`, `pro:contributor`, `pro:publisher`, `pro:series-editor` e `pro:editor`.

BiRO

BiRO [4] è un'ontologia creata con lo scopo di descrivere i riferimenti bibliografici secondo il modello FRBR.

Le classi e le proprietà utilizzate nel software prodotto sono:

```
@prefix biro: <http://purl.org/spar/biro/> .
```

- `biro:ReferenceList` lista utilizzata per descrivere la bibliografia di un articolo, di un libro o di una pubblicazione simile
- `biro:BibliographicReference` un singolo riferimento bibliografico a livello di Expression secondo il modello FRBR
- `biro:references` proprietà che associa un riferimento bibliografico ad un'impresa intellettuale

1.4.4 PRISM

PRISM[13] (*Publishing Requirements for Industry Standard Metadata*) è un insieme di strumenti che forniscono uno standard per la rappresentazione di metadati per la costruzione di multicanali efficienti per la pubblicazione editoriale. I metadati saranno indirizzati al materiale editoriale.

PRISM comprende un vocabolario RDF per la descrizione di questi metadati. Nell'ambito di questo lavoro ne sono state utilizzate alcune voci:

```
@prefix prism: <http://prismstandard.org/namespaces/basic/3.0/> .
```

- `prism:startingPage` proprietà che definisce la prima pagina della versione stampata della risorsa in questione
- `prism:endingPage` proprietà che ne definisce l'ultima pagina
- `prism:publicationDate` proprietà che identifica la data di pubblicazione per un contenuto digitale (data di post) e per un contenuto stampato (data di chiusura)

[13] http://www.idealliance.org/specifications/prism-metadata-initiative/prism

- `prism:subtitle` proprietà che identifica il sottotitolo della risorsa in questione
- `prism:issn` proprietà che riporta il codice ISSN della risorsa che dovrà identificare una pubblicazione in serie
- `prism:edition` proprietà che identifica l'edizione di una risorsa pubblicata
- `prism:isbn` proprietà che riporta il codice ISBN di una pubblicazione stampata, solitamente un libro
- `prism:url` proprietà che permette di inserire l'URL al quale il contenuto potrà essere raggiunto

1.5 SPARQL

SPARQL (*SPARQL Protocol and RDF Query Language*) è un linguaggio di query specificamente creato per interrogare i dataset RDF. La sintassi è incentrata sulle triple RDF con la dichiarazione di variabili utilizzando un nome preceduto dal punto interrogativo (`?attore`).

L'esempio riportato, estratto dal testo [3], ci offre chiaramente un punto di partenza per la compresione di questo linguaggio:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>

SELECT ?attore
WHERE{
  dbpedia:The_Matrix_Revolutions dbpedia-owl:starring ?attore .
}
```

La dichiarazione dei prefissi in SPARQL è simile a quella di RDF ma si differenzia per l'assenza del carattere `@` all'inizio e il punto alla fine della dichiarazione di ogni singolo prefisso. In questo esempio la query interroga il dataset di DBpedia per ottenere tutte le risorse che identificano un attore che ha avuto un ruolo nel film "The Matrix Revolutions".

Questo linguaggio permette la dichiarazione di più variabili con la possibilità di incrociare gli statement e raffinare i risultati.

L'esempio riporta l'utilizzo della query di tipo `SELECT`. SPARQL permette quattro tipologie di query:

- `SELECT` seguita da un elenco di variabili, questa tipologia di query interroga i dataset per tutte le risorse che adempiono alle condizioni indicate nella sezione `WHERE`
- `ASK` interroga i dataset se le risorse indicate esistano o meno e restituisce un valore boolean `true/false` sull'esito della query; di seguito abbiamo un breve esempio di query di tipo `ASK` che restituisce `true` se e soltanto se la risorsa indicata che dovrebbe identificare un film ha almeno un attore:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>

ASK{
  dbpedia:The_Matrix_Revolutions dbpedia-owl:starring ?attore .
}
```

- `CONSTRUCT` questa tipologia di query è composta da due sezioni: nella prima sezione indicata con `CONSTRUCT` vengono dichiarati statement con all'interno delle variabili e nella seconda sezione

indicata con `WHERE` vengono interrogati i dataset per dare valori alle variabili dichiarate precedentemente; il risultato di questa tipologia di query sono statement RDF della forma definita nella prima sezione del costrutto

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX facebook: <https://www.facebook.com/>

CONSTRUCT{
  facebook:demian.ariel foaf:knows ?attore .
}
WHERE{
  dbpedia:The_Matrix_Revolutions dbpedia-owl:starring ?attore .
}
```

Il risultato di questa query è il seguente:

```
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix dbpedia:        <http://dbpedia.org/resource/> .
<https://www.facebook.com/demian.ariel> foaf:knows      dbpedia:Carrie-Anne_Moss ,
  dbpedia:Harold_Perrineau ,
  dbpedia:Laurence_Fishburne ,
  dbpedia:Harry_Lennix ,
  dbpedia:Hugo_Weaving ,
  dbpedia:Jada_Pinkett_Smith ,
  dbpedia:Keanu_Reeves .
```

- `DESCRIBE` si richiede al server di restituire tutti ciò che si ha sulle variabili dichiarate che adempiono le condizioni indicate; il problema di questa tipologia di query è che attualmente ogni server può restituire i risultati in maniera diversa dagli altri, rendendo tale tipologia di query non interoperabile

Le query indicate possono essere provate immediatamente online dall'end-point SPARQL fornito apertamente da DBpedia[14]. DBpedia può essere brevemente descritto come la versione del Web Semantico di Wikipedia. Esso contiene gran parte dell'informazione di Wikipedia strutturata secondo le regole del Web Semantico.

1.6 Linked Data

Lo scopo principale del Web Semantico è strutturare la conoscenza e l'informazione umana condificata in bit in modo tale da consentirne l'interconnessione, una interconnessione facilmente usufruibile dagli umani e dalle macchine. Con il termine *Linked Data*[15] ci si riferisce all'intento di interconnettere l'informazione proveniente da fonti diverse sfruttando le tecnologie attuali con l'aggiunta degli strumenti offerti dal Web Semantico, risultando in una grossa rete eterogenea ma capace comunque di interoperare per fornire le informazioni richieste. Si parla anche di *Linked Open Data* per sottolineare l'apertura richiesta da questo intento: la possibilità offerta a chiunque di interrogare i dataset.

I quattro principi fondamentali del Linked Data sono:

1. utilizzare URI/URL per identificare le "cose" (per "cosa" si intende qualunque oggetto o concetto in qualunque ambito)

[14] http://dbpedia.org/sparql
[15] http://www.w3.org/DesignIssues/LinkedData.html

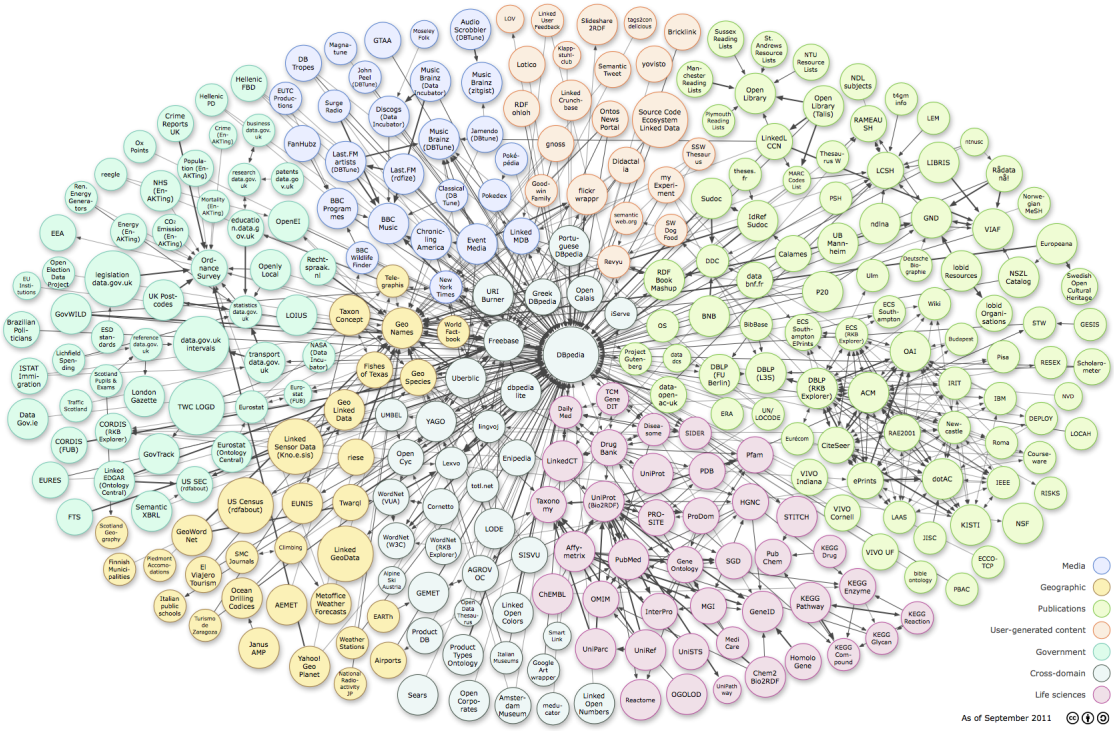

Figura 1.6: Linked Open Data a settembre 2011. Fonte: `http://en.wikipedia.org/wiki/File:LOD_Cloud_Diagram_as_of_September_2011.png`

2. è opportuno che questi URI utilizzino il protocollo HTTP per permettere una ricerca facile delle risorse a chiunque
3. quando viene cercata una risorsa attraverso un URI, restituire le informazioni utili sfruttando tecnologie standard (come RDF per strutturare le informazioni e SPARQL per permettere interrogazioni)
4. includere nelle descrizioni delle risorse link ad altri URI, in modo che possano essere scoperte altre risorse collegate

Dal 2009 è possibile ottenere un rating da 0 a 5 stelle per il proprio sito web in base al grado di conformità al progetto Linked Open Data. Per ottenere le varie stelle bisogna:

1. * fornire i propri dati sul web in formato aperto
2. ** fornire i dati in un formato strutturato e *machine readable*
3. *** fornire i dati in un formato non proprietario (ad es. CSV, JSON, XML)
4. **** utilizzo degli standard W3C quali RDF e SPARQL per identificare le risorse
5. ***** utilizzare RDF per interconnettere i propri dati con altri presenti sulla rete del Linked Data

Da notare che le risorse dovranno essere identificate da URI stabili nel tempo per consentire alle altre risorse di utilizzarli come riferimento e tutti gli URI dovranno contenere nomi significativi per gli utenti per non essere una semplice successione di caratteri alfanumerici apparentemente casuale. Queste considerazioni sono importanti per quanto riguarda il software prodotto. Nel quarto capitolo verranno esposte le soluzioni adottate.

Nella figura 1.6 possiamo osservare l'insieme dei dataset appartenenti al Linked Open Data a settembre 2011 con DBpedia al centro, anche se la rete in sé ha una natura decentralizzata.

1.7 Come pubblicare i dati sul Web Semantico?

La prima domanda da farsi è se abbiamo davvero bisogno di pubblicare i nostri dati secondo le regole specificate dal Web Semantico. Se il nostro intento è di rendere i nostri dati aperti e interoperabili, facili da comprendere dai singoli utenti e *machine readable* allora la risposta è affermativa. Bisogna innanzitutto avere un sito web che presenti in modo chiaro le informazioni di base riguardanti la propria persona o organizzazione (*front page*). È consigliabile che tutti gli URL definiti per le risorse siano raggiungibili utilizzando il protocollo HTTP e che non siano dei *broken link*, ossia degli indirizzi che non sono raggiungibili perché hanno una pagina corrispondente. A questo scopo è opportuno avere una infrastruttura software che crei dinamicamente ogni pagina per ogni risorsa. Nel capitolo relativo alla presentazione del software prodotto verrà descritto come è stata affrontata questa considerazione.

1.8 DOI

Il DOI[16] (*Digital Object Identifier*) è uno standard per la catalogazione e l'identificazione di qualunque entità assimilabile a proprietà intellettuale. Ogni singolo identificatore DOI è univoco e persistente ed è formato da due parti separate da uno slash (/):

- prefisso: identifica il nome del registrante della risorsa
- suffisso: nome scelto dal registrante per identificare una specifica risorsa associata al proprio prefisso

Ad es. il DOI
`10.1016/j.websem.2003.07.007`
(presente in uno dei documenti forniti) è formato dal prefisso `10.1016` che identifica l'organizzazione registrante della risorsa mentre `j.websem.2003.07.007` identifica la risorsa stessa.

Per risolvere un identificatore DOI e quindi arrivare alla risorsa specifica si può aprire la pagina `http://dx.doi.org/` e inserire il DOI oppure inserirlo direttamente nell'URL nel modo seguente, utilizzando come esempio il DOI sopra riportato:
`http://dx.doi.org/10.1016/j.websem.2003.07.007`
Se proviamo ad inserire questo indirizzo nella barra degli indirizzi di un browser, verremo reindirizzati alla pagina
`http://www.sciencedirect.com/science/article/pii/S1570826803000052`
che contiene l'articolo della casa editrice Elsevier, dove possiamo acquistarlo.

Il DOI può identificare non soltanto contenuto digitale ma anche oggetti fisici, lavori astratti, contratti, brevetti, ecc. Nell'ambito dello sviluppo del software prodotto, i codici DOI hanno svolto un ruolo cardine nell'identificazione delle risorse connesse ad ogni singolo documento. Nel quarto capitolo ne verrà descritto dettagliatamente l'utilizzo.

[16] http://www.doi.org/

Capitolo 2

XML, DTD e i documenti forniti

2.1 XML

Il formato XML (eXtensible Markup Language) è un linguaggio di markup che permette di aggiungere informazioni e dare significato alle parti di un documento. Questo linguaggio è formato da elementi denominati *tag* (etichette) o elementi. Di seguito abbiamo un semplice esempio di tag XML:

`<title>The Great Gatsby</title>`

Il tag `<title>` è il tag di apertura, il tag `</title>` è il tag di chiusura e la stringa "The Great Gatsby" è il contenuto testuale. Un tag senza contenuto testuale è un tag vuoto che può essere aperto e chiuso con un solo tag (ad es. `<title />`). Tutti i tag aperti devono anche essere chiusi.

La prima riga di un documento XML dovrebbe contenere il cosiddetto prologo XML:

`<?xml version="1.0" encoding="UTF-8"?>`

Esso ci indica che il documento in questione è un documento XML, la versione di XML è 1.0 e la codifica caratteri è UTF-8.

I tag possono contenere altri tag, creando una struttura ad albero. I tag interni sono detti tag annidati. I tag di apertura e di chiusura devono essere posti allo stesso livello dell'albero, quindi un tag non può essere chiuso prima che tutti gli altri tag posti all'interno di esso siano chiusi per non creare errori di annidamento. Un esempio di errore di annidamento è il seguente:

`<film><title>The Great Gatsby</film></title>`

Un documento XML si dice *Well Formed* (ben formato) se

- il primo elemento del documento XML è il prologo
- il documento XML contiene un solo tag all'interno del quale vengono posti tutti gli altri tag; questo elemento viene denominato elemento radice
- tutti i tag devono essere bilanciati, cioè non ci devono essere errori d'annidamento

Questo snippet di codice rappresenta un esempio di documento XML ben formato.

```
<?xml version="1.0" encoding="UTF-8"?>
<film>
    <title>The Great Gatsby</title>
</film>
```

Ogni elemento, detto anche nodo, può avere, oltre a un eventuale contenuto testuale o altri elementi, degli attributi. Gli attributi vengono inseriti all'interno del tag di apertura nel modo seguente:

`<title language="english" country="USA">The Great Gatsby</title>`

Un elemento può avere più di un attributo e il valore di ogni attributo deve essere posto all'interno dei doppi apici.

È importante notare che due o più elementi con lo stesso nome possono comparire allo stesso livello dell'albero del documento XML. Questo significa che possiamo avere l'elemento `<film>` che contiene due o più elementi `<title>`:

```
<?xml version="1.0" encoding="UTF-8"?>
<film>
    <title>The Great Gatsby</title>
    <title>The Matrix</title>
    <title>Titanic</title>
</film>
```

Per indicare un elemento all'interno di un documento XML possiamo sfruttare la sua struttura ad albero, quindi per identificare un certo nodo partiremo dal nodo o elemento radice e indicheremo via via i nodi da percorrere per arrivare al nodo desiderato (ad es. la stringa `/film/title/year/city` identifica tutti gli elementi `<city>` contenuti all'interno degli elementi `<year>`, che a loro volta sono contenuti all'interno degli elementi `<title>` all'interno dell'elemento radice `<film>`.

2.1.1 Namespace

Per evitare equivoci per quanto riguarda il nome dei tag XML sono stati introdotti i namespace. I namespace vengono dichiarati all'interno dell'elemento radice oppure all'interno di un qualunque elemento interno al documento XML nel modo seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<film xmlns:cs="http://cs.unibo.it">
    <cs:title>The Great Gatsby</title>
    <cs:title>The Matrix</title>
    <title>Titanic</title>
</film>
```

Nel nostro esempio, i tag `<cs:title>` rappresentano qualcosa di totalmente diverso rispetto al tag `<title>`. Durante l'elaborazione del documento, tutti i prefissi `cs:` verranno sostituiti dall'URI indicato.

Per dichiarare un namespace si utilizza l'attributo `xmlns="URI1"` oppure `xmlns:nome="URI2"`. Nel primo caso tutti gli elementi del documento senza prefisso saranno considerati appartenenti al namespace di base URI1 mentre tutti gli elementi con prefisso `nome:` saranno considerati appartenenti al namespace URI2.

2.2 Document Type Definition

Per fornire uno strumento che definisca una struttura rigida per i documenti XML abbiamo a disposizione il DTD: un insieme di dichiarazioni di markup che definiscono un tipo di documento per un linguaggio di markup della famiglia SGML, tra cui XML, HTML, XHTML. Il DTD può essere

- interno al documento XML: posto immediatamente dopo il prologo
- esterno al documento XML: bisogna fornire il percorso all'interno del filesystem o l'URL dove il DTD è collocato

Prendendo come esempio il documento XML presentato nell'esempio 2.1 possiamo definire un DTD interno al documento. Il risultato è il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE film [
    <!ELEMENT film (title)>
    <!ELEMENT title (#PCDATA)>
    <!ATTLIST title language CDATA #IMPLIED>
    <!ATTLIST title country CDATA #IMPLIED>
]>
<film>
    <title language="english" country="USA">The Great Gatsby</title>
</film>
```

Se volessimo avere il DTD esterno al documento, il codice XML sarà il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE film SYSTEM "film.dtd">
<film>
    <title language="english" country="USA">The Great Gatsby</title>
</film>
```

Con il file `film.dtd` contenente il codice DTD precedente:

```
<!DOCTYPE film [
    <!ELEMENT film (title)>
    <!ELEMENT title (#PCDATA)>
    <!ATTLIST title language CDATA #IMPLIED>
    <!ATTLIST title country CDATA #IMPLIED>
]>
```

Qual è il significato del codice DTD? Prima di tutto, una sezione di codice interna al documento XML deve essere introdotta con `<!DOCTYPE nome-elemento-radice [` e successivamente chiusa con `]>`. È assolutamente necessario che tutti gli elementi presenti nel documento XML siano definiti nel DTD. Per questo motivo viene definito uno strumento rigido di definizione dei documenti XML.

Un elemento viene definito con `<!ELEMENT nome-elemento (elementi-contenuti)`. La sezione degli elementi contenuti prevede come sintassi alcuni simboli per indicare strutture particolari.

`<!ELEMENT nome-elemento EMPTY>`

È una dichiarazione che ci indica che l'elemento deve essere vuoto.

`<!ELEMENT nome-elemento ANY>`

Ci indica che l'elemento può contenere qualunque tipologia di contenuto: sia testo sia qualunque altro elemento.

`<!ELEMENT nome-elemento (#PCDATA)>`

In questo caso l'elemento può contenere soltanto del testo e non può contenere altri elementi XML.

Le parentesi `( )` creano un raggruppamento di elementi. I caratteri `+`, `?` e `*` sono dei quantificatori che hanno lo stesso significato che assumono nell'ambito delle espressioni regolari.

- `+` l'elemento o il gruppo precedente deve comparire almeno una volta
- `?` l'elemento o il gruppo precedente può comparire o meno
- `*` l'elemento o il gruppo precedente può comparire zero o più volte

Gli attributi di un elemento vengono definiti con:

`<!ATTLIST nome-elemento nome-attributo tipologia valore-attributo>`

La tipologia dell'attributo può essere:

- `CDATA` l'attributo contiene del testo
- `(val1|val2|val3|...)` il valore deve essere uno dei valori indicati

- `ID` il valore deve essere un identificativo univoco all'interno del documento
- `IDREF` il valore deve essere l'ID di un altro elemento all'interno del documento
- `IDREFS` il valore è una lista di ID di altri elementi separati da spazi

Queste sono le tipologie di attributi più utilizzate.
Il valore-attributo può assumere uno dei seguenti valori:

- *valore* il valore di default dell'attributo
- `#REQUIRED` l'attributo deve per forza avere un valore
- `#IMPLIED` l'attributo è opzionale
- `#FIXED valore` l'attributo ha un valore fissato

Il DTD prevede anche la definizione di entità che sono dei valori che potranno essere richiamati all'interno dei documenti XML che utilizzano il DTD oppure all'interno del DTD stesso. La dichiarazione di entità può essere utilizzata anche per includere dentro un DTD un altro DTD esterno. Vediamo un esempio di utilizzo per ciascun caso.

```
<!ENTITY autore "Ariel Demian">
```

Dentro il documento XML possiamo invocare questa entità utilizzando la sintassi `&autore;`. Nella dichiarazione possiamo inserire il segno percentuale (`%`) per indicare che l'entità può essere richiamata dentro il codice DTD. L'esempio successivo è estratto dai DTD apertamente forniti dalla casa editrice Elsevier sul proprio sito web.

```
<!ENTITY % sb.titles "((sb:title, sb:translated-title?)|sb:translated-title)">+
<!ELEMENT sb:contribution (sb:authors?,(%sb.titles;)?)>
```

Nell'esempio riportato sopra abbiamo la definizione dell'entità `sb.titles` che viene utilizzato dentro la definizione di un altro elemento. Questa funzionalità risulta utile quando il DTD assume proporzioni difficili da gestire se non si hanno possibilità di modularità del codice. Per includere un documento DTD esterno all'interno del proprio DTD per renderlo di fatto modulare si utilizza la sintassi seguente. L'esempio è estratto dal DTD fornito da Elsevier:

```
<!ENTITY % common.ent SYSTEM "common120.ent">
%common.ent;
```

Il file common120.ent è presente nella stessa cartella del documento in questione.

Questa vuole essere soltanto una breve introduzione ai DTD. Questo formato contiene altri elementi che per motivi di tempo e di spazio non sono stati descritti in questo elaborato. Se durante l'analisi dei documenti XML e DTD fornitimi qualcosa non è stato introdotto allora sfrutterò l'occasione per completare con la spiegazione di quella parte.

Un documento XML, come indicato precedentemente alla pagina 17, si dice *Well Formed* (ben formato) se rispetta le regole indicate per tutti i documenti XML e si dice *valido* se è ben formato e in più rispetta le regole indicate nel suo DTD.

2.3 Documenti Elsevier

I documenti forniti constano in approssimativamente 20.000 documenti XML e i DTD relativi. Il numero totale esatto di documenti non è precisato perché i documenti XML, essendo catalogati in riviste specializzate (journal), vengono scaricati manualmente per ogni journal. Virtualmente abbiamo accesso

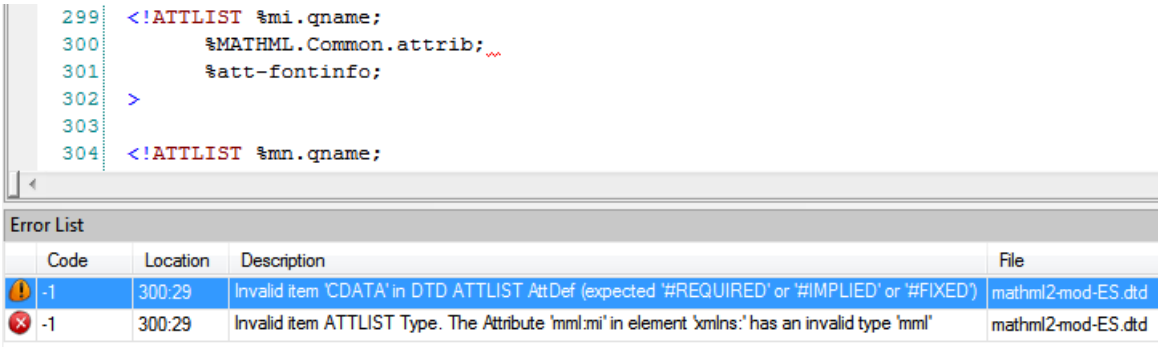

Figura 2.1: Errore generato dai DTD forniti da Elsevier in Liquid XML Studio

all'intera collezione di articoli Elsevier, codificati in XML, che conta approssimativamente 10 milioni di articoli. Per lo sviluppo del software e per effettuare i test sono stati utilizzati questi 20.000 documenti XML ma il sistema software può benissimo essere applicato all'intera collezione di documenti Elsevier.

Il primo passo del lavoro è stato aprire alcuni documenti per osservare la loro struttura e identificare la parte che ci interessa: i riferimenti bibliografici. Questa parte è stata subito identificata nella parte:

```
/full-text-retrieval-response/originalText/xocs:doc/xocs:serial-item/article/tail
/ce:bibliography
```

A questo punto risulta utile applicare le conoscenze acquisite e aprire i file DTD forniti da Elsevier[1]. È stato scaricata l'ultima versione Journal Article (JA) DTD 5.2.0 che comprende l'insieme di regole DTD distribuite su più file. È stato aperto il file `art520.dtd` con l'applicativo Liquid XML Studio, che verrà presentato nel prossimo capitolo. La prima cosa che viene notata è la presenza di un errore nel file `mathml2-mod-ES.dtd`. Nella figura 2.1 possiamo vedere la parte che genera l'errore.

Ho tentato più volte a seguire il filo dell'errore nei file DTD ma la moltitudine di regole presenti rende questo compito molto arduo, soprattutto se non ho conoscenza di qualcuno che abbia lavorato alla loro stesura. Il codice risulta poco commentato e difficile da interpretare manualmente a causa della dichiarazione di molte entità DTD. Siccome il DTD è un metodo di validazione molto rigido quindi ogni singolo elemento deve essere dichiarato e definito, e non si può fare a meno di una parte del DTD, si è rinunciato alla possibilità di risolvere questo errore e di validare i documenti per concentrarsi soltanto sulla parte che a noi interessa: i riferimenti bibliografici. Utilizzando l'applicativo PowerGREP, che verrà presentato nel prossimo capitolo, è stata eseguita una ricerca all'interno dei file DTD della stringa `ce:bibliography` che è appunto la parte dei documenti XML che contiene i riferimenti bibliografici.

La parte dei DTD che definisce i riferimenti bibliografici è stata trovata nel file `common120.ent` alla riga 405. A partire dall'analisi di questa parte di codice DTD è stato creato l'applicativo software.

```
<!ELEMENT   ce:bibliography     ( ce:section-title, ce:bibliography-sec+ )>
<!ELEMENT   ce:bibliography-sec ( ce:section-title?, ce:bib-reference+ )>
<!ELEMENT   ce:bib-reference    ( ce:label, ( ce:note |
                                  ( ( sb:reference | ce:other-ref )+, ce:note? ) ) )>
<!ELEMENT   ce:note             ( ce:simple-para+ )>

<!-- structured bibliographic reference (in name space ESSB) -->

<!ELEMENT   sb:reference        ( ce:label?, sb:comment?, ( sb:contribution, sb:comment? )?,(
    sb:host, sb:comment? )+ )>
<!ELEMENT   sb:contribution     ( sb:authors?, (%sb.titles;)? )>
<!ELEMENT   sb:host             ( ( ( ( sb:issue | sb:book | sb:edited-book ), sb:pages? )|
    sb:e-host ), ce:doi? )>
```

[1] http://www.elsevier.com/author-schemas/elsevier-xml-dtds-and-transport-schemas

```
<!ELEMENT   sb:comment          ( %nondisplay.data; )* >
<!ELEMENT   sb:authors          ( ( sb:collaboration | ( sb:author, sb:et-al? ) )+ )>
<!ELEMENT   sb:collaboration    ( %text.data; )* >
<!ELEMENT   sb:author           ( %name; )>
<!ELEMENT   sb:et-al            EMPTY>
<!ELEMENT   sb:title            ( sb:maintitle, sb:subtitle? )>
<!ELEMENT   sb:translated-title ( sb:maintitle, sb:subtitle? )>
<!ELEMENT   sb:maintitle        ( %text.data; )* >
<!ELEMENT   sb:subtitle         ( %text.data; )* >
<!ELEMENT   sb:issue            ( sb:editors?, (%sb.titles;)?, sb:conference?, sb:series,
    sb:issue-nr?, sb:date )>
<!ELEMENT   sb:conference       ( %text.data; )* >
<!ELEMENT   sb:editors          ( sb:editor+, sb:et-al? )>
<!ELEMENT   sb:editor           ( %name; )>
<!ELEMENT   sb:series           ( (%sb.titles;)?, sb:issn?, sb:volume-nr? )>
<!ELEMENT   sb:volume-nr        ( %richstring.data; )* >
<!ELEMENT   sb:issue-nr         ( %richstring.data; )* >
<!ELEMENT   sb:date             ( %richstring.data; )* >
<!ELEMENT   sb:pages            ( sb:first-page, sb:last-page? )>
<!ELEMENT   sb:first-page       ( %richstring.data; )* >
<!ELEMENT   sb:last-page        ( %richstring.data; )* >
<!ELEMENT   sb:book             ( (%sb.titles;)?, sb:edition?, sb:book-series?, sb:date+,
    sb:publisher?, sb:isbn? )>
<!ELEMENT   sb:edition          ( %richstring.data; )* >
<!ELEMENT   sb:publisher        ( sb:name, sb:location? )>
<!ELEMENT   sb:name             ( %richstring.data; )* >
<!ELEMENT   sb:location         ( %richstring.data; )* >
<!ELEMENT   sb:edited-book      ( sb:editors?, (%sb.titles;)?, sb:conference?,
    sb:edition?,sb:book-series?, sb:date+, sb:publisher?, sb:isbn? )>
<!ELEMENT   sb:book-series      ( sb:editors?, sb:series )>
<!ELEMENT   sb:e-host           ( ce:inter-ref?, sb:date? )>
<!ELEMENT   sb:issn             ( %string.data; )* >
<!ELEMENT   sb:isbn             ( %string.data; )* >

<!-- unstructured bibliographic reference -->

<!ELEMENT   ce:other-ref        ( ce:label?, ce:textref )>
<!ELEMENT   ce:textref          ( %textref.data; )* >
```

Per motivi di spazio, dal codice DTD sono state omesse tutte le dichiarazioni di attributi per il motivo che esse sono definite utilizzando entità quindi non forniscono molte informazioni al lettore, essendo le entità definite altrove nel documento e persino distribuite in altri file, creando una struttura difficile da rintracciare.

Per brevità, da ora in avanti, i riferimenti bibliografici saranno abbreviati con **rif. bib.**

Dall'analisi del DTD e dei documenti si può osservare che ci sono due tipologie di rif. bib.:

- strutturati (`sb:reference`): i campi vengono inseriti dentro tag XML quindi le informazioni dovrebbero essere di più facile estrazione
- non strutturati (`ce:other-ref`): il riferimento bibliografico viene inserito soltanto come una intera stringa di testo quindi le informazioni dovranno essere estratte manualmente utilizzando opportunamente le espressioni regolari (saranno presentate nel prossimo capitolo)

Di seguito abbiamo due esempi di rif. bib. strutturati:

```
<ce:bib-reference id="b0050">
    <ce:label>[10]</ce:label>
    <sb:reference id="h0045">
```

```
        <sb:contribution langtype="en">
            <sb:authors>
                <sb:author>
                    <ce:given-name>P.P.</ce:given-name>
                    <ce:surname>Prochazka</ce:surname>
                </sb:author>
                <sb:author>
                    <ce:given-name>J.</ce:given-name>
                    <ce:surname>Trckova</ce:surname>
                </sb:author>
                <sb:author>
                    <ce:given-name>T.-S.</ce:given-name>
                    <ce:surname>Lok</ce:surname>
                </sb:author>
            </sb:authors>
            <sb:title>
                <sb:maintitle>Influence of dislocations on bumps occurrence in deep
                    mines</sb:maintitle>
            </sb:title>
        </sb:contribution>
        <sb:host>
            <sb:issue>
                <sb:series>
                    <sb:title>
                        <sb:maintitle>Acta Geodynam Geomater</sb:maintitle>
                    </sb:title>
                    <sb:volume-nr>7</sb:volume-nr>
                </sb:series>
                <sb:issue-nr>4</sb:issue-nr>
                <sb:date>2010</sb:date>
            </sb:issue>
            <sb:pages>
                <sb:first-page>475</sb:first-page>
                <sb:last-page>487</sb:last-page>
            </sb:pages>
        </sb:host>
    </sb:reference>
</ce:bib-reference>
<ce:bib-reference id="BIB10">
    <ce:label>10.</ce:label>
    <sb:reference>
        <sb:contribution langtype="en">
            <sb:authors>
                <sb:author>
                    <ce:given-name>H.K.</ce:given-name>
                    <ce:surname>Lim</ce:surname>
                </sb:author>
                <sb:author>
                    <ce:given-name>T.H.</ce:given-name>
                    <ce:surname>Lee</ce:surname>
                </sb:author>
                <sb:author>
                    <ce:given-name>T.S.</ce:given-name>
                    <ce:surname>Low</ce:surname>
                </sb:author>
            </sb:authors>
            <sb:title>
                <sb:maintitle>An investigation into neural net control system with integral
                    action</sb:maintitle>
```

```
            </sb:title>
        </sb:contribution>
        <sb:host>
            <sb:edited-book>
                <sb:book-series>
                    <sb:series>
                        <sb:title>
                            <sb:maintitle>
                                <ce:italic>Proc. Int. Joint Conf. Neural Networks
                                    (IJCNN'91)</ce:italic>,Singapore
                            </sb:maintitle>
                        </sb:title>
                        <sb:volume-nr>Vol. 2</sb:volume-nr>
                    </sb:series>
                </sb:book-series>
                <sb:date>Nov. 1991</sb:date>
            </sb:edited-book>
            <sb:pages>
                <sb:first-page>1053</sb:first-page>
                <sb:last-page>1058</sb:last-page>
            </sb:pages>
        </sb:host>
    </sb:reference>
</ce:bib-reference>
```

Il problema dei riferimenti bibliografici è che in questo formato sono inutilizzabili e non sono interconnesse (per sapere tutte le opere di un certo autore è necessario effettuare una ricerca su tutti i documenti). L'obiettivo del software sarà di analizzare le informazioni contenute e di rielaborarle in modo da renderle facilmente fruibili e pubblicabili online.

Dall'osservazione del DTD e di qualche decina di esempi sono state accertate alcune cose:

- l'attributo `id` del tag `ce:bib-reference` ha un valore univoco all'interno di ogni documento quindi può essere sfruttato per la catalogazione dei rif. bib.
- ci sono quattro tipologie di host per i documenti a cui i rif. bib. fanno riferimento: issue, book, edited book ed e-host
- le informazioni presenti in ogni singolo elemento possono variare di molto, ad es. nel primo rif. bib. riportato sopra, l'elemento `sb:volume-nr` contiene il numero intero 7 mentre nel secondo rif. bib. lo stesso elemento contiene la stringa `Vol. 2`; questo rende la catalogazione uniforme delle informazioni molto ardua
- anche gli elementi stessi sono molto variabili all'interno dei rif. bib. strutturati in quanto il DTD è molto permissivo e questo rende la catalogazione ancora più difficile

Di seguito vengono riportati tre esempi di rif. bib. non strutturati presi casualmente dai documenti:

```
<ce:bib-reference id="BIB14">
    <ce:label>[14]</ce:label>
    <ce:other-ref>
        <ce:textref>W. Mitchell, How weak is the closed, unbounded filter?,
            Preprint.</ce:textref>
    </ce:other-ref>
</ce:bib-reference>
<ce:bib-reference id="b0085">
    <ce:label>[17]</ce:label>
    <ce:other-ref id="j0025">
```

```
        <ce:textref>Goldberg DE. Genetic algorithms in search, optimization, and, machine
            learning; 1989.</ce:textref>
    </ce:other-ref>
</ce:bib-reference>
<ce:bib-reference id="bib20">
    <ce:label>[20]</ce:label>
    <ce:other-ref>
        <ce:textref>Şen Z. Genetik Algoritmalar ve Eniyilieme
            Yöntemleri (Genetic algorithms and optimization methods). Water Foundation,
            Istanbul; 2004 [in Turkish].</ce:textref>
    </ce:other-ref>
</ce:bib-reference>
```

Da questi esempi si evince chiaramente quanto siano ancora più variabili e difficili da catalogare le informazioni presenti all'interno dei rif. bib. non strutturati.

Nel primo esempio abbiamo soltanto il nome dell'autore "W. Mitchell", il titolo dell'opera "How weak is the closed, unbounded filter?" e il fatto che è ancora una bozza. Non è presente alcuna informazione riguardo all'anno di pubblicazione e al luogo di produzione, la virgola viene usata come separatore fra il nome dell'autore e il nome dell'opera ma viene anche usata all'interno del nome dell'opera.

Nel secondo esempio il nome dell'autore non è chiaro, il nome dell'autore viene separato dal titolo dell'opera con un punto ma poi viene usato un punto e virgola per separare il titolo dell'opera dall'anno. Si può notare anche una virgola di troppo dopo `and`.

Nel terzo esempio abbiamo il titolo dell'opera in lingua originale e tra parentesi in inglese. Dopo l'anno di pubblicazione è presente la stringa `[in Turkish]` che ci indica che la lingua originale è il turco.

A questo punto possiamo passare ai prossimi capitoli per analizzare in dettaglio la soluzione adottata, che sarà per forza di cose una soluzione euristica.

Capitolo 3

Strumenti utilizzati

3.1 PowerGREP

PowerGREP[1] è un software derivante dal **grep** (General Regular Expression Print) che è un comando dei sistemi UNIX che ricerca in uno o più file di testo le parti che corrispondono all'espressione regolare o alla stringa letterale indicata e fornisce una lista delle righe corrispondenti con i nomi dei file che le contiene.

Il problema che si cerca di risolvere con questo software è di poter eseguire una ricerca veloce in un insieme molto grande di file in formato testuale. La ricerca ha come obiettivo alcune determinate stringhe che possono essere generalizzate utilizzando espressioni regolari. Le limitazioni del semplice comando grep dei sistemi UNIX sono le prestazioni e le difficoltà di visualizzazione dei risultati quando si ha un numero molto elevato di file di grandi dimensioni.

La figura 3.1 riporta la schermata iniziale del software.

Nella parte sinistra abbiamo la possibilità di scegliere le cartelle o anche i singoli file da analizzare. Al centro si possono inserire espressioni regolari, stringhe letterali, dati binari oppure una lista delle cose precedenti. Nella figura 3.2 possiamo osservare i risultati di un'interrogazione nei documenti utilizzando la stringa letterale `<ce:other-ref` (non si inserisce il segno > alla fine della stringa perché il tag potrebbe avere degli attributi e quindi non finire immediatamente con il segno indicato) che compaiono al centro dello schermo.

Facendo doppio click su un risultato, viene aperto l'editor testuale con il file contenente il risultato alla riga corrispondente. Queste sono le funzionalità utilizzate per quanto riguarda il lavoro di tesi. Il software offre alcune funzionalità addizionali, come ad es. il salvataggio della sequenza di azioni svolte su un file e riprodurre questa sequenza su altri file.

La caratteristica che rende questo software adeguato al compito richiesto è la velocità con cui analizza migliaia di file e il fatto che la seconda volta che viene eseguita un'interrogazione sugli stessi file, il tempo richiesto è molto minore. Queste caratteristiche rendono PowerGREP molto più potente di altri software simili a grep su Windows.

PowerGREP è stato utilizzato in Free Trial e soltanto per scopi educativi, che non divergono dallo scopo della tesi. Tutti i diritti appartengono a Just Great Software Co. Ltd. Copyright © 2002-2013 Jan Goyvaerts.

[1] http://www.powergrep.com/

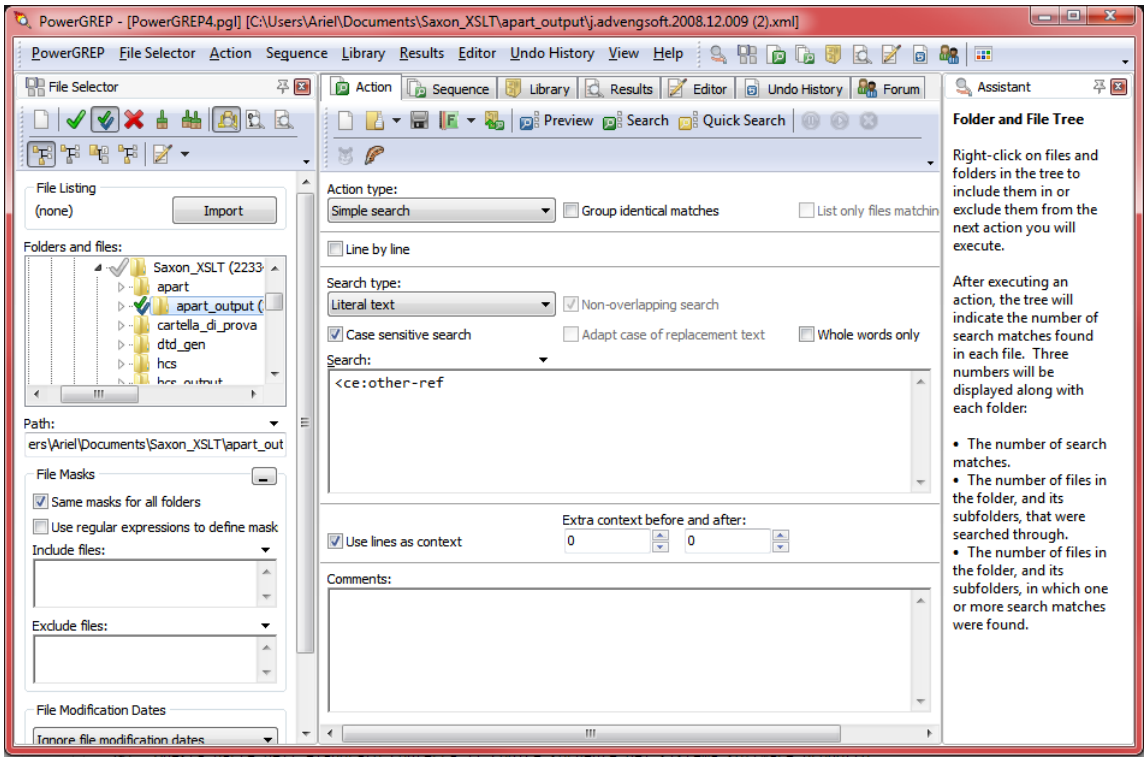

Figura 3.1: Schermata iniziale di PowerGREP

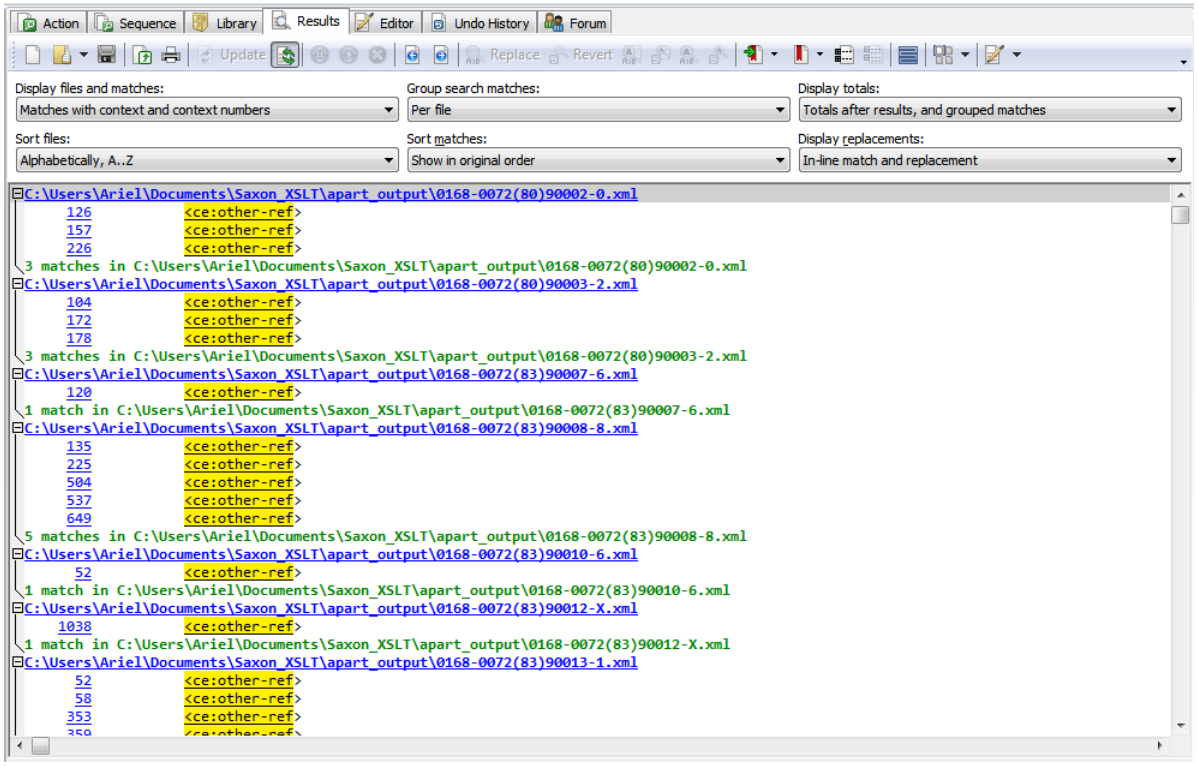

Figura 3.2: Schermata dei risultati di un'interrogazione in PowerGREP

3.2 Espressioni regolari

Il fulcro del funzionamento di PowerGREP sono le espressioni regolari. Nell'ambito del lavoro della tesi sono state ampiamente utilizzate per trovare parti specifiche dei documenti XML e DTD. In questo sottocapitolo verrà esposta una breve introduzione a questo strumento utilissimo nell'ambito dell'analisi di documenti testuali informatici.

Una espressione regolare (*regular expression* detta anche *regex* o *regexp*) è definita come una sequenza di caratteri che assume un certo significato dopo essere stata interpretata da un processore di espressioni regolari, e che corrisponde a determinate stringhe.

Un carattere può essere interpretato in due modi:

- letterale: il carattere corrisponde allo stesso carattere nella stringa da analizzare
- metacarattere: il carattere ha un significato speciale in base alla stringa analizzata e in base agli altri metacaratteri presenti

I caratteri alfanumerici e lo spazio sono tutti interpretati come caratteri letterali quindi la stringa `Abc 1234` corrisponde a `Abc 1234`.

Nella pagina seguente verranno presentati i principali metacaratteri delle espressioni regolari.

Questi sono gli elementi di base delle espressioni regolari e sono sufficienti a creare espressioni regolari complesse e utili a catturare un grande range di stringhe.

Ecco alcuni esempi di utilizzo delle espressioni regolari con gli elementi presentati sopra:

- `(ab)?` corrisponde ad `ab` oppure alla stringa vuota
- `(ab)|c+` corrisponde ad `ab`, a `c`, a `cc`, a `ccc` ...
- `^[^c].*[dce]$` corrisponde a una stringa che non inizia con `c`, ha un numero arbitrario di caratteri e finisce per `d`, `c` oppure `e`
- `[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}` corrisponde a un indirizzo email

Metacarattere	Significato
`.`	il carattere punto corrisponde a qualunque carattere, ad eccezione del carattere di newline
`^`	questo carattere corrisponde all'inizio della stringa o della riga; viene posto all'inizio dell'espressione regolare per far corrispondere il seguito dell'espressione all'inizio della stringa o della riga
`$`	questo carattere corrisponde alla fine della stringa; viene posto alla fine dell'espressione regolare per far corrispondere la parte precedente dell'espressione alla fine della stringa o della riga
`( )`	le parentesi tonde raggruppano una sottoespressione regolare che può corrispondere a una certa stringa, la quale può essere referenziata più avanti nell'espressione regolare
`\n`	ad `n` si sostituisce un numero e corrisponde alla `n`-esima sottoespressione regolare trovata
`[ ]`	le parentesi quadre corrispondono a un solo carattere che deve essere presente all'interno delle parentesi; con questo strumento si possono anche definire range, ad es. `[a-z]` corrisponde ad un carattere latino minuscolo; tutti gli altri caratteri, eccetto `^` assumono significato letterale
`[^ ]`	le parentesi quadre con all'inizio il carattere `^` corrispondono a un solo carattere che non deve essere presente all'interno delle parentesi; in altre parole, rispetto al precedente ne è la negazione; con questo strumento si possono anche definire range, ad es. `[^0-9]` corrisponde ad un carattere che non sia una cifra; tutti gli altri caratteri, eccetto `^` assumono significato letterale
`+`	viene aggiunto immediatamente dopo un elemento e corrisponde ad almeno una o più ripetizioni successive concatenate di quel elemento
`?`	viene aggiunto immediatamente dopo un elemento e corrisponde a nessuna o una sola volta l'elemento
`*`	viene aggiunto immediatamente dopo un elemento e corrisponde a nessuna o più ripetizioni successive concatenate di quel elemento
`{n,m}`	con `n` $\leq$ `m` viene aggiunto immediatamente dopo un elemento e corrisponde all'elemento ripetuto almeno `n` e al più `m` volte
`|`	l'operatore di scelta viene posto in mezzo a due elementi e fa corrispondere l'espressione regolare al primo oppure al secondo elemento
`\`	questo metacarattere è molto importante e server per fare l'escape del carattere immediatamente successivo se è un metacarattere oppure assume un significato diverso se è una lettera
`\d`	corrisponde a una cifra; si può anche scrivere come [0-9]
`\D`	corrisponde a un carattere che non sia una cifra; si può anche scrivere come `[^0-9]`
`\a`	corrisponde a un carattere dell'alfabeto latino
`\s`	corrisponde al carattere spazio oppure al tab

Un altro tool importante delle espressioni regolari, senza il quale alcune interrogazioni sarebbero state impossibili è il cosiddetto *lookaround*. Per spiegare questo strumento è molto semplice e immediato utilizzare degli esempi. I lookaround si distinguono in:

- look-ahead: dice al processore di regex di fermarsi e di guardare avanti nella stringa per cercare una corrispondenza
  - positivo: ha la forma `q(?=(ab))` e corrisponde a `q` seguito da `ab` ma senza includere `ab` nella corrispondenza
  - negativo: ha la forma `q(?!(ab))` e corrisponde a `q` che non deve essere seguito da `ab` ma senza includere i due caratteri seguenti nella corrispondenza
- look-behind: dice al processore di regex di fermarsi e di guardare indietro nella stringa per cercare una corrispondenza
  - positivo: ha la forma `(?<=(ab))c` e corrisponde a un `c` preceduto da `ab` ma senza includere i due caratteri nella corrispondenza
  - negativo: ha la forma `(?<!(ab))c` e corrisponde a un `c` che non sia preceduto da `ab` ma senza includere i due caratteri nella corrispondenza

Di questi strumenti, il più significativo per questo lavoro di tesi è stato il look-ahead negativo per consentire la corrispondenza di stringhe che non contengano una data sottostringa. Ad es. se vogliamo corrispondere tutte le stringhe che non contengano la parola `prof` bisogna utilizzare la seguente espressione regolare:

```
^((?!(prof)).)*$
```

Questa regex inizia ad analizzare la stringa vuota perché abbiamo immediatamente un look-ahead. Se la stringa vuota non è immediatamente seguita dalla parola `prof` allora il processore considera la seguente porzione della regex che è il punto e che corrisponde a qualunque carattere. Dopo il primo carattere corrisposto viene di nuovo presa in considerazione la stringa vuota immediatamente successiva e si ripete lo stesso processo perché abbiamo il metacarattere asterisco `*`. Questo processo viene eseguito per ogni singolo carattere della stringa, dall'inizio alla fine grazie ai metacaratteri `^` e `$`.

Da notare che l'ambiente di sviluppo utilizzato, Java, non implementa pienamente i lookaround[2] quindi non si possono inserire espressioni regolari complesse dentro il lookaround. Questo rende il lavoro con le espressioni regolari più arduo, in particolare per quanto riguarda i riferimenti bibliografici non strutturati.

3.3 Jena

Jena[3] è una libreria java pubblicata da *The Apache Software Foundation* sotto licenza *Apache License, Version 2.0*[4] per la manipolazione dello RDF (Resource Description Framework), per l'interrogazione di dataset RDF con SPARQL, per archiviare e offrire la possibilità di interrogazioni SPARQL ad alte prestazioni con *TDB* componente di Jena, per offrire i propri dataset RDF come un end-point interrogabile con SPARQL su HTTP grazie a *Fuseki* componente di Jena, per creare nuove ontologie con le Jena Ontology API.

Il primo passo consiste nello scaricare la libreria dal sito ufficiale[5]. In base all'ambiente di sviluppo java utilizzato ci sono diversi passi da seguire per utilizzare la libreria.

In Eclipse 5.0.0[6], dopo aver creato un nuovo progetto java si esegue il seguente import:

```
import com.hp.hpl.jena.rdf.model.Model;
```

[2] http://www.regular-expressions.info/lookaround.html#limitbehind
[3] https://jena.apache.org/
[4] http://www.apache.org/licenses/LICENSE-2.0
[5] https://jena.apache.org/download/index.cgi
[6] https://www.eclipse.org/

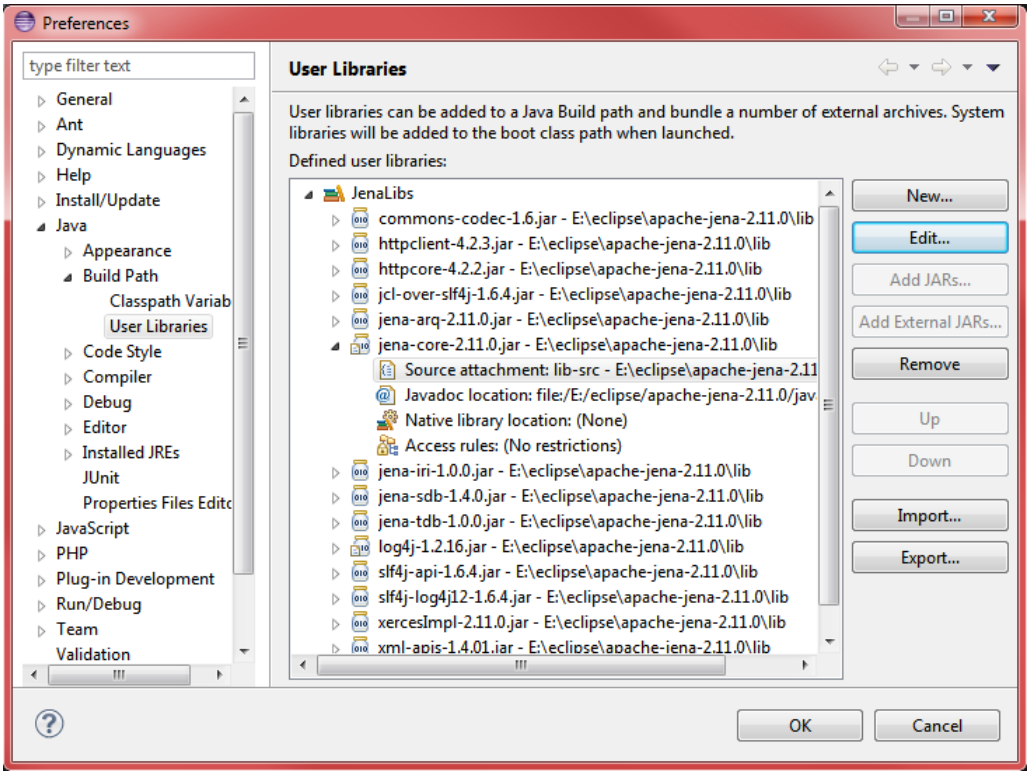

Figura 3.3: Configurare Jena su Eclipse 5.0.0

A questo punto, se si tenta di compilare verrà fornito un errore dal compilatore in quanto la libreria è mancante. È necessario aprire il menu

`Windows` → `Preferences` → `Java` → `Build Path` → `User Libraries` → `New ...`

Dopo aver creato la nuova libreria bisogna aggiungere i file .jar di Jena con il pulsante

`Add External JARs...`

Dai file .jar appena aggiunti si seleziona `jena-core-2.11.0.jar` poi `Source attachment:` e si fa click su `Edit...` come si può vedere nella figura 3.3. Si raggiunge la cartella dove sono stati dearchiviati i file di jena e si seleziona la cartella `apache-jena-2.11.0/lib-src`.

Lo stesso procedimento bisogna fare per la voce `Javadoc location:`

Dalla configurazione del progetto si sceglie `Build Path` → `Add Libraries...` e si aggiunge la libreria appena configurata.

Il motivo per cui ho inserito in questo elaborato la configurazione dell'ultima versione di Jena a questa data (versione 2.11.0) è che non vi è ancora una guida ufficiale su come configurare questa versione della libreria.

Per riprodurre il semplice statement RDF a pagina 5 in Jena e stampare a schermo lo statement in formato RDF/XML possiamo eseguire il codice:

```
import org.apache.jena.atlas.logging.Log;
import com.hp.hpl.jena.datatypes.xsd.XSDDatatype;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.Resource;

public class Main {
  public static void main(String[] args) {
    Log.setLog4j("jena-log4j.properties");
    Model model = ModelFactory.createDefaultModel();
```

```
    String itwiki = "http://it.wikipedia.org/wiki/";
    String foaf = "http://xmlns.com/foaf/0.1/";
    Resource resource = model.createResource(itwiki + "Paolo_Borsellino");
    Property property = model.createProperty(foaf + "familyName");
    resource.addProperty(property, "Borsellino", XSDDatatype.XSDstring);
    model.write(System.out, "RDF/XML");
  }
}
```

Il comando `Log.setLog4j("jena-log4j.properties");` serve per indicare alla libreria Jena il file di configurazione per il componente del log. Il file `jena-log4j.properties` deve essere incluso nella cartella del progetto.

L'output del codice sarà:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="http://xmlns.com/foaf/0.1/" >
  <rdf:Description rdf:about="http://it.wikipedia.org/wiki/Paolo_Borsellino">
    <j.0:familyName
        rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Borsellino</j.0:familyName>
  </rdf:Description>
</rdf:RDF>
```

Per verificare la validità degli statement fatti possiamo utilizzare il meccanismo fornito da Jena. Questo meccanismo può essere considerato come il reasoner più semplice. Il codice sorgente per la verifica della validità viene aggiunto dopo l'ultima riga del codice sorgente precedente.

```
import java.util.Iterator;
import org.apache.jena.atlas.logging.Log;
import com.hp.hpl.jena.datatypes.xsd.XSDDatatype;
import com.hp.hpl.jena.rdf.model.InfModel;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.reasoner.ValidityReport;
import com.hp.hpl.jena.reasoner.ValidityReport.Report;

public class Main {

  public static void main(String[] args) {
    Log.setLog4j("jena-log4j.properties");
    ...
    model.write(System.out, "RDF/XML");
    InfModel infModel = ModelFactory.createRDFSModel(model);
    ValidityReport validity = infModel.validate();
    if (validity.isValid()) {
      System.out.println("Valid!");
    } else {
      System.out.println("Conflicts");
      for (Iterator<Report> i = validity.getReports(); i.hasNext();) {
        System.out.println(" - " + i.next());
      }
    }
  }
}
```

3.4 Liquid XML Studio

Liquid XML Studio[7] è un software per l'elaborazione di file XML. Nel download viene incluso anche Liquid Large File Editor e Liquid XML Data Binder.

Liquid XML Studio è stato utilizzato in Free Trial e soltanto per scopi educativi, che non divergono dallo scopo della tesi. Tutti i diritti appartengono a © 2014 Liquid Technologies Limited.

[7] http://www.liquid-technologies.com/

Capitolo 4

Software prodotto

Dalla breve introduzione all'inizio dell'elaborato, è stato messo in evidenza che lo scopo del sistema software è di analizzare ed elaborare i documenti XML forniti per estrarre e rendere disponibili in un formato riutilizzabile le informazioni relative ai riferimenti bibliografici (qui abbreviati con "rif. bib."). Dopo un'introduzione essenziale dei documenti nel secondo capitolo, in questo capitolo verranno esposti sommariamente i meccanismi dietro il software prodotto.

Dall'analisi con PowerGREP dei documenti XML, 20.877 in totale, vi sono 361.210 rif. bib. strutturati e 103.289 non strutturati. La ricerca è stata effettuata prima con la stringa `<sb:reference` per i strutturati e `<ce:other-ref` per i non strutturati.

Il linguaggio di programmazione utilizzato principalmente è stato Java, con JRE 7 (Java Runtime Environment). Dalle istruzioni per l'installazione delle librerie Jena nel capitolo 3.3, si rileva che l'IDE utilizzato è stato Eclipse 5.

Altre due librerie utilizzate in parte sono state:

- Commons Validator[1] fornito da The Apache Software Foundation per identificare e validare gli URL
- Restlet Framework[2] per Java utilizzato soltanto in piccola parte per aggiungere i parametri GET necessari agli URL creati per le risorse identificate

La soluzione adottata utilizza una gerarchia di regole. Ogni regola riconosce se il rif. bib. che riceve in ingresso adempie alle caratteristiche indicate da quella regola. Nel caso in cui il rif. bib. adempia pienamente alle caratteristiche indicate, viene elaborato e le informazioni estratte vengono aggiunte all'output, altrimenti si passa alla regola successiva. Come ultima regola della gerarchia abbiamo la regola "cattura-tutto" che salva il rif. bib. come semplice testo, considerato che le regole precedenti non siano riuscite a identificarne la tipologia e il formato. Questa gerarchia viene implementata con un `java.util.List` di regole. La prima regola riconosce i rif. bib. strutturati. La parte restante delle regole sono adibite al riconoscimento dei vari formati di rif. bib. non strutturati. In figura 4.1 possiamo osservare questo semplice meccanismo.

La lista contiene diversi oggetti che implementano l'interfaccia `PatternRecognizer`. La struttura delle classi può essere visualizzata in figura 4.2. Questa scelta implementativa permette di aggiungere un numero indefinito di regole per il riconoscimento e l'elaborazione di rif. bib. non strutturati.

[1] http://commons.apache.org/validator
[2] http://restlet.org/

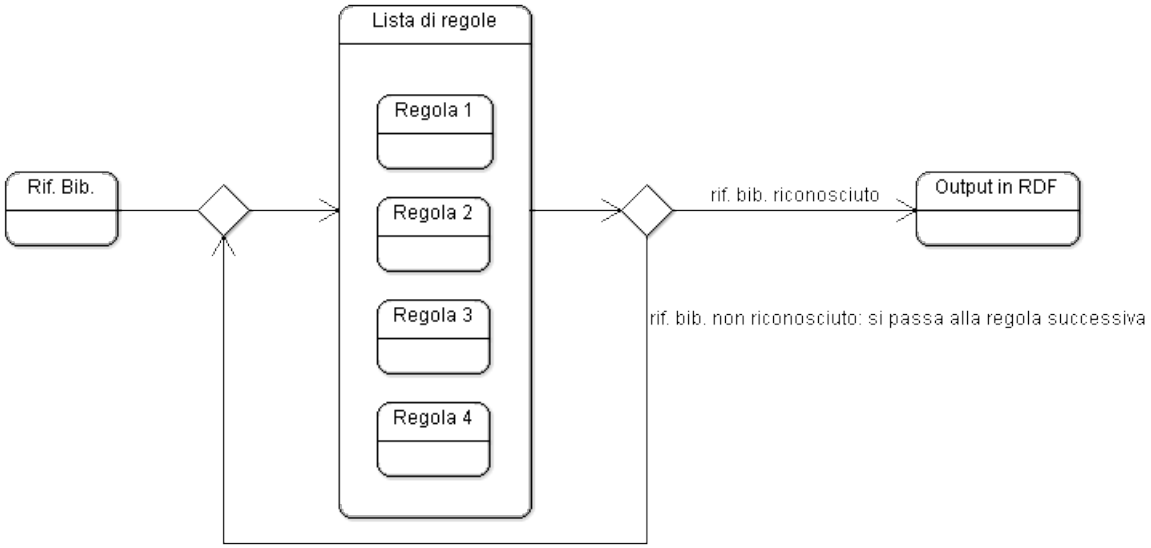

Figura 4.1: Funzionamento della lista di regole.

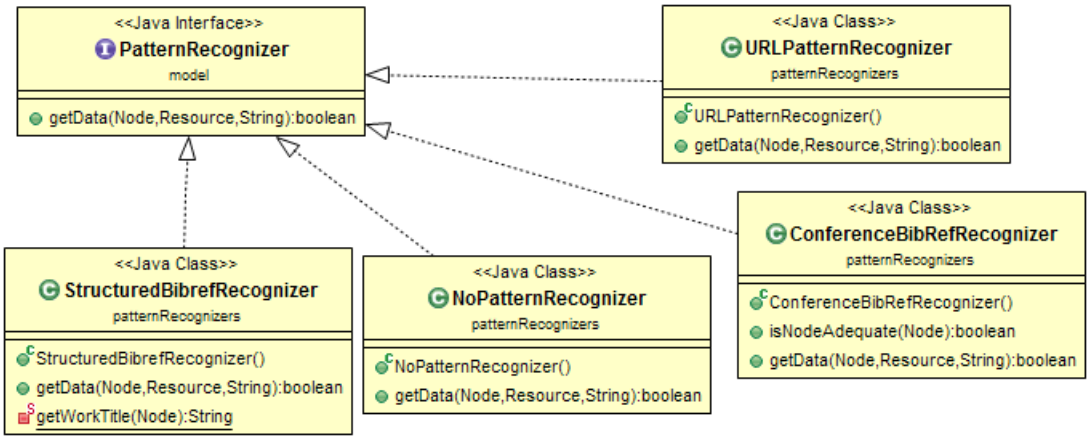

Figura 4.2: Struttura delle classi delle regole

Le classi di regole implementate presenti in figura 4.2 sono:

- `StructuredBibrefRecognizer` riconosce ed elabora i rif. bib. strutturati
- `ConferenceBibRefRecognizer` riconosce ed elabora i rif. bib. non strutturati che presentano una certa struttura riconoscibile ad articoli che sono prodotti di una conferenza
- `URLPatternRecognizer` riconosce ed elabora i rif. bib. non strutturati che presentano un semplice link URL alla risorsa riferita oppure che non sono stati elaborati dai PatternRecognizer precedenti
- `NoPatternRecognizer` scelta finale che salva il rif. bib. come una semplice stringa di testo

Come esempio di funzionamento possiamo analizzare i vari stadi del software a runtime durante l'analisi di un rif. bib. strutturato. La classe `Main` (fig. 4.4) inizializza tutte le variabili di sistema e le variabili necessarie al funzionamento di tutte le classi. I parametri in ingresso richiesti dal software sono:

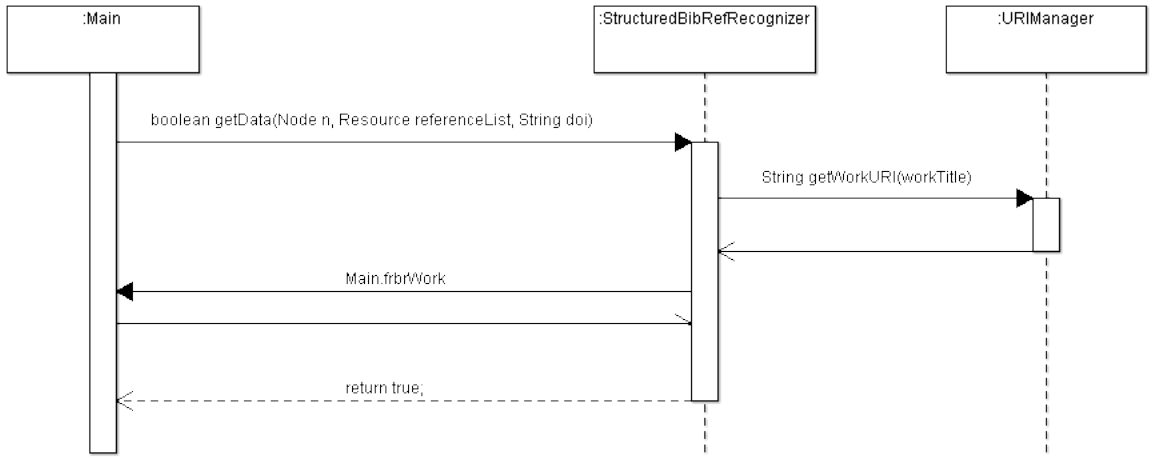

Figura 4.3: Diagramma di sequenza del riconoscimento di un rif. bib. strutturato

1. il path del file `namespaces.xml` contenente i namespace necessari per la generazione degli statement; i namespace necessari sono: fabio, biro, frbr, pro, dc, prism, foaf, rdfs, xsd, swc, geo, co, rdf, doco, doi; ad eccezione di "doi", tutti gli altri namespace sono ontologie
2. il path della cartella contenente i file XML da analizzare
3. il path del file dove verrà scritto l'output
4. il file `jena-log4j.properties` per la corretta configurazione del log di Jena
5. un URL che verrà utilizzato come base per gli statement RDF prodotti

Al completamente della configurazione delle variabili, viene analizzato ogni singolo file utilizzando espressioni XPath per ottenere i nodi XML che ci interessano. Una volta individuato un certo nodo contenente un rif. bib. esso viene passato alla lista di `PatternRecognizer` per essere riconosciuto dal recognizer adatto.

Nel diagramma di sequenza in figura 4.3 possiamo osservare la chiamata della classe `Main` alla classe `StructuredBibrefRecognizer`. I parametri passati al metodo `getData()` sono:

- `Node n` il nodo contenente il rif. bib.
- `Resource referenceList` la risorsa RDF creata nel `Main` alla quale verranno aggiunte tutte le altre risorse RDF che descrivono il singolo rif. bib.
- `String doi` l'identificativo DOI dell'articolo contenuto nel documento XML

Per creare gli statement RDF, il recognizer ha bisogno degli URL che devono essere generati utilizzando le informazioni estrapolate. A questo scopo la classe `URIManager` (fig. 4.5) fornisce il risultato necessario attraverso il metodo indicato. Oltre all'URL generato per le risorse, il recognizer ha bisogno dei vari URL che denotano le classi e le proprietà delle ontologie impiegate che sono inizializzate come attributi `public` della classe `Main`. Alla fine dell'elaborazione gli statement vengono aggiunti e collegati alla risorsa `referenceList`, e il valore di ritorno della funzione è `true` se e soltanto se il rif. bib. è stato riconosciuto e analizzato correttamente.

Il funzionamento degli altri recognizer è simile alla descrizione sopra riportata.

4.1 Riferimenti bibliografici strutturati

La classe `StructuredBibrefRecognizer` è adibita al riconoscimento ed elaborazione dei rif. bib. strutturati. In assenza del buon funzionamento dei DTD forniti dalla casa editrice Elsevier, bisogna validare manualmente la struttura ad albero dei nodi che identificano i rif. bib. strutturati per una corretta catalogazione delle informazioni. Ricordando gli esempi riportati a pagina 24, essi dovranno essere validati per ogni singolo nodo presente secondo la struttura indicata dal DTD fornito da Elsevier riportato a pagina 22.

Viene riportato un esempio di codice che valida una porzione della struttura dei rif. bib. strutturati:

```
...
if (n2.getNodeName().equals("sb:contribution")){
  numberOfContribution++;
  int numberOfAuthors = 0, numberOfTitle = 0, numberOfTranslatedTitle = 0;
  for (Node n3 : new MyNode(n2)){
    if (n3.getNodeName().equals("sb:authors")){
      numberOfAuthors++;
      ...
    }
    if ((n3.getNodeName().equals("sb:title") && numberOfTitle == numberOfTitle++) ||
        (n3.getNodeName().equals("sb:translated-title") && numberOfTranslatedTitle ==
        numberOfTranslatedTitle++)){
      ...
    }
  }
  if (numberOfAuthors > 1 || numberOfTitle > 1 || numberOfTranslatedTitle > 1 ||
      numberOfTitle + numberOfTranslatedTitle == 0)
    return false;
}
...
```

Questo snippet di codice viene attivato quando si incontra l'elemento con nome `sb:contribution`. Il codice DTD che ne definisce la struttura è il seguente:

```
<!ENTITY % sb.titles "((sb:title, sb:translated-title?)|sb:translated-title)" >
...
<!ELEMENT sb:contribution (sb:authors?, (%sb.titles;)? )>
```

La prima istruzione incrementa la variabile che segna il numero di occorrenze dell'elemento per fare una verifica successiva. Questo elemento, secondo le specifiche del DTD, contiene al massimo un elemento `sb:authors`, al massimo un elemento `sb:title`, al massimo un elemento `sb:translated-title` e almeno uno degli ultimi due elementi. Ogni volta che viene incontrato un nodo con il nome corrispondente ne viene incrementata la variabile. Il controllo viene fatto alla fine del ciclo iterativo. Se la struttura non è adeguata viene ritornato un valore `false` e non vengono aggiunti gli statement relativi a questo rif. bib. al modello Jena.

Per iterare sui nodi figli di un nodo è stata implementata una nuova classe `MyNode` che implementa l'interfaccia `Iterable<Node>`, impiegando il design pattern Iterator.

Gli statement contenenti le informazioni estrapolate dal singolo rif. bib. vengono aggiunte a un modello Jena locale. Se viene rispettata la struttura indicata dal DTD, il modello locale viene aggiunto al modello globale contenente tutti gli statement.

4.2 Riferimenti bibliografici non strutturati

Il problema dei rif. bib. non strutturati è che non esiste un solo formato standard con il quale sono espressi, però esistono dei pattern ricorrenti e analizzando manualmente i documenti si è cercato di riconoscere i più frequenti. Le classi di regole per i rif. bib. non strutturati fanno largo uso di espressioni regolari per riconoscere i pattern corrispondenti. L'espressione regolare presente in `ConferenceBibRefRecognizer` per riconoscere i rif. bib. non strutturati contenenti un articolo generato durante una conferenza è la seguente:

```
^((\p{L}+ )+\p{Lu}+(,|\.))+ .+?. In: ((?!(,|:)).)+?; \d{3,4}(\.)?$
```

Il costrutto `\p{L}` identifica una qualunque lettera secondo la codifica Unicode, il costrutto `\p{Lu}` identifica una qualunque lettera maiuscola secondo la codifica Unicode. Il costrutto `\d{3,4}` identifica precisamente un sequenza di 3 o 4 cifre. L'espressione regolare sopra riportata identifica pienamente la seguente stringa:

```
Liu GP, Rees D. Stability criteria of networked predictive control systems with random
    network delay. In: 44th IEEE conference on decision and control; 2005.
```

Il presente esempio contiene i nomi degli autori all'inizio, seguiti dal titolo dell'opera "Stability criteria of networked…", dal nome della conferenza "44th IEEE conference…" e dall'anno presumibilmente di svolgimento della conferenza "2005". Le informazioni dovranno essere estratte una ad una con un processo non standardizzato ma che dovrà essere adattato per ogni formato di rif. bib. riconoscendo parti dell'intera stringa e ritagliando le parti che ci interessano. Di seguito abbiamo l'estrazione del titolo dalla stringa precedente:

```
...
if (textRef.matches("^((\\p{L}+ )+\\p{Lu}+(,|\\.))+ .+?. In: ((?!(,|:)).)+?;
    \\d{3,4}(\\.)?$")){
  String title = textRef.replaceFirst("^(((\\p{L})+ )+\\p{Lu}+(,|\\.) )+", "");
  title = title.replaceFirst("\\. In: .*?; \\d{3,4}(\\.)?$", "");
  title=title.trim();
  ...
}
...
```

In questo esempio la modalità di estrazione del titolo consiste nel tagliare fuori le parti che non ci interessano utilizzando il metodo `String.replaceFirst(String regex, String replacement)`. Il primo argomento è l'espressione regolare che riconosce la parte da rimuovere e il secondo argomento è una stringa vuota da sostituire alla parte riconosciuta.

Questa breve presentazione in questo sottocapitolo ha lo scopo di delineare al lettore un'idea delle difficoltà incontrate nel riconoscimento dei pattern ricorrenti nei rif. bib. non strutturati ma, come dichiarato precedentemente alla fine del secondo capitolo, la soluzione non sarà definitiva e totale perché è tendente all'impossibile il riuscire a identificare tutti i pattern non strutturati. Grazie ai test, possiamo affermare che i rif. bib. riconosciuti siano corretti. Per i rimanenti, il sistema software sviluppato offre la possibilità allo sviluppatore di creare facilmente PatternRecognizer da inserire successivamente per il riconoscimento di pattern. Oltre a queste considerazioni bisogna anche specificare che ci sono dei rif. bib. non strutturati che presentano errori di formattazione e che rendono il riconoscimento di pattern ancora più difficile. Di seguito abbiamo alcuni esempi di errori o di formati che compaiono nei documenti una sola volta senza alcuna apparente struttura ricorrente:

```
<ce:bib-reference id="bib12">
  <ce:label>[12]</ce:label>
  <ce:other-ref>
    <ce:textref>ADAMS/Engine documentation, MSC.ADAMS 2005 r2.</ce:textref>
```

```
  </ce:other-ref>
</ce:bib-reference>
...
<ce:bib-reference id="bib16">
  <ce:label>[16]</ce:label>
  <ce:other-ref>
    <ce:textref>AVL Boost user's guide, Version 4.0.3, July 2003.</ce:textref>
  </ce:other-ref>
</ce:bib-reference>
...
<ce:bib-reference id="BIB15">
    <ce:label>[15]</ce:label>
    <ce:other-ref>
        <ce:textref>Karypis C, Kumar V. A fast and high quality multilevel scheme for
            partitioning irregular graphs.  SIAM Journal on Scientific Computing,
            1998;20:359{392.</ce:textref>
    </ce:other-ref>
</ce:bib-reference>
...
<ce:bib-reference id="bib14">
  <ce:label>[14]</ce:label>
  <ce:other-ref>
    <ce:textref>Hagan MT, Demuth HB, Beale M. Neural network design. PWS 1996.</ce:textref>
  </ce:other-ref>
</ce:bib-reference>
...
<ce:bib-reference id="bib24">
  <ce:label>[24]</ce:label>
  <ce:other-ref>
    <ce:textref>The Workflow Management Coalition, Workflow Process Definition Interface { XML
        Process Definition Language, version 1.0, The Workflow Management Coalition
        2002.</ce:textref>
  </ce:other-ref>
</ce:bib-reference>
...
<ce:bib-reference id="bib3">
  <ce:label>[3]</ce:label>
  <ce:other-ref>
    <ce:textref>Creasy PN, Ellis, G.A Conceptual graph approach to conceptual schema
        integration. In: Proceedings of the ICCS'93, conceptual graphs for knowledge
        representation: first international conference on conceptual structures, Quebec,
        Canada; 1993.</ce:textref>
  </ce:other-ref>
</ce:bib-reference>
```

In questo ultimo esempio di rif. bib. abbiamo un errore nell'elenco dei nomi degli autori in corrispondenza di "Ellis, G.A". Probabilmente l'autore di cognome Ellis e con due nomi di iniziali "GA" sarebbe dovuto essere indicato con "Ellis GA.". Con questo esempio possiamo osservare come una singola virgola può invalidare il riconoscimento di pattern tramite espressioni regolari.

4.3 Output in RDF

Utilizzando le ontologie presentate nel capitolo 1.4, le informazioni estrapolate sono state codificate in RDF per i rif. bib. correttamente riconosciuti. Questa parte dello sviluppo è stata di una difficoltà moderata per il fatto che per ogni singola informazione doveva essere ricercata la proprietà o la classe

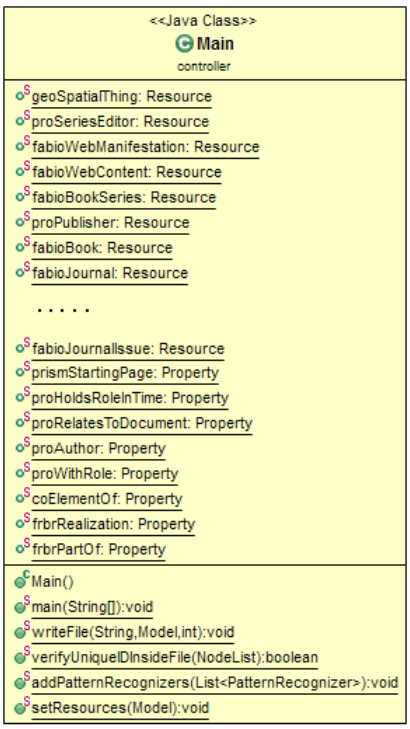

Figura 4.4: Classe Main con Resource e Property

adeguata nelle numerose ontologie a disposizione. In questa ricerca bisognava conoscere a grandi linee quale fosse lo scopo di ogni singola ontologia e ricercare coerentemente.

Le ontologie utilizzate che sono implementate nel codice di Jena sono soltanto FOAF, DC, RDF e RDF-S. Tutte le altre ontologie necessarie devono essere dichiarate manualmente. Il software richiede in ingresso un file `namespaces.xml` nel quale vengono dichiarati tutti i namespace necessari, vengono letti e utilizzati per dichiarare i `Resource` e i `Property` necessari alla generazione dei dataset. Nella figura 4.4 possiamo osservare come vengano dichiarati come attributi `static` della classe `Main` per permettere un facile utilizzo da parte delle altre classi, infatti è sufficiente riportare `Main.proPublisher` per ottenere l'URI della risorsa `pro:publisher`.

Un altro ostacolo maggiore che doveva essere superato con questo sistema software era la scelta degli URL per le varie risorse dichiarate negli statement RDF. Da un'analisi superficiale dei documenti XML è stato appurato che ogni documento contiene un solo articolo, il quale viene identificato univocamente dal proprio codice DOI (cap. 1.8) all'inizio del documento. Di fatto ogni documento contiene un elemento `coredata` contenente i metadati di rilievo, nel quale possiamo trovare il DOI dichiarato come stringa di testo dentro l'elemento:

```
<prism:doi>10.1016/j.websem.2003.07.007</prism:doi>
```

Questo identificativo viene utilizzato per dichiarare il resto delle risorse. È stato scelto per via della propria natura derivante dal DOI: persistente e raggiungibile via protocollo HTTP. L'unico svantaggio è che l'URL in sé non offre molte informazioni sulla natura dell'articolo.

I primi statement RDF in TURTLE risultanti da un articolo senza rif. bib. saranno:

```
@prefix dc:    <http://purl.org/dc/terms/> .
@prefix geo:   <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix frbr:  <http://purl.org/vocab/frbr/core#> .
@prefix fabio: <http://purl.org/spar/fabio> .
@prefix doco:  <http://purl.org/spar/doco/> .
@prefix biro:  <http://purl.org/spar/biro/> .
```

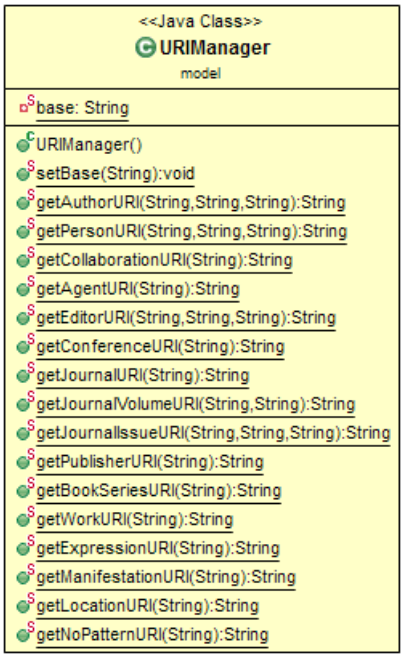

Figura 4.5: Classe responsabile della generazione degli URL delle risorse

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix co:    <http://purl.org/co/> .
@prefix pro:   <http://purl.org/spar/pro/> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix swc:   <http://data.semanticweb.org/ns/swc/swc_2009-05-09.html#> .
@prefix doi:   <http://dx.doi.org/> .
@prefix prism: <http://prismstandard.org/namespaces/basic/3.0/> .

<http://dx.doi.org/10.1016/j.websem.2003.07.007#ReferenceList>
        a            biro:ReferenceList ;
        frbr:partOf  <http://dx.doi.org/10.1016/j.websem.2003.07.007#Bibliography> .

<http://dx.doi.org/10.1016/j.websem.2003.07.007>
        a                 frbr:Work ;
        frbr:realization  <http://dx.doi.org/10.1016/j.websem.2003.07.007#Expression> .

<http://dx.doi.org/10.1016/j.websem.2003.07.007#Expression>
        <http://purl.org/dc/elements/1.1/title>
                "The National Cancer Institute's Thésaurus and Ontology"^^xsd:string .

<http://dx.doi.org/10.1016/j.websem.2003.07.007#Bibliography>
        a            doco:Bibliography ;
        frbr:partOf  <http://dx.doi.org/10.1016/j.websem.2003.07.007#Expression> .
```

Tutte le altre risorse relative ai rif. bib. saranno dichiarate e connesse attraverso statement RDF alle risorse sopra riportate.

Il passo successivo è l'inserimento attraverso i parametri in ingresso al software di un URL di base, ad es. `http://www.example.org/rdf` per identificare le risorse a cui i rif. bib. fanno riferimento.

Per ottenere l'URL delle risorse a partire dalla base URL inserita è stata sviluppata la classe `URIManager`, presentata in figura 4.5. La stringa `base` viene impostata con l'URL fornito con il parametro in ingresso. Utilizzando la classe `org.restlet.data.Reference`, le informazioni relative alla singola risorsa vengono aggiunte al proprio URL attraverso i parametri HTTP GET. La tipologia

della risorsa viene definita attraverso il parametro GET `type`. Di seguito abbiamo il codice sorgente di uno dei metodi:

```
public static String getPersonURI(String givenName, String familyName, String suffix){
  Reference ref = new Reference(base);
  ref.addQueryParameter("type", "person");
  ref.addQueryParameter("givenName", givenName);
  ref.addQueryParameter("familyName", familyName);
  ref.addQueryParameter("suffix", suffix);
  return ref.toString();
}
```

Attraverso questo metodo, una persona con nome “Sebastian”, cognome “Rudolph” e senza suffisso ottiene il seguente URL:

```
http://www.example.org/rdf?type=person&givenName=Sebastian&familyName=Rudolph&suffix=
```

I motivi di questa scelta implementativa sono molteplici:

1. le informazioni possono essere lette facilmente dall’URL: si riesce a comprendere che l’URL fa riferimento a una risorsa che indica una persona con nome, cognome e suffisso
2. questo metodo permette la codifica di caratteri non ASCII
3. un modulo esterno, ricevendo questa stringa, può facilmente estrarne le informazioni in quanto sono codificate utilizzando i parametri HTTP GET
4. il modulo può anche tradurre l’URL e reindirizzare l’utente a un altro URL che faccia riferimento diretto alla risorsa

Questa scelta di impostare gli URL è in linea con le raccomandazioni del Linked Open Data (cap. 1.6).

Capitolo 5

Conclusioni e risultati raggiunti

In conclusione, lo studio necessario allo sviluppo di questo software è stato adeguato ad ampliare la mia personale conoscenza sull'argomento e a consentirmi di comprendere i meccanismi principali dell'analisi di documenti XML e le meccaniche del Web Semantico.

I punti critici dello sviluppo del software sono stati soprattutto la comprensione dei documenti forniti e la configurazione degli strumenti utilizzati relativi all'ambiente di sviluppo.

I documenti forniti sono risultati molto problematici per i motivi indicati nel secondo capitolo. L'errore nel DTD fornito (fig. 2.1) mi ha impedito di sfruttare questo strumento cardine dell'analisi di documenti XML. Anche se fosse stato del tutto funzionante, il DTD sarebbe stato di poco aiuto perché è molto permissivo e, a un'attenta analisi, sarebbe anche errato dal punto di vista concettuale in alcuni casi. Come esempio abbiamo il seguente rif. bib. strutturato:

```
<ce:bib-reference id="b0305">
  <ce:label>[61]</ce:label>
  <sb:reference>
    <sb:contribution langtype="en">
      <sb:authors>
        <sb:author>
          <ce:given-name>Jeffrey D.</ce:given-name>
          <ce:surname>Ullman</ce:surname>
        </sb:author>
      </sb:authors>
      <sb:title>
        <sb:maintitle>Principles of Database and Knowledge Base Systems</sb:maintitle>
      </sb:title>
    </sb:contribution>
    <sb:host>
      <sb:book>
        <sb:date>1989</sb:date>
        <sb:publisher>
          <sb:name>Computer Science Press</sb:name>
        </sb:publisher>
      </sb:book>
    </sb:host>
  </sb:reference>
</ce:bib-reference>
```

Il titolo del libro è situato all'interno del nodo `sb:contribution` ma anche l'elemento `sb:book` prevede a sua volta un elemento che possa contenere un titolo. In questo caso il titolo riportato è il titolo del libro o il titolo di un articolo all'interno del libro senza titolo?

Un ragionamento logico ci porterebbe a considerare che il titolo contenuto all'interno dell'elemento `sb:contribution` è di un articolo contenuto all'interno del libro, e quest'ultimo dovrebbe avere le proprie informazioni collocate all'interno dell'elemento `sb:book`, fra le quali il proprio titolo.

Oltre a un DTD molto permissivo per i rif. bib. strutturati, abbiamo i rif. bib. non strutturati che danno esempio di massima eterogeneità e rendono difficoltosa la creazione di un insieme di recognizer limitato che riconosca gran parte dei pattern.

Il software sviluppato ha posto le basi per una corretta ed efficace analisi automatica dei rif. bib. dei documenti per consentirne la pubblicazione in RDF. Il codice sviluppato è stato commentato in inglese e comprende variabili nominate secondo una logica adatta al sistema per una facile modifica e aggiunta di ulteriori recognizer o regole.

Di seguito vengono riportati i risultati raggiunti:

```
Files processed: 20877
Bib. ref. found: 463748
Bib. ref. processed: 296850
```

Dei risultati sopra esposti bisogna considerare tutti gli elementi che sono stati scartati per una struttura di difficile comprensione o che comunque non presentavano le informazioni necessarie ad una corretta pubblicazione in RDF.

Gli strumenti descritti in questa elaborato per l'analisi dei documenti XML sono necessari e sufficienti all'analisi ed elaborazione di un qualunque documento XML allo scopo di estrapolarne le informazioni di maggior interesse.

Con il lavoro esposto in questo elaborato e i risultati ottenuti possiamo effettuare un insieme di rielaborazioni sui dati riconosciuti, ad es:

- eseguire query SPARQL sui dati per
  - ottenere il numero medio di rif. bib. nei documenti di una certa conferenza
  - ottenere la lista di tutti i lavori riferiti nelle conferenze di un certo anno
  - ottenere tutti i lavori riferiti di un certo autore
  - ecc.
- analisi di tipo bibliometrico
- test con altre fonti di riferimenti a opere e lavori intellettuali

Si ringraziano il Chiar.mo Prof. Paolo Ciancarini per il tempo, la pazienza e l'attenzione accordatami, e il Dr. Francesco Poggi per il suo aiuto nell'ottenere i documenti ed elaborare la soluzione qui esposta.

Appendice A

Sviluppo

Per lo sviluppo del sistema software sono stati necessari:

- ambiente di sviluppo utilizzato
  - Sistema Operativo: MS Windows 7
  - Sistema HW: Inter Core I5 - 4GB RAM - HDD 500GB - ATI HD 5470
  - IDE: Eclipse 5.0
- numero LOC prodotte (inclusi i commenti): 1610
- valutazione del tempo speso: 90 ore-uomo per lo studio del materiale necessario, dei documenti forniti e per l'elaborazione della soluzione, 110 ore-uomo per lo sviluppo del codice, debugging e testing

Bibliografia

[1] Paolo Ciccarese and Tim Clark. FaBiO, the FRBR-aligned Bibliographic Ontology. `http://purl.org/spar/fabio/`, 2014.

[2] Paolo Ciccarese and Silvio Peroni. Essential FRBR in owl2 dl. `http://purl.org/spar/frbr`, 2011.

[3] Tommaso di Noia, Roberto De Virgilio, Eugenio Di Sciascio, and Francesco M Donini. *Semantic Web. Tra Ontologie e Open Data.* APOGEO, 2013.

[4] David Shotton and Silvio Peroni. Biro, the Bibliographic Reference Ontology. `http://purl.org/spar/biro`, 2013.

[5] David Shotton and Silvio Peroni. PRO, the Publishing Roles Ontology. `http://purl.org/spar/pro`, 2013.