Finding similar papers using ontologies

Francesco Gaetano, Luigi Lomasto, Marco Mecchia, Andrea Soldá

Gennaio 2016

1 Introduzione al problema

Il problema di trovare lavori scientifici simili scritti in linguaggio naturale é un compito molto difficile dal punto di vista informatico: Tali documenti infatti non hanno una struttura fissa, e sono pieni di elementi non facilmente confrontabili come formule, notazioni ed immagini. Inoltre, nonostante una netta predominanza dell'inglese, i documenti sono scritti in lingue diverse. Tutte queste caratteristiche, insite in lavori di ricerca di questo tipo, rendono gli approcci basati sul confronto testuale non utilizzabili. Il nostro lavoro, basato sulle meta informazioni dei documenti e sull'utilizzo di tecnologie del Semantic Web, mira a fornire un approccio alternativo ai metodi tradizionali, nonché un'infrastruttura riutilizzabile anche in altri ambiti.

Il resto del lavoro é organizzato come segue: nella sezione 2 viene presentata l'analisi da noi condotta ed i vari step che hanno portato al risultato finale. Nella sezione 3 vengono analizzati nel dettaglio le tecnologie del Web Semantico da noi utilizzate. Nella sezione 4 vengono presentate nel dettaglio le varie componenti introdotte nella sezione 2, illustrando e commentando estratti di codice del progetto. Infine, nella sezione 5 verranno commentati i risultati ottenuti ed eventuali applicazioni ed estensioni di quanto fatto.

2 Workflow

Il lavoro é stato suddiviso in diverse fasi.

2.1 Selezione del database ed estrazione delle meta-informazioni

In questa fase preliminare del lavoro, abbiamo costruito il dataset sul quale basarci per tutto il resto del lavoro. Per prima cosa abbiamo scelto il database dal quale attingere i lavori da confrontare. La nostra scelta e' ricaduta su DBLP[**?**] in quanto punto di riferimento centrale per la sottomissione dei paper nella comunita' informatica. DBLP inoltre mette a disposizione ampi dataset di meta-informazioni sugli articoli scaricabili in formato standard xml, con relativi url alle pagine web degli articoli. Mediante gli url letti dal

file dblp.xml, per ogni articolo abbiamo estratto dalla pagina sorgente l'abstract ed eventualmente topics e keywords (se presenti). Questo lavoro ci ha permesso di ottenere un primo dataset che, oltre alle informazioni di partenza, contiene anche gli abstract con eventuali topics o keywords. Poiché topics e keywords sono state estratte dalla pagina sorgente dell'articolo, é stato deciso di assegnare, per ogni topic una relevance pari ad 1, e per ogni keyword una relevance pari a 0.99.

A partire da questa prima versione, con l'utilizzo delle API di Alchemy abbiamo, per ogni abstract, estratto le keywords con le rispettive relevance, ottenendo cosí il dataset finale, sostituendo all'abstract le keywords ottenute. Il dataset finale contiene per ogni articolo le seguenti informazioni:

- Titolo del lavoro
- Autori del lavoro
- Anno di pubblicazione
- Topics (se presenti)
- Keyword (se presenti)
- Rivista
- URL

2.2 Progettazione e popolamento dell'ontologia

In questa fase, é stata studiata la progettazione di un'ontologia adatta a gestire le informazioni estratte nella fase precedente. Il passaggio ad un'ontologia é stato necessario per almeno due motivi:

1. La possibilitá di interrogare il database di meta informazioni tramite query semantiche.
2. La possibilitá di collegare il lavoro a strumenti giá esistenti per i *linked data*, in modo da rendere lo strumento integrabile.

Per l'ontologia, la nostra scelta é ricaduta su CIDOC/CRM[**?**]. Il modello concettuale CIDOC/CRM é un stato progettato per la gestione di contenuti relativi alla storia ed alle opere d'arte, quindi si é rivelato adatto al nostro scopo.

2.3 Interrogazione dell'ontologia

Una volta popolata l'ontologia é stata necessaria la progettazioni di query adatte al contesto del progetto. Le query su cui é stata dedicata maggiore attenzione sono due:

1. A partire dal titolo di un articolo, restituire keywords e topics.
2. A partire da keywords e topic ottenuti dalla query precedente, restituire la lista degli articoli che hanno un'sottoinsieme di keywords e topics in comune.
3. A partire dal titolo di un articolo, restituire tutte le informazioni quali: Autori, anno di pubblicazione, rivista, url ...

2.4 Presentazione dei risultati

Una volta progettate ed eseguite le query, abbiamo studiato un modo per poter proporre i risultati in modo elegante, ma che allo stesso tempo ponesse enfasi sullo strato semantico che lega i documenti. Per fare ció, abbiamo generato in maniera ricorsiva un grafo centralizzato: la radice é il documento di partenza, ed il solo nodo presente nel grafo. Il livello $i + 1$-esimo del grafo viene generato semplicemente lanciando la query principale su tutti gli articoli del livello i-esimo. Il processo viene reiterato finché non si arriva alla profonditá desiderata.

3 Strumenti utilizzati

Le tecnologie utilizzate per lo sviluppo di questo progetto sono molteplici:

- Java 8 per gran parte del backend, cioé lo scraping e la popolazione dell'ontologia. Abbiamo usato le seguenti librerie:
  - jsoup per l'utilizzo di espressioni Xpath nella fase di scraping.
  - Apache Jena per la popolazione dell'ontologia e la creazione del file .owl.
  - AlchemyAPI per l'estrazione delle keywords da ogni topic.
- Abbiamo utilizzato le seguenti tecnologie del Semantic Web:
  - Protege per creare ed estendere lo schema ontologico.
  - OWL come linguaggio per definire l'ontologia.
  - SPARQL come linguaggio di query per interrogare l'ontologia.
  - Apache Fuseki come server per gestire le query.
- PHP7 per la formulazione e la sottomissione delle query lato server.

- Javascript per la parte di frontend, utilizzando le seguenti librerie:
  - vis.js per il rendering del grafo.
  - JQuery per gestire meglio le richieste ajax agli script Php lato server.
  - Bootstrap per la gestione dell'aspetto della pagina.

3.1 Jsoup

Libreria Java che permette di lavorare con documenti HTML. Fornisce delle API molto semplici con le quali é possibile estrarre e manipolare i dati a partire dal DOM di una pagina mediante l'uso di espressioni XPath.

Esempio

```
Document doc = Jsoup.connect(URL).timeout(50*1000).get
    ();
Elements elemsAbstract = doc.select("p.Para");
```

3.2 Protégé

Protégé é un framework open source sviluppato presso l'universitá di Stanford, esso dispone di un'interfaccia grafica per la definizione di schemi ontologici e della semantica alla base di essi. Tramite il tool é stato possibile importare lo schema ontologico tipico di CIDOC-CRM sulla quale successivamente si é reso necessario la definizione di classi e propriet supplementari per estendere lo schema in uno maggiormente adatto per la nostra applicazione.

3.3 Apache Jena

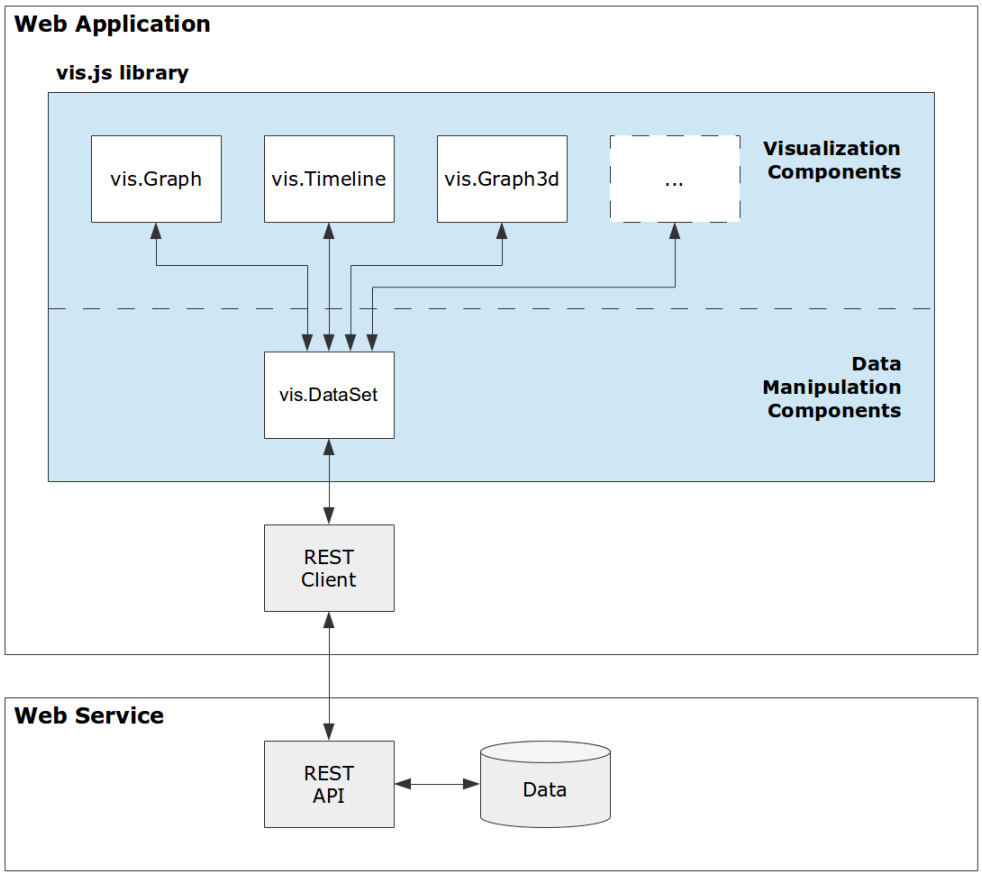
Apache Jena un framework open source per Java, il quale fornisce API per la lettura e scrittura di grafi RDF. In particolare, é stato utilizzato per la lettura dello schema precedentemente realizzato con Protégé e per la successiva scrittura di uno schema OWL comprendente le istanze a partire da un dataset in input.

3.4 AlchemyAPI

AlchemyAPI utilizza algoritmi per l'apprendimento automatico che permettono di estrarre meta-dati semantici dal contenuto desiderato, come ad esempio informazioni su persone, luoghi, aziende, gli argomenti, i fatti, le relazioni, gli autori e le lingue. I meta-dati possono essere restituiti in formato XML, JSON e RDF.
Ad ogni parola estratta viene associato una relevance (valore numerico compreso tra 0 e 1), che indica l'incidenza della parola all'interno del testo. Nel nostro caso sono state usate parole con relevance maggiore o uguale di 0.6.

3.5 Vis.js

Vis.js é una libreria javascript che permette la visualizzazione dinamica di grafi, timeline e grafici a due o tre dimensioni. La struttura architetturale é illustrata nell'immagine seguente.



Nel nostro caso abbiamo usato tale libreria per generare un grafo contenete gli articoli con le relative relazioni semantiche. Un grafo in Vis.js é una struttura costituita da nodi ed archi. Di seguito é riportato un esempio con cinque nodi e quattro archi.

```
<!doctype html>
<html>
<head>
  <title>Network | Basic usage</title>

  <script type="text/javascript" src="../../dist/vis.
      js"></script>
</head>
```

```
<body>

<div id="mynetwork"></div>

<script type="text/javascript">
  // create an array with nodes
  var nodes = [
    {id: 1, label: 'Node 1'},
    {id: 2, label: 'Node 2'},
    {id: 3, label: 'Node 3'},
    {id: 4, label: 'Node 4'},
    {id: 5, label: 'Node 5'}
  ];

  // create an array with edges
  var edges = [
    {from: 1, to: 2},
    {from: 1, to: 3},
    {from: 2, to: 4},
    {from: 2, to: 5}
  ];

  // create a network
  var container = document.getElementById('mynetwork')
     ;
  var data= {
    nodes: nodes,
    edges: edges,
  };
  var options = {
    width: '400px',
    height: '400px'
  };
  var network = new vis.Network(container, data,
     options);
</script>

</body>
</html>
```

4 Implementazione

4.1 Costruzione del dataset

Per costruire il dataset di meta-informazioni dei documenti presenti su DBLP, é stato sviluppato un package java chiamato *scraper*, costituito dalle seguenti classi:

.java Contenente il main. Si occupa, preso in input il dataset di meta-informazioni degli articoli scaricabile da DBLP, di eliminafare le informazioni superflue e di invocare le classi di scraping quando si trova l'elemento ¡url¿.

SuperScraper.java Superclasse astratta degli scraper, utile per il polimorfismo.

FactoryScraper.java Implementazione del Factory Method per gli scraper.

IJIEMScraper.java Istanza di SuperScraper.

JDisplaScraper.java Istanza di SuperScraper.

JkdbScraper.java Istanza di SuperScraper.

StandardScraper.java Istanza di SuperScraper.

Count

```
1  public class Count {
2    public static void main(String [] args) throws
         FileNotFoundException, IOException {
3      BufferedReader reader = new BufferedReader(new FileReader(
           args[1]));
4      FileWriter w = new FileWriter(args[2]);
5
6      parsing(reader,line,w);
7      w.close();
8      reader.close();
9    }
10
11   private static void parsing(BufferedReader reader, FileWriter
       w) throws IOException {
12     int nArticoli=0;
13     SuperScraper scraper;
14     FactoryScraper f = new FactoryScraper();
15
16     do{
17       line = reader.readLine();
18       if(line.contains("<article")){
19         ++nArticoli;
20       }
21       else
```

```
22          if(line.contains("<ee>")){
23            scraper = f.createScraper(line);
24            scraper.scrape(w, line);
25          }
26        if(GENERATE)
27          w.write(line+"\n");
28      }while(line!=null && nArticoli<=5000);
29      w.flush();
30      System.out.println(nArticoli);
31    }
32  }
```

Listing 1: Count.java

Il main prende due argomenti da linea di comando: il primo corrisponde al percorso del file xml di informazioni di DBLP, ed il secondo al nome del file dove ricopiare il dataset aggiornato. Il cuore della classe é costituito dal metodo statico parsing: Tale metodo legge il file XML riga per riga, fino a trovare la riga contentente l'URL dell'articolo; una volta trovato, la factory crea uno scraper apposito in base al contenuto della linea. Lo scraper si occupa di effettuare la connessione all'url, di estrarre l'abstract dalla pagina web e di aggiornare il file di output. Grazie al Factory Method ed al polimorfismo, aggiungere nuovi scraper e' semplicissimo, e non richiede l'intervento diretto sul main. Il numero di articoli é stato impostato a 5000 poiché tale numero si é rivelato sufficiente per costruire il nostro dataset.

StandardScraper La classe StandardScraper é un'istanza della superclasse SuperScraper, e si occupa di scrivere in output le informazioni estratte da un articolo in un file xml.

```
1  public class StandardScraper extends SuperScraper {
2
3    @Override
4    public void scrape(FileWriter w, String line) throws
         IOException {
5      if(GENERATE)
6        w.write("<abstract>\n");
7
8      String URL[]=line.split("<ee>");
9      String URL2[]=URL[1].split("</ee>");
10     try{
11       Document page=returnPage(URL2[0]);
12       Elements elemsAbstract = page.select("p.Para");
13       for(Element elem: elemsAbstract){
14         if(DEBUG)
15           System.out.println(elem.text().toString());
16         if(GENERATE)
17           w.write(elem.text().toString());
18       }
19       if(GENERATE)
20         w.write("\n</abstract>\n");
```

```
21          Elements elemsKeyWord = page.select("ul.abstract-about-
                subject > li > a");
22          for(Element elem: elemsKeyWord){
23            if(DEBUG)
24              System.out.println(elem.text().toString());
25            if(GENERATE)
26              w.write("<topic>");
27            w.write(elem.text().toString());
28            w.write("</topic>\n");
29          }
30
31        }
32        catch(ConnectException e){ w.write("\n</abstract>\n");
              failureConnect++;if(DEBUGException){System.out.print("
              Connessione rifiutata "+URL2[0]+" ");System.out.println(
              failureConnect);}}
33        catch(HttpStatusException e){w.write("\n</abstract>\n");
              failureConnect++;if(DEBUGException){System.out.print("
              Status=404 "+URL2[0]+" ");System.out.println(
              failureConnect);}}
34        catch(SocketTimeoutException e){w.write("\n</abstract>\n");
              failureConnect++;if(DEBUGException){System.out.print("
              Socket timeout "+URL2[0]+" ");System.out.println(
              failureConnect);}}
35        catch(UnknownHostException e){w.write("\n</abstract>\n");
              failureConnect++;if(DEBUGException){System.out.print("dx.
              doi.org "+URL2[0]+" ");System.out.println(failureConnect)
              ;}}
36        catch(UnsupportedMimeTypeException e){w.write("\n</abstract
              >\n"); failureConnect++;if(DEBUGException){System.out.
              print("jsoup "+URL2[0]+" ");System.out.println(
              failureConnect);}}
37
38      }
39
40  }
```

Listing 2: StandardScraper.java

Questo tipo di output e' lo stesso per tutte le istanze degli scraper. Quello che differenzia ogni scraper é la struttura della pagina che vanno ad esaminare, per cui sono richiesti controlli ed espressioni Xpath diverse per estrarre le informazioni giuste. Se il documento contiene delle keyword, esse vengono aggiunte in un apposito tag dopo i topic, prima della chiusura del tag relativo all'articolo.

```
<article mdate="2011-01-11" key="journals/acta/Saxena96">
<author>Sanjeev Saxena</author>
<title>Parallel Integer Sorting and Simulation Amongst CRCW
    Models.</title>
<pages>607-619</pages>
<year>1996</year>
```

```
<volume>33</volume>
<journal>Acta Inf.</journal>
<number>7</number>
<url>db/journals/acta/acta33.html#Saxena96</url>
<abstract>In this paper a general technique for reducing
    processors in simulation without any increase in time is
    described.</abstract>
<topic>Logics and Meanings of Programs</topic>
<topic>Computer Systems Organization and Communication
    Networks</topic>
<topic>Software Engineering/Programming and Operating Systems<
    /topic>
<topic>Data Structures, Cryptology and Information Theory</
    topic>
<topic>Theory of Computation</topic>
<topic>Information Systems and Communication Service</topic>
<ee>http://dx.doi.org/10.1007/BF03036466</ee>
</article>
```

Listing 3: Esempio di xml prodotto dallo StandardScraper

FactoryScraper

```
1  public class FactoryScraper {
2
3    private SuperScraper jdispla;
4    private SuperScraper jkdb;
5    private SuperScraper ijiem;
6    private SuperScraper standard;
7
8    public FactoryScraper(){
9      jdispla = new JDisplaScraper();
10     jkdb = new JkdbScraper();
11     ijiem = new IJIEMScraper();
12     standard = new StandardScraper();
13   }
14
15   public SuperScraper createScraper(String line){
16
17     if(line.contains("<ee>")){
18       if(line.contains("j.displa.") || line.contains("j.compind"
             )){
19         return jdispla;
20       }
21       else{
22         if(line.contains("jkdb")){
23           return jkdb;
24         }else if(line.contains("IJIEM.")){
25           return ijiem;
26         }
```

```
27          else{
28            return standard;
29          }
30        }
31      }
32      return null;
33    }
34  }
```

Listing 4: FactoryScraper.java

La FactoryScraper alla sua creazione crea un tipo di oggetto per ogni specializzazione della classe SuperScraper; in questo modo, quando c'é bisogno di fare il parsing di un nuovo documento non si crea ogni volta un nuovo oggetto, ma si utilizza sempre lo stesso. Cosí facendo, gli oggetti vengono riutilizzati e vengono risparmiati memoria e processore, in quanto la Garbage Collector di Java non deve essere invocata di continuo. Il metodo createScraper si deve solamente occupare di verificare le condizioni per cui deve essere creato un tipo di Scraper piuttosto che un altro.

4.2 Estrazione delle Keyword

Per l'estrazione delle keyword a partire dall'abstract é stato fatto uso delle API messe a disposizione dal software Alchemy[**?**], scaricabili gratuitamente. Tale software é composto da molti strumenti utili per l'analisi linguistica, tra cui l'estrazione di parole chiave da un testo con rilevanze normalizzate nell'intervallo $[0, 1]$. Gli unici limiti riscontrati sono stati il fatto di doversi registrare per ottenere una chiave per utilizzare le API ed il relativo limite di chiamate giornaliere.

Il pacchetto *keyword* sviluppato nel progetto é composto da due classi:

KeywordExtractor é una classe di wrapper per la chiamata al metodo di AlchemyAPI che dato un testo restituisce le keyword rankate.

ScraperForKeyword é la classe contentente il metodo main del pacchetto. Si occupa di aggiungere al dataset generato con il pacchetto scraper le keyword estratte con la classe KeywordExtractor.

ScraperForKeyword

```
captioncaption
public class scraperForKeyword {
  private static String getStringFromDocument(Document doc)
      throws IOException {
    try {
      DOMSource domSource = new DOMSource(doc);
      StringWriter writer = new StringWriter();
      StreamResult result = new StreamResult(writer);
      String toReturn="";
      TransformerFactory tf = TransformerFactory.newInstance();
```

```
10        Transformer transformer = tf.newTransformer();
11        transformer.transform(domSource, result);
12
13        String [] split=writer.toString().split("<keyword>");
14
15        for(int i=1;i<split.length;i++){
16          String []relevance1=split[i].split("<relevance>");
17          String []relevance2=relevance1[1].split("</relevance>");
18          String []text1=split[i].split("<text>");
19          String []text2=text1[1].split("</text>");
20
21          double relevance = Double.parseDouble(relevance2[0]);
22          if(relevance*10 >= 6){
23            toReturn+="<keyword>\n";
24            toReturn+="<relevance>"+relevance+"</relevance>\n";
25            toReturn+="<text>"+text2[0]+"</text>\n";
26            toReturn+="</keyword>\n";
27          }
28        }
29
30        return toReturn;
31      } catch (TransformerException ex) {
32        ex.printStackTrace();
33        return null;
34      }
35    }
36
37    public static void main(String [] args) throws IOException,
          XPathExpressionException, SAXException,
          ParserConfigurationException{
38      int nArticoli = Integer.parseInt(args[0]);
39      int numeroRigheLette;
40      int articoliGialetti=nArticoli;
41      int articoliDaLeggere=Integer.parseInt(args[1]);
42      boolean abstractB=false;
43      BufferedReader reader = new BufferedReader(new FileReader(
            args[2]));
44      String line = reader.readLine();
45      String abstract_Text="";
46      FileWriter w=new FileWriter(args[3]);
47      String api_path = args[4];
48      int counter=0;
49
50      while(counter<=nArticoli){
51        line = reader.readLine();
52        if(line.contains("<article"))
53          ++counter;
54      }
55      while(line!=null && nArticoli<articoliDaLeggere+
            articoliGialetti) {
56        if(line.contains("<article")){
57          ++nArticoli;
58          if(nArticoli<articoliDaLeggere+articoliGialetti){
59            w.write(line+"\n");}
```

```
60            abstractB=false;
61          }
62          else
63            if(line.contains("<abstract>")){
64              abstract_Text+=line;
65              abstractB=true;
66
67            }
68            else if(abstractB && line.contains("</abstract>")){
69              abstractB=false;
70              abstract_Text=abstract_Text.replace("<abstract>","");
71              abstract_Text=abstract_Text.replace("</abstract>","");
72              if(!abstract_Text.equals("")){
73                try{
74                  Document s=KeywordExtractor.extractKeyword(
                        abstract_Text, api_path);
75                  String keywordDocument = getStringFromDocument(s);
76                  w.write(keywordDocument);
77                }
78                catch(IOException e){
79                  System.out.println("Error making API call:
                        unsupported-text-language.");
80                }
81              }
82              abstract_Text="";
83            }
84            else if(abstractB){
85              abstract_Text+=line;
86            }
87            else
88              w.write(line+"\n");
89          line = reader.readLine();
90        }
91        w.close();
92        System.out.println("#Articoli letti: "+ --nArticoli);
93    }
94  }
```

./src/keywords/scraperForKeyword.java

Il main prende in input 5 argomenti: il numero di articoli giá letti, il numero di articoli da leggere, il percorso del file di input[1], il percorso del file di output ed il percorso della chiave per utilizzare le API di Alchemy. Il numero di articoli giá letti e quelli da leggere sono parametri necessari introdotti dal limite di chiamate di AlchemyAPI; in questo modo il lavoro é stato diviso tra i componenti del team in maniera facilmente ricostruibile.

Dopo aver saltato gli articoli giá letti, il parser per ogni articolo trova l'abstract ed esegue una chiamata al metodo statico ExtractKeyword della classe KeywordExtractor. Grazie al metodo di utilitá getStringFromDocument, il documento xml contenente le keyword dell'abstract viene serializ-

[1]Nel file ci sono giá gli abstract, estratti con il pacchetto scraper.

zato. La stringa corrispondente al documento serializzato a questo punto viene concatenata alle informazioni del documento e viene quindi scritta in output.

KeywordExtractor La classe é composta dal solo metodo statico ExtractKeyword. Tale metodo prende in input il testo da cui estrarre le parole chiave ed il percorso del file contenente la chiave fornita dal sito web di Alchemy. Il metodo restituisce un documento xml sotto forma di oggetto Document; in questo modo é facilmente serializzabile, oltre che navigabile con gli strumenti del DOM.

```
1 public class KeywordExtractor {
2
3   public static Document extractKeyword(String text, String
        api_path) throws IOException, XPathExpressionException,
        SAXException, ParserConfigurationException {
4     AlchemyAPI alchemyObj = AlchemyAPI.GetInstanceFromFile(
          api_path);
5     return alchemyObj.TextGetRankedKeywords(text);
6   }
7
8 }
```

Listing 5: KeywordExtractor.java

```
<keywords>
<keyword>
<relevance>0.986122</relevance>
<text>tree schemas</text>
</keyword>
<keyword>
<relevance>0.868575</relevance>
<text>NP-complete problems</text>
</keyword>
<keyword>
<relevance>0.830641</relevance>
<text>certain NP-complete problems</text>
</keyword>
</keywords>
```

Listing 6: Esempio di documento restituito dalla classe KeywordExtractor

4.3 Lo schema ontologico

Una volta realizzato il dataset con le relative informazioni, il passo successivo é stato quello di definire la semantica sottostante. L'idea é stata quella di partire da uno schema esistente che fosse flessibile per avere l'opportunitá di estenderlo nel caso fosse stata necessaria l'aggiunta di ulteriori entitá.

4.3.1 Conceptual Reference Model (CRM);

Il modello utilizzato é il CIDOC-CRM, usato da musei ed altre istuzioni culturali per la descrizione di relazioni tra entitá per valorizzare lo scambio di informazioni tra fonti eterogenee.

Per definizione, un'onotologia é CRM compatibile se rispetta il seguente schema proposto dagli autori:

E1 CRM Entity
E2 - Temporal Entity
E4 - - Period
E5 - - - Event
E7 - - - - Activity
E11 - - - - - Modification
E12 - - - - - - Production
E13 - - - - - Attribute Assignment
E65 - - - - - Creation
E63 - - - - Beginning of Existence
E12 - - - - - *Production*
E65 - - - - - Creation
E64 - - - - End of Existence
E77 - Persistent Item
E70 - - Thing
E72 - - - Legal Object
E18 - - - - Physical Thing
E24 - - - - - Physical Man-Made Thing
E90 - - - - Symbolic Object
E71 - - - Man-Made Thing
E24 - - - - *Physical Man-Made Thing*
E28 - - - - Conceptual Object
E89 - - - - - Propositional Object
E30 - - - - - - Right
E73 - - - - - - Information Object
E90 - - - - - *Symbolic Object*
E41 - - - - - - Appellation
E73 - - - - - - *Information Object*
E55 - - - - - Type
E39 - - Actor
E74 - - - Group
E52 - Time-Span
E53 - Place
E54 - Dimension
E59 Primitive Value
E61 - Time Primitive
E62 - String

Figura 1: Schema CRM ridotto

Lo schema di base attualmente comprende 93 classi e 165 proprietá, delle quali solo un sottoinsieme sono risultate funzionali al nostro obiettivo:

- ***Entitá***

    1. **E12_Production:** In CRM un oggetto reale é opportuno considerarlo come il risultato di un'attivitá (in questo caso l'attivitá é descritta come produzione, infatti, nello schema viene definita come sottoclasse dell'entitá E7_Activity). Nel rispetto di questo, abbiamo ritenuto opportuno associare ad ogni articolo una produzione della quale esso é il risultato finale.

2. **E31_Document:** Questa classe identifica oggetti tramite i quali si effettuano affermazioni sulla realtá tramite testo, immagini o altri mezzi visivi. Viene definito come sottoclasse di - E73_Information Object.
3. **E35_Title:** Questa classe comprende i nomi che identificano un qualsiasi lavoro.
4. **E52_Time-Span:** Questa classe descrive l'arco temporale durante il quale una generica attivitá ha avuto luogo.

- ***Object Properties***

1. **P14_carried_out_by:** Descrive la partecipazione di un generico attore ad una attivitá. Nel nostro caso il dominio é rappresentato dall'entitá E12_Production e il range all'entitá Author.
2. **P108_has_produced:** Identifica la realizzazione di un generico oggetto fisico (prodotto dall'uomo) in seguito ad un'attivitá. Nel nostro caso il dominio é rappresentato dall'entitá E12_Production e il range dall'entitá E31_Document.
3. **P4_has_time-span:** Descrive l'arco di tempo di una qualsiasi entitá temporale (eg. attivitá). Nel nostro caso il dominio é rappresentato dall'entitá E12_Production e il range dall'entitá E52 Time-span.
4. **P102_has_title:** Associa istanze dell'entitá titolo ad un generico oggetto fisico. Nel nostro caso il dominio é rappresentato dall'entitá E31_Document e il range dall'entitá E35_Title.
5. **P56_bears_feature:** Associa un'istanza di un generico oggetto fisico ad istanze di E26_Physical_Feature (classe che comprende le caratteristiche e gli attributi di un generico oggetto). Nel nostro caso il dominio é rappresentato dall'entitá E31_Document e il range dall'entitá Topic.
6. **P149_is_identified_by:** Associa istanze di un generico oggetto ad istanze di E75_Conceptual_Object_Appellation (classe che comprende appellativi per un generico oggetto). Nel nostro caso il dominio é rappresentato dall'entitá E31_Document e il range dall'entitá Keyword.
7. **P67_refers_to:** Proprietá molto generica. Collega una qualsiasi entitá CRM ad istanze di E89_Propositional_Object (classe che comprende astrazioni di oggetti non materiali). Nel nostro caso il dominio é rappresentato dall'entitá E31_Document e il range dall'entitá Url.

Si nota come alcune classi citate si differenziano dalla notazione tipica di CRM (lettera E o P seguita da un numero), esse rappresentano le classi aggiunte allo schema di base.

4.3.2 Estensioni al CRM

Essendo un modello principalmente progettato per le istituzioni culturali, per rappresentare in modo corretto le informazioni estratte da DBLP, é stato necessario aggiungere allo schema di base del CRM, altre classi e proprietá che elencheremo di seguito con le opportune motivazioni.

- ***Entitá***

    1. **Author:** Definita come sottoclasse di E21_Person e della piú generica entitá E39_Actor, rappresenta l'insieme degli autori dei vari articoli. É stato necessario creare questa classe per poterla collegare, tramite la proprietá CRM P14_carried_out_by, all'attivitá produzione, come descritto in precedenza.
    2. **Topic:** Definita come specializzazione di un generico oggetto concettuale ed informativo (sottoclasse di E73_information Object). Essa descrive l'insieme dei topic di ogni articolo, é stata aggiunta in quanto lo schema di base non prevedeva questo tipo di concetto. Istanze di questa classe sono connesse alle istanze della classe E31_Document tramite la proprietá P56_bears_feature.
    3. **Keyword:** Come la classe Topic, anche la classe Keyword é stata definita come sottoclasse dei generici oggetti concettuali ed informativi, descrive l'insieme delle parole chiave associate ad ogni articolo. É stata aggiunta in quanto lo schema di base non prevedeva questo tipo di concetto. Istanze di questa classe sono connesse alle istanze della classe E31_Document tramite la proprietá CRM P149_is_identifiied_by.
    4. **Url:** Concettualmente definito come le classi Topic e Keyword, rappresenta l'indirizzo Web della risorsa documento, non previsto nello schema CRM di base. Ogni istanza della classe E31_Document é connessa ad un'istanza di questa classe tramite la proprietá CRM P67_refers_to.
    5. **Journal:** Concettualmente inserito come oggetto informativo al pari della classe E31_Document. Anche in questo caso CRM non prevedeva questo tipo di concetto. Ogni istanza della classe E31_Document é connessa ad un'istanza di questa classe tramite la propriet published_by.

- ***Object Properties***

    1. **published_by**: Associa un'istanza della classe E31_Document ad un'istanza della classe Journal. É stato necessario aggiungere questa proprietá in quanto lo schema CRM non definiva nessuna proprietá che collegasse due entitá E73_Information_Object (o piú genericamente, due entitá E71_Man-Made_Thing).

- ***Data Properties***

  1. **Anno**: Associa un range di tipo int all'entitá E52_Time-span. Usata per visualizzare l'anno di pubblicazione degli articoli.
  2. **Journal_value**: Associa un range di tipo string all'entitá Journal. Usata per visualizzare il nome della riviste.
  3. **Name**: Associa un range di tipo string all'entitá Author. Usata per visualizzare il nome degli autori.
  4. **Relevance**: Associa un range di tipo double all'entitá Keyword. Usata per visualizzare la rilevanza delle parole chiave.
  5. **Text**: Associa un range di tipo string all'entitá Keyword. Usata per visualizzare il nome delle parole chiave.
  6. **Title_value**: Associa un range di tipo string all'entitá E35_Title. Usata per visualizzare il titolo degli articoli.
  7. **Topic_value**: Associa un range di tipo string all'entitá Topic. Usata per visualizzare i topic di un articolo.
  8. **Url_value**: Associa un range di tipo string all'entitá Url. Usata per visualizzare l'indirizzo Web della risorsa associata all'articolo.

4.4 Popolamento dell'ontologia

Una volta definito lo schema completo delle entitá e delle relazioni semantiche tra di esse, il passo successivo consisteva nella generazione delle istanze. É stata dunque implementata una classe java funzionale allo scopo, utilizzando le API messe a disposizione dal framework Apache Jena. Di seguito mostreremo le parti principali del codice con le relative spiegazioni:

La prima fase consiste nell'import di tutte le classi e le proprietá che verranno successivamente utilizzate. Si nota che i namespace utilizzati sono due: uno relativo alle classi del modello CRM, l'altro invece relativo alle classi che abbiamo aggiunto.

```
1  public static void main(String[] args) throws IOException,
       UnsupportedEncodingException {
2
3    FileWriter out = new FileWriter("output/docs_ontology.owl");
4    readExternalModel(model);
5    int count=0;
6
7    /*Select classes*/
8    OntClass author = model.getOntClass(NS+"Author");
9    OntClass production = model.getOntClass(CRM_NS+"
         E12_Production");
10   OntClass e_document = model.getOntClass(CRM_NS+"E31_Document
         ");
11   OntClass time_span = model.getOntClass(CRM_NS+"E52_Time-Span
         ");
```

```
12      OntClass title = model.getOntClass(CRM_NS+"E35_Title");
13      OntClass topic = model.getOntClass(NS+"Topic");
14      OntClass keyword = model.getOntClass(NS+"Keyword");
15      OntClass e_url = model.getOntClass(NS+"Url");
16      OntClass e_journal = model.getOntClass(NS+"Journal");
17
18
19      /*Select properties*/
20      OntProperty carried = model.getOntProperty(CRM_NS+"
            P14_carried_out_by");
21      OntProperty hasProduced = model.getOntProperty(CRM_NS+"
            P108_has_produced");
22      OntProperty has_time_span = model.getOntProperty(CRM_NS+"
            P4_has_time-span");
23      OntProperty hasTitle = model.getOntProperty(CRM_NS+"
            P102_has_title");
24      OntProperty name = model.getOntProperty(NS+"name");
25      OntProperty bears_features = model.getOntProperty(CRM_NS+"
            P56_bears_feature");
26      OntProperty is_identified_by = model.getOntProperty(CRM_NS+"
            P149_is_identified_by");
27      OntProperty hasText = model.getOntProperty(NS+"Text");
28      OntProperty hasRelevance = model.getOntProperty(NS+"
            Relevance");
29      OntProperty hasAnno = model.getOntProperty(NS+"Anno");
30      OntProperty title_value = model.getOntProperty(NS+"
            Title_value");
31      OntProperty topic_value = model.getOntProperty(NS+"
            Topic_value");
32      OntProperty journal_value = model.getOntProperty(NS+"
            Journal_value");
33      OntProperty url_value = model.getOntProperty(NS+"Url_value")
            ;
34      OntProperty published_by = model.getObjectProperty(NS+"
            published_by");
35      OntProperty refers_to = model.getOntProperty(CRM_NS+"
            P67_refers_to");
```

Listing 7: JenaModel.java

Prima di tutto viene richiamata la seguente funzione:

```
1   private static void readExternalModel(OntModel model){
2     try {
3       FileInputStream fin = new FileInputStream(SOURCE_PATH);
4       model.read(fin, NS);
5       }
6       catch (Exception ex) {
7       System.out.println(ex);
8       }
9   }
```

Listing 8: JenaModel.java

La funzione é molto semplice, permette di importare il modello costruito precedentemente con Protégé utilizzando la funzione read() fornita da Jena. Quest'ultima viene richiamata su un oggetto OntModel, precedentemente dichiarato, che rappresenta il nostro schema ontologico. I parametri passati al metodo sono un oggetto rappresentante lo stream in input (creato passando il path dove é fisicamente presente il modello da importare) e il namespace di quest'ultimo, precedentemente dichiarato.

La seguente porzione di codice rappresenta il core della classe:

```
1   /*Reading XML File*/
2   BufferedReader br = new BufferedReader(new FileReader(
        DATA_PATH+"dataset.xml"));
3   String line = br.readLine();
4   while(line!=null){
5
6     /*New article has to be added to the owl file*/
7     if(line.startsWith(ARTICLE)){
8       count++;
9
10      /*Create Individuals*/
11      Iterator<String> aut_it = row_authors.iterator();
12      Individual doc = model.createIndividual(NS +""+count,
            e_document);
13      Individual prod = model.createIndividual(NS+"Working at
            writing paper"+count,production);
14      Individual url = model.createIndividual(NS+row_url,e_url
            );
15      Individual time = model.createIndividual(NS +row_time,
            time_span);
16      Individual e_title = model.createIndividual(NS+row_title
            ,title);
17      Individual journal = model.createIndividual(NS+
            row_journal,e_journal);
18
19      /*Add properties*/
20      e_title.addProperty(title_value, URLDecoder.decode(
            row_title,"UTF-8"));
21      time.addProperty(hasAnno, row_time);
22      doc.addProperty(published_by, URLDecoder.decode(
            row_journal,"UTF-8"));
23      journal.addProperty(journal_value, URLDecoder.decode(
            row_journal,"UTF-8"));
24      url.addProperty(url_value, URLDecoder.decode(row_url, "
            UTF-8"));
25      doc.addProperty(hasTitle, e_title);
26      prod.addProperty(hasProduced, doc);
27      prod.addProperty(has_time_span, time);
28      doc.addProperty(refers_to, url);
29
30      /*For each topic*/
31      Iterator<String> topic_it = row_topics.iterator();
```

```
32          while(topic_it.hasNext()){
33            String curr_topic = topic_it.next();
34            Individual top = model.createIndividual(NS+curr_topic+
                  count,topic);
35            top.addProperty(topic_value, URLDecoder.decode(
                  curr_topic, "utf-8"));
36            doc.addProperty(bears_features, top);
37
38          }
39
40          /*For each keyword*/
41          Iterator<Keyword> keyword_it = row_keywords.iterator();
42          while(keyword_it.hasNext()){
43            Keyword curr_keyword = keyword_it.next();
44            Individual Key = model.createIndividual(NS+
                  curr_keyword.getText()+count,keyword);
45            Key.addProperty(hasText, URLDecoder.decode(
                  curr_keyword.getText(),"utf-8"));
46            Key.addProperty(hasRelevance, ""+curr_keyword.
                  getRelevance());
47            doc.addProperty(is_identified_by, Key);
48
49          }
50
51
52          /*For each author*/
53          while(aut_it.hasNext()){
54            String curr_aut = aut_it.next();
55            Individual aut = model.createIndividual(NS+curr_aut,
                  author);
56            aut.addProperty(name, URLDecoder.decode(curr_aut,"utf
                  -8"));
57            prod.addProperty(carried, aut);
58          }
59
60          //cleaning
61          refreshVariables();
62
63        }
64        else if(line.startsWith(AUTHOR)){
65          row_authors.add(extractInfo(line));
66        }
67        else if(line.startsWith(TITLE)){
68          row_title = extractInfo(line);
69        }
70        else if(line.startsWith(YEAR)){
71          row_time = extractInfo(line);
72        }
73        else if(line.startsWith(TOPIC)){
74          row_topics.add(extractInfo(line));
75        }
76        else if(line.startsWith(KEYWORD)){
77
78          double relevance=Double.parseDouble(extractInfo(br.
```

```
            readLine()));
79          String text=extractInfo(br.readLine());
80          Keyword newKeyword =new Keyword(text, relevance);
81          row_keywords.add(newKeyword);
82        }
83        else if(line.startsWith(URL)){
84          row_url = extractInfo(line);
85        }
86        else if(line.startsWith(JOURNAL)){
87          row_journal = extractInfo(line);
88        }
89
90
91        line = br.readLine();
92      }
93      addIndividuals(model,out);
94      br.close();
```

Listing 9: JenaModel.java

Una volta aperto lo stream input del dataset (strutturato in xml), esso viene letto riga per riga estraendone, in maniera differenziata, le varie informazioni, dopodiché sequenzialmente vengono generate le istanze. Trattandosi di OWL, le istanze vengono generate come individuali tramite la funzione Jena createIndividual(), la quale prende come parametri namespace e oggetto relativo alla classe di appartenenza dell'individuale in questione. Generati gli individuali, successivamento vengono connessi semanticamente tra loro tramite l'aggiunta delle proprietá. Il metodo addProperty() viene richiamato sull'oggetto che rappresenta il dominio e riceve due parametri espliciti che rappresentano rispettivamente il tipo di proprietá e il range. Poiché ad ogni articolo puó essere connesso piú di un autore, topic, o keyword, le istanze e le proprietá vengono aggiunte in un ciclo. Il motivo della decodifica utf-8 viene spiegato nella funzione che estrae le informazioni dal documento:

```
1   /**Provides to extract the text between XML tags**/
2   private static String extractInfo(String line){
3     /*remove eventual html tags within the string*/
4     line = Jsoup.clean(line, Whitelist.basic());
5
6     /*utf-8 encoded is needed because of eventual illegal
          characters in the line
7      * ES: '#' is coded as '%23'*/
8     try {
9       line = URLEncoder.encode(line, "utf-8");
10    } catch (UnsupportedEncodingException e) {
11      e.printStackTrace();
12    }
13
14    return line;
15  }
16}
```

Listing 10: JenaModel.java

Tramite libreria Jsoup vengono estrapolate le informazioni testuali dai tag xml, la codifica in utf-8 si é resa necessaria per rimuovere eventuali caratteri speciali dalle URI delle risorse (poiché ció causava un errore in fase di scrittura).

Infine, una volta terminata la lettura del file in input, viene scritto il modello risultante tramite il metodo addIndividuals():

```
1  /**Print the model**/
2  private static void addIndividuals(OntModel model, FileWriter
       out) {
3    if(Key.PRINT_ON_FILE){
4      try{
5        model.write(out, "RDF/XML");
6
7      }
8      catch(BadURIException e){
9        e.printStackTrace();
10     }
11
12   }
13 }
```

Listing 11: JenaModel.java

Viene richiamata la funzione Jena: write() che stampa, operando su uno stream output, le triple in formato RDF. Il risultato dell'esecuzione della classe descritta sul dataset finale utilizzato per la nostra applicazione é un grafo di 176208 triple, sul quale sono state effettuate interrogazioni tramite le seguenti query sparql che andremo a descrivere.

4.5 Query SPARQL

Per poter effettuare al meglio le query sull'ontologia abbiamo deciso di implementare la classe *query-sparql* (file query.php). Tale classe costituita da funzioni che, quando chiamate, restituiscono la query ad esse associate.

La lista degli articoli che contengono un sottoinsieme di keywords e topics dei medesimi dell'articolo di partenza, viene resituita in formato JSON dal file ricerca.php all'interno del quale vengono fatte le chiamate a funzione della classe *query-sparql*. Il JSON restituito é un array contenente la lista di keywords e topics dell'articolo di partenza e da un array contenete oggetti di tipo *article* dove per ognuno riportata la lista di keyword e topic che corrispondono, il titolo e l'id che identifica univocamente il nodo del grafo successivamente associato. Di seguito sono riportate alcune delle query Sparql usate per interrogare l'ontologia. Per ulteriori chiarimenti consultare i file query.php e ricerca.php.

- **Dal titolo di un articolo estrarre topic.**

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix sd: <http://www.semanticweb.org/francesco/
   ontologies/2016/docs#>

SELECT  ?title_value ?topic_value

  WHERE{
  ?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value "Representation of Graphs
     ".
  ?title sd:Title_value ?title_value.
  ?doc crm:P56_bears_feature ?topic.
  ?topic sd:Topic_value ?topic_value.
}
```

- **Dal titolo di un articolo estrarre keywords e relevance.**

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix sd: <http://www.semanticweb.org/francesco/
   ontologies/2016/docs#>

SELECT  ?title_value ?t ?r

  WHERE{
  ?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value "Representation of Graphs
     ".
  ?title sd:Title_value ?title_value.
  ?doc crm:P149_is_identified_by ?key.
  ?key sd:Text ?t.
  ?key sd:Relevance ?r
}
```

- **Dalle keywords dell'articolo di partenza estrae gli articoli che hanno un sottoinsieme di keyword in comune, escludendo l'articolo di partenza.**

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix owl: <http://www.w3.org/2002/07/owl#>
```

```
prefix sd: <http://www.semanticweb.org/francesco/
    ontologies/2016/docs#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>


SELECT   ?title_value  (SUM(xsd:double(?r)) as ?
    totalR)

WHERE
{
  {?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value ?title_value.
  ?doc crm:P149_is_identified_by ?key.
  ?key sd:Relevance ?r.
  ?key sd:Text ?t.
    FILTER(?t="following sense")} UNION
  {?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value ?title_value.
  ?doc crm:P149_is_identified_by ?key.
  ?key sd:Relevance ?r.
  ?key sd:Text ?t.
    FILTER(?t="transition systems")} UNION
  {?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value ?title_value.
  ?doc crm:P149_is_identified_by ?key.
  ?key sd:Relevance ?r.
  ?key sd:Text ?t.
    FILTER(?t="label-disjoint cycles")}UNION
  {?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value ?title_value.
  ?doc crm:P149_is_identified_by ?key.
  ?key sd:Relevance ?r.
  ?key sd:Text ?t.
    FILTER(?t="finite labelled transition")}UNION
   {?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value ?title_value.
  ?doc crm:P149_is_identified_by ?key.
  ?key sd:Relevance ?r.
  ?key sd:Text ?t.
```

```
    FILTER(?t="finite set")}

  FILTER(?title_value!="A decomposition theorem
     for finite
  persistent transition systems").
}

group by ?title_value
ORDER BY DESC(?totalR)
```

- **Dai topics dell'articolo di partenza estrae gli articoli che hanno un sottoinsieme di topic in comune, escludendo l'articolo di partenza.**

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix sd: <http://www.semanticweb.org/francesco/
   ontologies/2016/docs#>

SELECT   ?title_value (COUNT(DISTINCT ?topic_value
   ) AS ?count)

  WHERE{
  ?prod crm:P108_has_produced ?doc.
  ?doc crm:P56_bears_feature ?topic.

  {?topic sd:Topic_value "Software Engineering/
     Programming and Operating Systems".} UNION
  {?topic sd:Topic_value "Computer Systems
     Organization and Communication Networks".}
     UNION
  {?topic sd:Topic_value "Computational
     Mathematics and Numerical Analysis".} UNION
  {?topic sd:Topic_value "Information Systems and
     Communication Service".} UNION
  {?topic sd:Topic_value "Theory of Computation".}
      UNION
  {?topic sd:Topic_value "Data Structures,
     Cryptology and Information Theory".}.
  ?topic sd:Topic_value $topic_value.

  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value ?title_value.
```

```
  FILTER (?title_value != "Representation of
      Graphs").
}
GROUP BY ?title_value
ORDER BY ?title_value
```

Terminata la generazione del grafo, é stato necessario effettuare ulteriori query per estrarre le informazioni di ogni articolo ottenuto. Il file *datiArticolo.php* restituisce, come visto prima per il file *ricerca.php*, un file JSON contenente le informazioni richieste.
Di seguito le query usate:

- **Restituisce gli autori di un articolo**

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix sd: <http://www.semanticweb.org/francesco/
    ontologies/2016/docs#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT  ?author_name

  WHERE{
  ?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value "Representation of Graphs
      ".
  ?prod crm:P14_carried_out_by ?author.
  ?author sd:name ?author_name
}
```

- **Restituisce anno di pubblicazione e rivista**

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix sd: <http://www.semanticweb.org/francesco/
    ontologies/2016/docs#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT  ?journal ?anno

  WHERE{
  ?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
```

```
  ?title sd:Title_value "On the Power of Chain
     Rules in Context Free Grammars".
  ?doc sd:published_by ?journal.
  ?prod crm:P4_has_time-span ?date.
  ?date sd:Anno ?anno


}
```

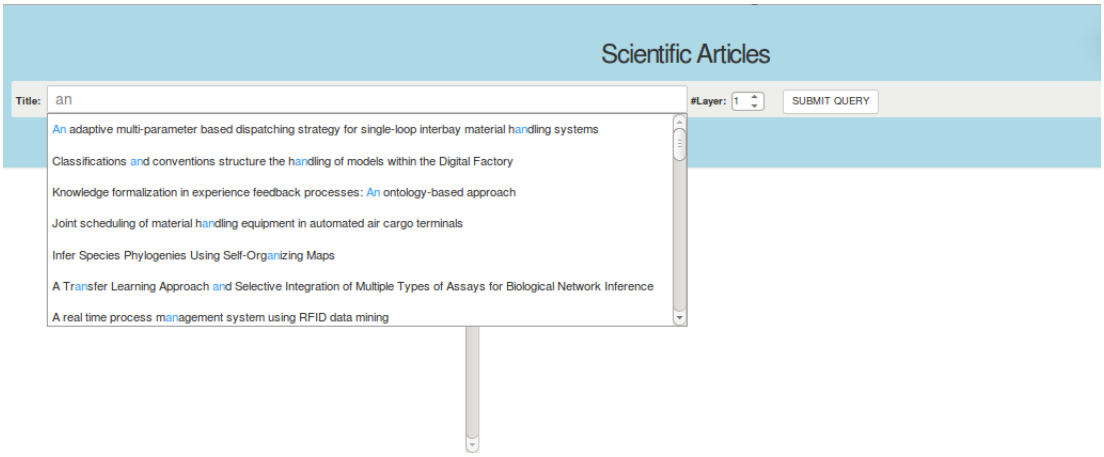
- **Restituisce l'url di un articolo**

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix sd: <http://www.semanticweb.org/francesco/
   ontologies/2016/docs#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?url_value
   WHERE{
  ?prod crm:P108_has_produced ?doc.
  ?doc crm:P102_has_title ?title.
  ?title sd:Title_value "Representation of Graphs
     ".
  ?doc crm:P67_refers_to ?url.
  ?url sd:Url_value ?url_value
}
```

4.6 Front-end

Il lavoro viene presentato come una pagina web interattiva mediante la quale possibile, partendo dall'articolo inserito, visualizzare il grafo generato dalle relazioni semantiche che l'articolo cercato ha con gli altri articoli dell'ontologia. Il layout é molto semplice e sfrutta lo stile di base offerto da Bootstrap.



La Search Bar é stata implementata in javascript (*autocomplete.js*). I titoli degli articoli sono contenuti in un array che viene, ad ogni immissione, richiamato per mostrare i titoli che soddisfano la ricerca. É possibile scegliere il numero di livelli da usare per generare il grafo secondo la logica discussa nei paragrafi precedenti. Sottomessa la query, viene invocata la funzione *generate-graph*.

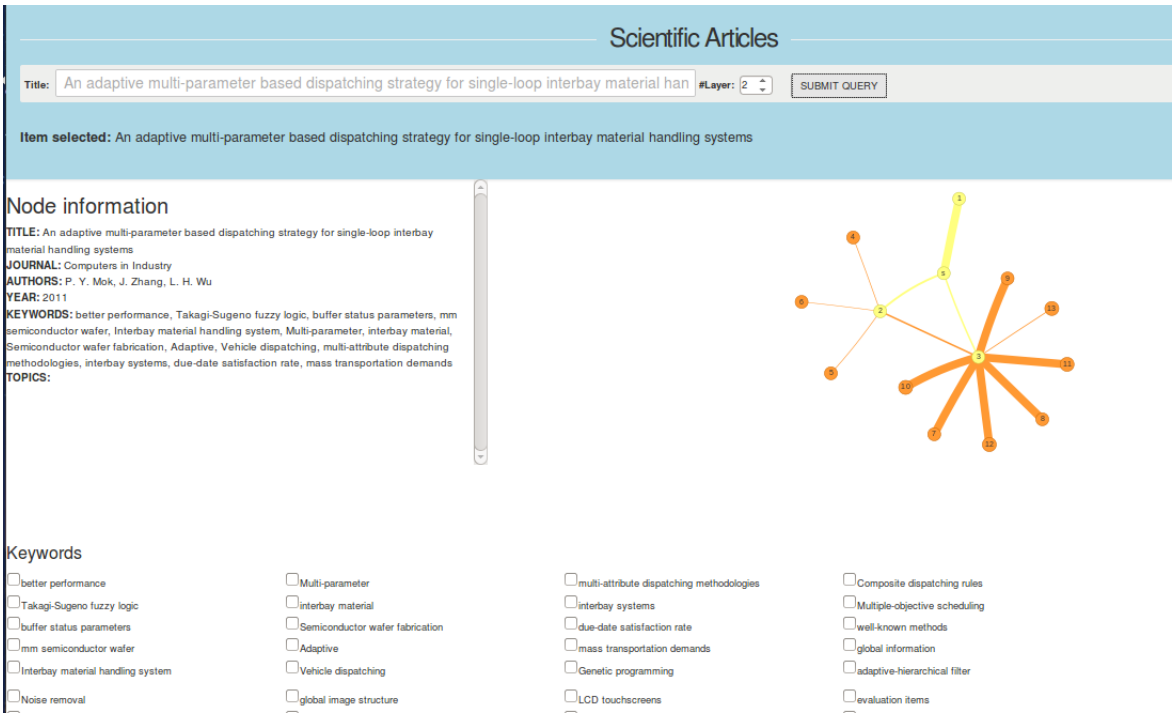
```
1       function generate_graph(data, title, layer) {
2
3     try {
4             var parsedData = JSON.parse(data);
5             }
6         catch(err) {
7              console.log(data);
8              }
9
10    //adding root if nodes is empty
11    if(nodes.length == 0){
12      nodes.push(JSON.parse('{"id":"' + title +'", "label": "s",
            "title":"Sorgente", "color": "' + color[0] + '"}'));
13    }
14    for(var key in parsedData.articles){
15
```

```
16        var existFlag = false;
17        var existEdgeFlag = false;
18        var article = parsedData.articles[key];
19        var actualId = article.id;
20        var actualTitle = article.title;
21        var relation="<b>Topics:</b><BR>";
22        var actualEdge = {};
23        var edgeValue = 0;
24        var sumOfRelevance = 0;
25        actualEdge["sourceTitle"] = title;
26        actualEdge["title"] = actualTitle;
27        actualEdge["keyword"] = article.keyword;
28        actualEdge["topic"] = article.topic;
29
30        for(var key2 in article.topic ){
31          relation+=article.topic[key2]+"<BR>";
32        }
33
34        relation += "<b>Keywords:</b><BR>";
35
36        for(var key2 in article.keyword ){
37          relation+=key2+"<BR>";
38          sumOfRelevance +=parseFloat(article.keyword[key2]);
39        }
40
41        for (var oldEdge in edges){
42          if((edges[oldEdge].from == actualTitle && edges[oldEdge
                ].to == title) || (edges[oldEdge].from == title &&
                edges[oldEdge].to == actualTitle))  {
43            existEdgeFlag = true;
44            break;
45          }
46        }
47        if(nodes.indexOf(actualTitle) < 0){
48            actualTitle = actualTitle.replace('"', '');
49          actualTitle = actualTitle.replace('"', '');
50
51          var nodeToPush = JSON.parse('{"id":"' + actualTitle +'",
                "label":"' + id + '", "color":"' + color[layer] + '
               "}');
52
53          for (var oldNode in nodes) {
54              if(nodes[oldNode].id == actualTitle) {
55                existFlag = true;
56                break;
57              }
58          }
59          if(!existFlag && sumOfRelevance>0) {
60            nodes.push(nodeToPush);
61              newNodesNextLevel.push(actualTitle);
62              id++;
63            existFlag = false;
64          }
65        }
```

```
66
67
68
69          for(var currentValue in article.keyword) {
70            edgeValue +=parseFloat(article.keyword[currentValue]);
71          }
72          if(!existEdgeFlag && edgeValue>0){
73            edges.push(JSON.parse('{"from":"' + title + '", "to": "'
                  + actualTitle + '", "id": ' + edgeId + ', "value":'
                + edgeValue + ', "title":"' + relation + '", "color
                ":"' + color[layer] + '"}'));
74            edgesMap[edgeId] = actualEdge;
75            edgeId++
76          }
77      }
```

Listing 12: indexVisjs.html

Tale funzione popola gli array node ed edge che vengono poi usati dalla libreria Vis.js per generare il grafo. Di seguito viene mostrato un esempio.



Da questo grafo é possibile, mediante le checkbox sottostanti, scegliere quali keywords o topic usare per costruire un nuovo grafo contenente i soli archi che presentano la selezione fatta. Questo permette di restringere la ricerca qualora si fosse interessati solo a particolari contesti. Viene inoltre

fornita la possibilitá di tornare al grafo di partenza senza dover sottomettere la query di partenza.

Come si puó notare dall'immagine precedente, la parte sinistra del layout é riservata ad un contenuto informativo. Il contenuto varia a seconda di quello che si vuole visualizzare.

- **Node Information:** Se si effettua onMouseHover su di un nodo. Vengono presentate tutte le informazioni di un articolo quali: titolo, autori, rivista, anno di pubblicazione...
- **Edge Information:** Se si effettua onMouseHover su un arco. Viene mostrata la relazione semantica tra il nodo sorgente ed il nodo ottenuto dalla query mediante i campi: Source, Dest, Somma relevance. Sull'arco vengono mostrate le liste di topics e keywords che hanno generato la relazione semantica tra i nodi.
  Di seguito é riportato un esempio.

Facendo un doppio click su di nodo viene fatto il redirect alla sorgente dell'articolo, in modo da poter acquisire ulteriori informazioni.

Osservando il grafo si puó notare come lo spessore degli archi non sia lo stesso. Questo dipende dal valore della relevance, quindi dal legame semantico che c'é tra i nodi. Questa differenza permette di percepire subito quali sono gli articoli piú correlati, facilitando quindi l'aria d'interesse che si desidera.

Di seguito un estratto di codice che mostra come vengono creati i nodi e gli archi.

Creazione Node:

```
var nodeToPush = JSON.parse('{"id":" +actualTitle +'", "label
    ":" ' + id + '", "color": "' + color[layer] + '"}');
```

- id: é il titolo dell'articolo.
- label: identificativo univoco del nodo nel grafo (valore numerico, solo per la sorgente é s).
- color: colore del layer d'appartenenza.

Creazione Edge:

```
edges.push(JSON.parse('{"from":" ' + title + '", "to": "' +
    actualTitle + '", "id": ' + edgeId + ', "value":' + edgeValue
     + ', "title":" ' + relation + '", "color": "' + color[layer]
    + '"}'));
```

- from: nodo sorgente.
- to: nodo destinatario.
- id: identificativo dell'arco.
- value: spessore dell'arco.
- color: colore del layer d'appartenenza.

5 Conclusioni

Un altro utilizzo interessante, anche se non pienamente pertinente con gli obiettivi del lavoro, é la possibilitá di prevedere query che prendano in input dei topic e restituiscano tutti gli articoli che hanno quel topic e le relative parole chiave. In questo modo, cambiando il contesto (e.g. l'ontologia viene popolata con un set diverso di articoli, magari provenienti da una serie di conferenze), si puó mostrare come lo stesso argomento viene trattato in maniera diversa a seconda del contesto. Se ad esempio il topic é *Bioinformatica*,

in un contesto di proceedings di conferenze informatiche le parole chiave potrebbero essere *Algoritmi*, *Strutture Dati*, *Complessitá Computazionale*, etc. Mentre lo stesso topic in un contesto di proceedings di conferenze biologhe potrebbe avere come parole chiave *Ribosomi*, *Proteine*, etc.