

**Semantic Web**

**Report**

****

Contributors:

|  |  |
| --- | --- |
| OZER Mert | MESSAOUDI Amin |
| Master Data & Knowledge, Télécom Paris-Tech, Paris-Saclay University  22.10.2017 | |

Table of content

[**Specify your task**](#_kr8jgqa0ydki) **2**

[**Analyse the task**](#_j4ozqw46sfrx) **2**

[**How to find the answer**](#_yooyp6hiovsl) **3**

[Manual](#_dvt1h89ju8pd) 3

[Do web scraping](#_6qqp8qtbhnj5) 4

[Use of semantic approaches](#_mp27qnuflrm) 5

[Relations](#_2oihrllv2tv3) 5

[RDF](#_az9lpu99ph7a) 5

[**Conclusion**](#_9enczkkxnyf9) **9**

[**Results**](#_7ces0ty8fcai) **10**

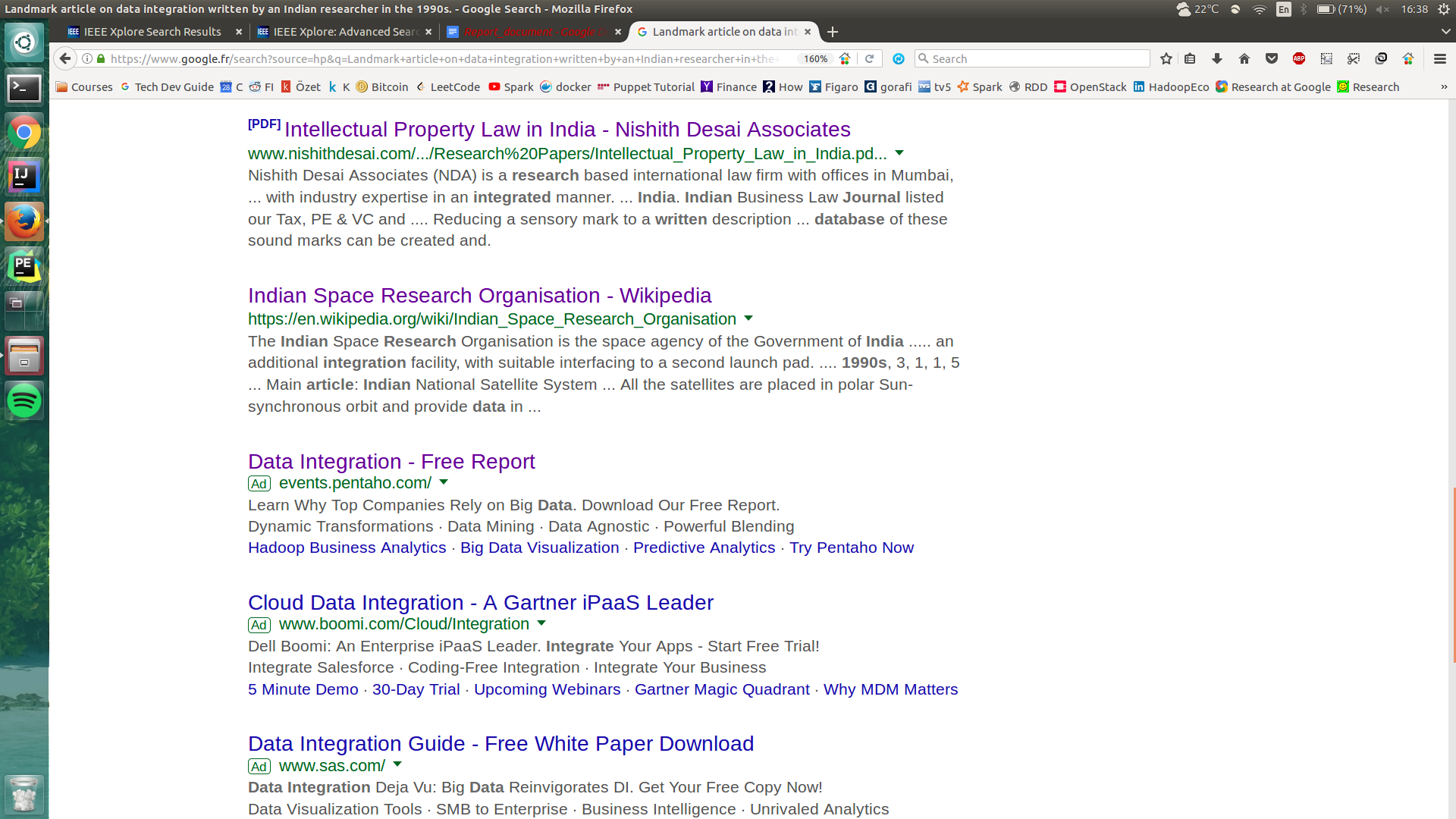
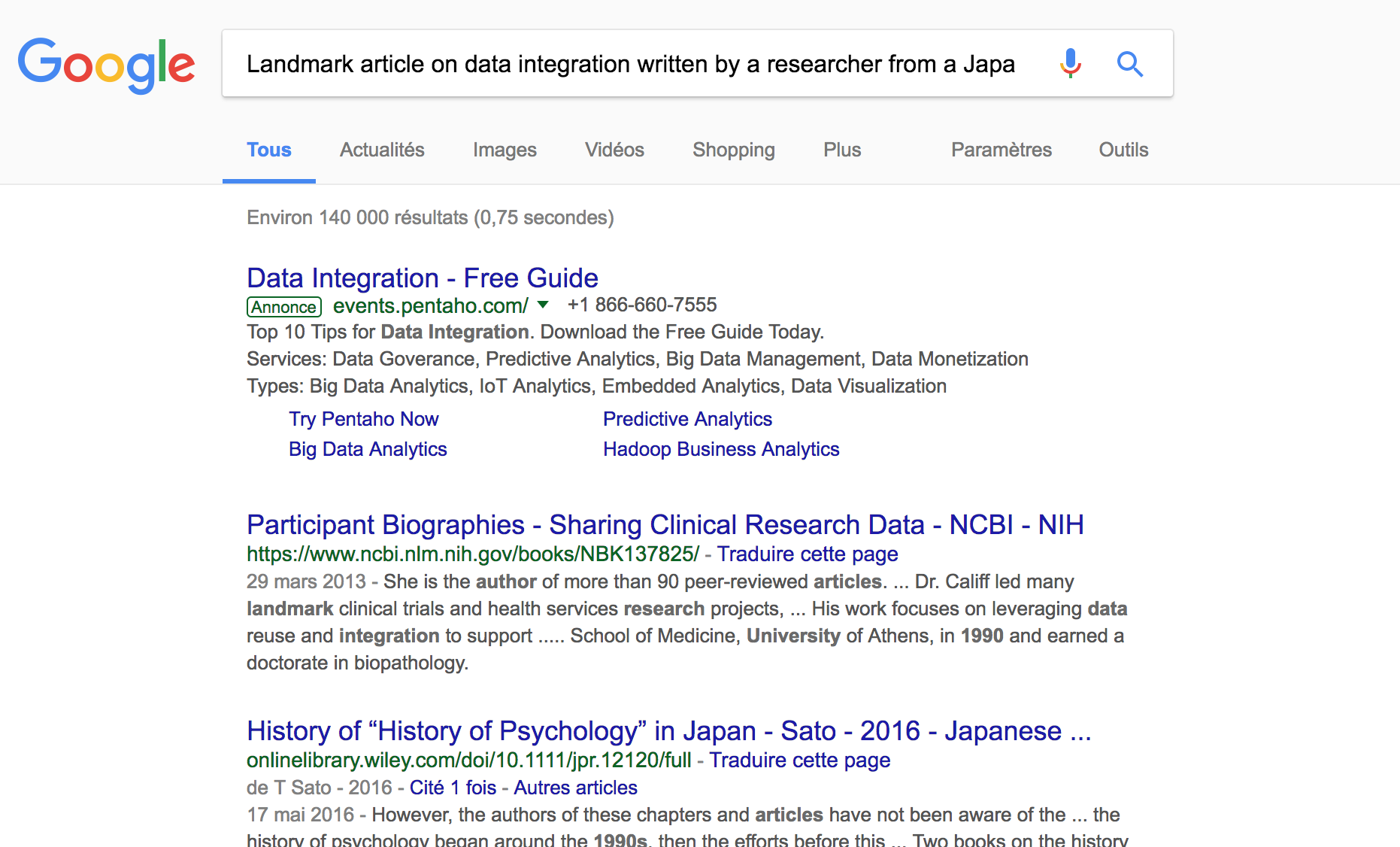
[**Sources**](#_awo7eadv7k6j) **10**

# Specify your task

Semantic web is today important for the world wide web. It gives an easier way to share information. And makes it is easy to find by giving more precise results.

The goal of this report is to show how to answer to the question : “***Landmark article on data integration written by a researcher from a Japanese University in the 1990s***”.

Searching this question on Google search engine we are not able to have the answer (Fig 1).



*Figure 1: Results by Google not answering to the question (preview of three firsts results).[[1]](#footnote-0)*

# Analyse the task

We can ask why Google can not answer to it. One way to see it, is that this is a complex question. The question have several parameters to take in account:

Let’s analyse the question itself (Fig 2):



*Fig 2: Parameters of the question*

Several parameters needs to be handled here, that’s why the task is complicated for Google.

So if we translate it in a regular expression (regex after) we have something like:

“*Landmark(\s)+****([[:alpha:]]+)****(\s)+on(\s)****([[:alpha:]]+)****(\s)+ etc.*”. In this regex we can capture the type of document we want “article” (first bold parenthesis) but also the first word of subject which is “data” (second bold parenthesis). But as we can see it is just a small version of the complete regex expected. And we are only covering one type of question, starting from “Landmark”.

We would like to be able to answer to a question like “Find me data integration articles written in the 1990’s in Japanese University”. The format is different but the meaning remains the same.

But there is a solution, using LUA language we can transform our task into something more general with patterns: “#beginQuestion #typeDocument #pronoun #subjects #verb #proverb #a #job etc.”.

Doing this bring some semantic to the question and we can have in #beginQuestion keywords like “Landmark”, “Find me”,“Is there” and we can do the same with the other patterns (except “Is there” question where the answer will be Yes or No instead of outputting a list of articles).

And then we can reorder as we wish the order of the patterns and be able to accept more type of questions. By saying we expect #beginQuestion at the beginning but we expect type of document somewhere after that, it could be after #subjects like:

* ^#beginQuestion (?=.\*typeDocument.\*) meaning that we want to start with begin question and we expect after it somewhere to have the type of document.

# How to find the answer

## Manual

The easiest way should be to google the question and try to see results. Since, we tried this step and couldn’t find it, we are moving to the second step (Figure 1).

Second step is to find all research papers providers. So after doing some research we found a site gives all names of research papers:

<https://www.sciencebuddies.org/science-fair-projects/competitions/finding-and-accessing-scientific-papers>

After that, we need to search our topic in all research paper provider's site and then we will look to every document in the results (and we don’t know if we are seeing all results we want),

Next thing will be analyzing the documents. So, for every document we have, we will look if at least one research is from Japanese University, keywords are related to “data integration”, published year is in 1990s and note id of the paper, but we know we don’t have such time and we can make mistakes. So, we should automize our task in order to find our results faster and with more accuracy. Which lead us to two solutions : web scraping and semantic approaches with RDF.

## Do web scraping

The idea is to access to DOM[[2]](#footnote-1) elements of a web page, and then parse them and return matching elements. In order to retrieve relevant data from it. Let’s give an example for this paper: <http://ieeexplore.ieee.org/document/8073390/>

We search data integration and for every element do these tasks.

Get the next element(Here it will be the link we provided)

For all elements

Get class "u-pb-1 doc abstract-pubupdate ng-binding ng-score" match with date time and if date is between 1990-2000.

For all authors from going to the link+/authors (ex:<http://ieeexplore.ieee.org/document/8073390/authors>)

For all element from class "item.bio.graphic"

Look if it is a Japanese University(Not going to the details of it but before we defined it with web scraping again with this link: [http://univinjapan.com/list.html](http://univinjapan.com/list.html)))

If it is an Japanese University then go to keywors section (<http://ieeexplore.ieee.org/document/8073390/keywords>)

Get all keywords and id of the document(because later on we'll do a MapReduce job to see if the keywords are really about "data integration"

End for

End for

End If

End for

At this point our data will look like **“First Phase”.** After mapping every keyword and reducing by how many times they appeared, our data will look like **“After MapReduce Job”.**

If we apply the algorithm for top elements in the keywords of **“After MapReduce Job”**

and return all documents we will solve our problem.

**First Phase After MapReduce Job**

(Data integration, 1) -----> (Data integration, 3678)

(Entangled Cyberspace, 1) -----> (Data cleaning, 857)

... ...

(Data and information sources, 1) -----> (Information inter-dependence, 1)

(Information inter-dependence, 1) -----> (Entangled Cyberspace, 1)

Advantages:

We can answer the question, also with the help of technologies that we can use, we can scale our code (use SAX instead of DOM) and our application won’t get affected by the number of documents.

Disadvantages:

Doing web scraping is a good solution, but also could be really hard to implement when we have a lot of resources to handle. We mentioned before about IEEE website but we can have also Google Scholar, NCBI PubMed and so on. And to retrieve data homogeneously from each website can bring problem to having an well formed data. Finding a good web scraping algorithm for all sites is required.

## Use of semantic approaches

### Relations

Using semantic we can bring some relations between data. For example we would like to say for an article : hasUniversity(document 1, university 1), hasType(document 1, article), hasCountry(university 1, Japan) or even hasDate(article, 1990’s). But also have article which sub topic is data integration.

We can request (small pseudo code of the request[[3]](#footnote-2)) the answer by saying:

for each document "d" and university "u":

# def store(knowledge, array) function storing knowledge into array

storeIf(hasType(d, article) ^hasUniversity(d, u) ^ hasUniversityCountry(u, Japan)

^ hasSubject(d,data+integration) ^ hasAuthor(d, auth)

^ isResearcher(auth, researcher)

^ hasYear(d,"1990's"), knowledgeArray)

End for

Where hasUniversity(d,u) is a way to say it exists one author at least that has written the paper from Japan.

The algorithm above is just an example how we could illustrate our request. We can easily through semantic translate the request into relations, and have an answer of it. Here we store the answer as an array, containing list of articles that respects all conditions.

### RDF

Another approach is to use semantic through RDF and ontology. Using DBLP biography website we are able to find, using their search engine, related subjects to “data integration”. These subjects could be part of a book, theses, journals, conferences/workshops papers and so on.

The ontology of DBLP is represented as an RDF triples (Fig 3). The predicate is the relationship that links subject to the object.



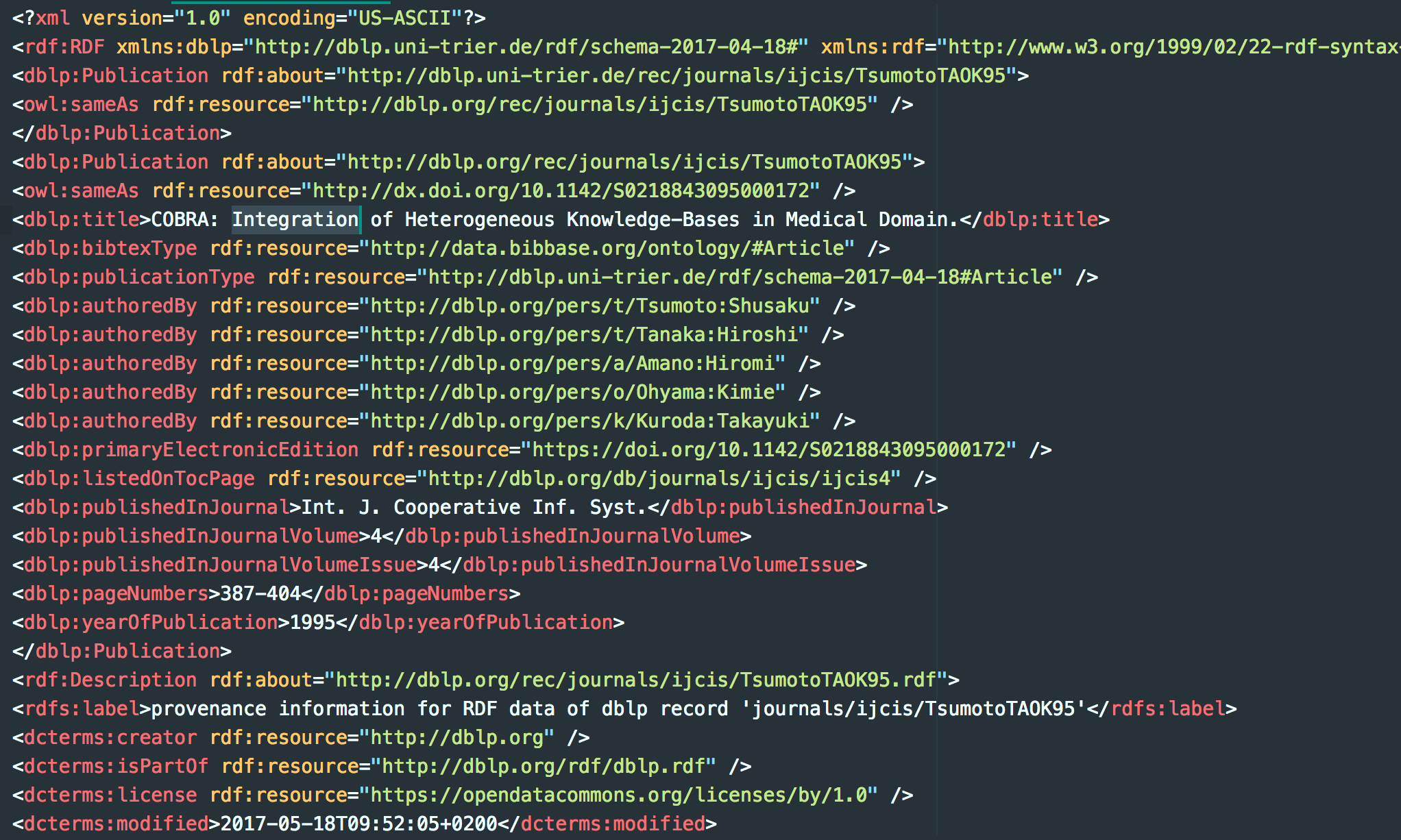
*Fig 3: RDF triples*

A small version of this graph can be found on <http://dblp.l3s.de/d2r/page/publications/journals/accs/AlievAAG07>. This url is already deep inside the RDF telling us we are looking for publications/ of type journals/ as the journal “accs” which is “Automatic Control and Computer Sciences”.

Requesting data integration articles from 1990 need is requested by this url :

<http://dblp.uni-trier.de/search?q=data%20integration%20type%3AJournal_Articles%3A%20year%3A1990%3A>.

Taking an article related to data integration from the search engine give us an RDF xml represented like that:



*Fig 4: “COBRA: Integration of Heterogeneous Knowledge-Bases in Medical Domain, Tsumoto Shusaku, Tanaka Hiroshi, Amano Hiromi, Ohyama Kimie, Kuroda Takayuki” RDF XML representation preview[[4]](#footnote-3)*

Then we need to parse this data and make a custom RDF from it. We will filter only relevant data.

The algorithm is:

def loopGraphOntology(Graph G, conditions):

rdfsemantic = new RDFSemantic() # Our custom RDF graph rdfsemantic

for all nodes in G.getNodes():

for all document D inNodes.getDocumentsType():

relevantData = keepRelevant(D, conditions) # Take relevant data

if(relevantData != null): # if we found that it matches condition

kw = buildKeyWords(D) # find key words related to this

# kw is a list of keywords, for each of it we add the document D in those keywords

# Also second step is to increment weight between each keyword found with others

# ⇒ Same idea as map reduce so if keyword 1, keyword 2 predicate is score = 3

# mean that those two keywords appeared 3 times together in several articles

# store RDF

rdfsemantic.pushRDFNode(kw,relevantData)

Store(rdfsemantic)

End



*Fig 5: Graph representation of the RDF ontology*

***Legend*** : Black background nodes represent the relevant data for answering to the question

White background represents other nodes not relevant (not stored)

Green background represents what what we have to construct to improve ontology

If the green node (Fig 5) is missing in the RDF, we could ask ourselves how DBLP search engine manage to find data integration articles. It will simply look on the title, in the description and in the label of the book.

Each green node will have a weight with other keywords. More the weight is big, more the keywords are linked.

In the example above we choose to merge “data and integration” into one node to simplify it. But it should be two distinct nodes and union of both keywords and articles (ordered by how many times that item occurred, like in ElasticSearch) will be processed.

Unfortunately, this solution is not enough, we could have books talking about data integration without having it in title.

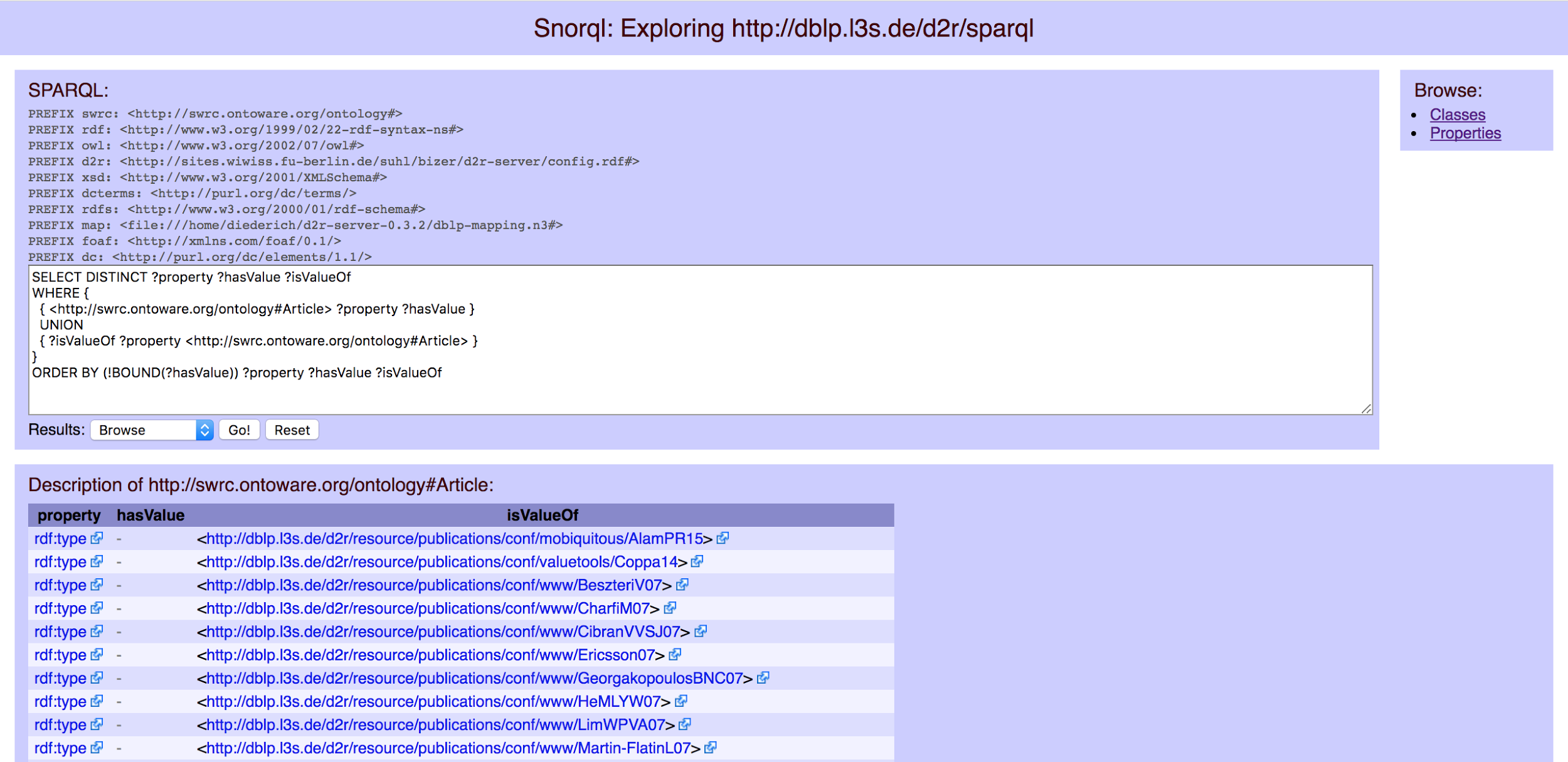
That’s why we thought that adding keywords to our RDF, inspired by scrapping solution from IEEE example where keywords are available. We can add keywords dimension to the RDF graph and have more accurate answers. Since DBLP look in several websites at same time, we could insert keywords found in IEEE (or other website) in RDF.

The idea is to start from the root, look for all articles. Filter them by keywords linked to data and/or integration.

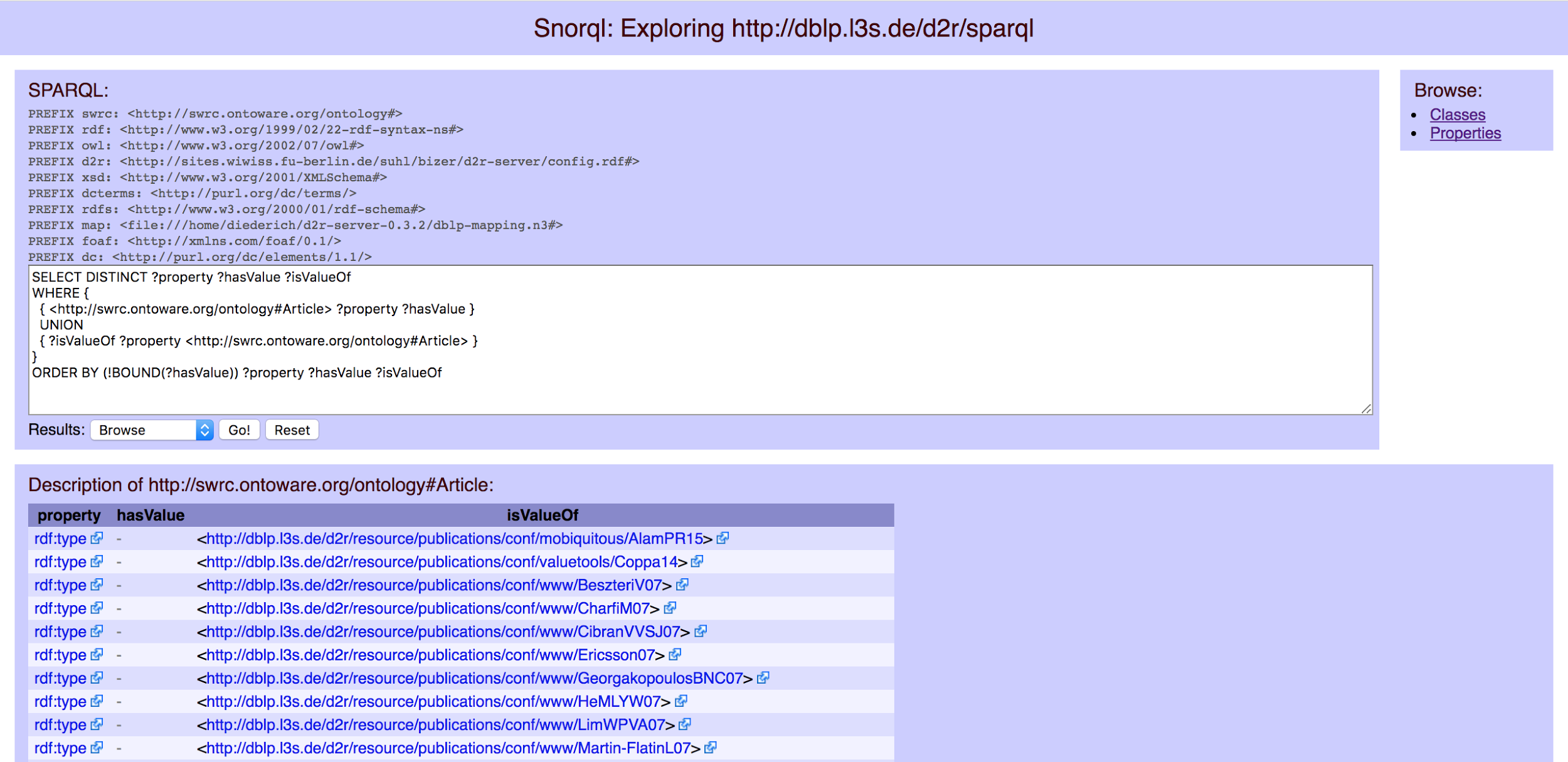
Then for each article found we need to filter them by date, type of authors and university linked to the country we would like to search.

Once all condition gathered we can store the articles found in a list of result to show to user.

Going through all nodes of DBLP will be done in SPARQL. We have a shortened version of the database. We can run any algorithm we want. If we run the algorithm we described in chapter “[Use of semantic approaches](#_mp27qnuflrm)“ we can get our results. We just wanted to show a simple output for all articles in graph (Fig 5-1 and 5-2).



*Fig 5-1: Get all articles in graph (Query)[[5]](#footnote-4).*



*Fig 5-2: Get all articles in graph (Result)[[6]](#footnote-5).*

Advantages

The good thing in this solution is that we can easily iterate in a graph. And finding relation has never been easier in RDF.

We have more choice in this ontology, the answer will be more accurate. It is also maintainable in time comparing to web scraping.

In web scraping if the website changes it’s html or DOM elements the algorithm need to be re-written, while in RDF here even if graph have more nodes, we will be able to handle it more easy than web DOM changes.

Disadvantages

As seen before in the figure A, a noticeable improvement that we can imagine is adding keywords features. As explained in web scraping we can have real meaning using related keywords for a “data integration” subject, to find more related documents.

Getting through all nodes of DBLP RDF could cost a lot, we need to have good resources and machines.

# Conclusion

In [analyze the task](#_j4ozqw46sfrx) section, we evaluated our two approaches:

[Web scraping](#_6qqp8qtbhnj5) was hard to implement, needed unique approaches for every web site, but could be scalable.

[Semantic approaches](#_mp27qnuflrm) was easy to implement because of there is only one ontology to deal with, if we will try read a huge document cannot read into the memory it might.

In this case DBLP website has not a complete database, resources are not accessible from Google Scholar. But still, answers can be generated by the resources already available.

Any programming language that we will use to get data from our ontology could be run as a daemon of the operating system, and since it is easier to implement, with lots of test cases we can make sure that our code will run safe.

We could easily change parameters (Fig 2) and change the year of publication, the university country by changing the input of the program.

# Results

Results should look like a list of articles that matches the query. If no article found for this specific question we can answer “No article found”. In our case we have a preview of the answers:

* [**Akihiko Takahashi**](http://dblp.uni-trier.de/pers/hd/t/Takahashi:Akihiko)**,** [Masatoshi Ishikawa](http://dblp.uni-trier.de/pers/hd/i/Ishikawa:Masatoshi):

Signal processing architecture with bidimensional network topology for flexible sensor **data integration** system. [IROS **1993**](http://dblp.uni-trier.de/db/conf/iros/iros1993.html#TakahashiI93): 407-413, Nat. Inst. of Biosci. & Human-technol., Tsukuba City, **Japan**

* [**Shusaku Tsumoto**](http://dblp.uni-trier.de/pers/hd/t/Tsumoto:Shusaku), [Hiroshi Tanaka](http://dblp.uni-trier.de/pers/hd/t/Tanaka:Hiroshi), [Hiromi Amano](http://dblp.uni-trier.de/pers/hd/a/Amano:Hiromi), [Kimie Ohyama](http://dblp.uni-trier.de/pers/hd/o/Ohyama:Kimie), [Takayuki Kuroda](http://dblp.uni-trier.de/pers/hd/k/Kuroda:Takayuki):

COBRA: **Integration** of Knowledge-Bases with Case-Databases in the Domain of Congenital Malformation. [AIME **1995**](http://dblp.uni-trier.de/db/conf/aime/aime1995.html#TsumotoTAOK95): 393-394

Department of Information Medicine Medical Research InstituteTokyo Medical and Dental University Tokyo, **Japan**

* [**Yahiko Kambayashi**](http://dblp.uni-trier.de/pers/hd/k/Kambayashi:Yahiko)**:**

Research and Development of Advanced **Database** Systems for **Integration** of Media and User Environments. [DASFAA **1999**](http://dblp.uni-trier.de/db/conf/dasfaa/dasfaa99.html#Kambayashi99): 3-5, Kyoto University, **Japan**

* [**Atsuyuki Morishima**](http://dblp.uni-trier.de/pers/hd/m/Morishima:Atsuyuki), [Hiroyuki Kitagawa](http://dblp.uni-trier.de/pers/hd/k/Kitagawa:Hiroyuki):

InfoWeaver: Dynamic and Tailor-made **Integration** of Structured Documents, Web, and **Databases**. [ACM DL **1999**](http://dblp.uni-trier.de/db/conf/dl/dl99.html#MorishimaK99): 235-236, Institute of Information Sciences and Electronics, University of Tsukuba, Tennohdai, Tsukuba, Ibaraki 305-8573, **Japan**

# Sources

[1] Lectures of teacher Yue Ma, LRI

[2] CS 499/699 – Logic for Computer Scientists, Pascal Hitzler

<http://daselab.cs.wright.edu/teaching/w11/2011-03-10-logic-02.pdf>

[3] RDF graph of DBLP (Dataset) <https://old.datahub.io/dataset/l3s-dblp>

[4] DBLP website <http://dblp.org/>

[5] Demo

<http://dblp.l3s.de/d2r/snorql/?describe=http%3A%2F%2Fdblp.l3s.de%2Fd2r%2Fresource%2Fpublications%2Fjournals%2F4or%2FAissi06>

[6] DBLP RDF description <http://dblp.org/faq/What+do+I+find+in+dblp+xml>

[7] Lei Zou, Jinghui Mo, Lei Chen, M. Tamer Oezsu, Dongyan Zhao : gStore : Answering SPARQL Queries via Subgraph Matching. PVLDB 4(8) : 482-493 (2011)

1. Source : Google search engine [↑](#footnote-ref-0)
2. Document Object Element [↑](#footnote-ref-1)
3. Not in Python [↑](#footnote-ref-2)
4. Source : DBLP RDF [↑](#footnote-ref-3)
5. Source : [http://dblp.l3s.de/d2r/snorql/](http://dblp.l3s.de/d2r/snorql/?describe=http%3A%2F%2Fdblp.l3s.de%2Fd2r%2Fresource%2Fpublications%2Fjournals%2F4or%2FAissi06) [↑](#footnote-ref-4)
6. Source : [http://dblp.l3s.de/d2r/snorql/](http://dblp.l3s.de/d2r/snorql/?describe=http%3A%2F%2Fdblp.l3s.de%2Fd2r%2Fresource%2Fpublications%2Fjournals%2F4or%2FAissi06) [↑](#footnote-ref-5)