TOWARDS SEMANTIC ACCESS TO SCIENTIFIC PUBLICATIONS

Author: Asier ADURIZ
Tutor: Dra. Ana FREIRE
Tutor: Dr. Diego SAEZ

Web Research Group, Universitat Pompeu Fabra
# Contents

ABSTRACT ....................................................................................................................... IV

ACKNOWLEDGEMENTS ................................................................................................. V

INTRODUCTION ............................................................................................................. 1

  PROBLEM STATEMENT ......................................................................................... 1

MOTIVATION .................................................................................................................. 2

THESIS STATEMENT ...................................................................................................... 2

1 – BACKGROUND ........................................................................................................ 2

  1.1 – WEB CRAWLING ....................................................................................... 2
      1.1.1 – Jsoup ................................................................................................. 4
  1.2 – DEALING WITH BIG DATA ......................................................................... 4
      1.2.1 – Indexing in Elastic Search ................................................................. 6
      1.2.2 – Document ......................................................................................... 6
      1.2.3 – Type .................................................................................................. 6
      1.2.4 – Mapping ........................................................................................... 6
  1.3 – DOCTOR INVENTOR TEXT MINING FRAMEWORK .................................... 7
      1.3.1 – Babelfy .............................................................................................. 8

2 – PROPOSAL .................................................................................................................. 8

  2.1 – OVERALL PROCESS OVERVIEW .............................................................. 8
      2.1.1 – Step 1: Crawling Knowledge ............................................................... 9
      2.1.2 – Step 2: Downloading Content ............................................................. 9
      2.1.3 – Step 3: Content Transformation .......................................................... 9
      2.1.4 – Step 4: Content Indexation ................................................................. 9
  2.2 – ARCHITECTURE OVERVIEW ...................................................................... 10
      2.2.1 – Database Server ............................................................................. 10
      2.2.2 – GlassFishWS ............................................................................... 11
      2.2.3 – WebSeek ......................................................................................... 12
      2.2.4 – WebDeliver ..................................................................................... 12
      2.2.5 – DocConvertible ............................................................................. 13

4 – RESULTS .................................................................................................................... 14

  4.1 EXPERIMENTAL SETUP ............................................................................... 14
  4.2 USE CASE 1: MOST REPEATED CONCEPTS ............................................. 14
  4.3 USE CASE 2: MOST REPEATED CONCEPTS OVER THE YEARS ............. 15
  4.4 USE CASE 3: SPECIFIC CONTENT INFORMATION .................................... 15
  4.5 USE CASE 4: ANOTHER SPECIFIC CONTENT INFORMATION .................. 16

5 – CONCLUSION ............................................................................................................ 16

5.1 – FUTURE WORK ................................................................................................. 16

REFERENCES .................................................................................................................. 17

LIST OF FIGURES .......................................................................................................... 18

LIST OF TABLES .............................................................................................................. 18
APPENDIX ................................................................................................................................. 19

A – MAPPING .......................................................................................................................... 19

B – USE CASES ...................................................................................................................... 23

B.1 – Use Case 1 ..................................................................................................................... 23
  B.1.1 – Query ....................................................................................................................... 23
  B.1.2 – Tabular Result ......................................................................................................... 23
B.2 – Use Case 2 ..................................................................................................................... 23
  B.2.1 – Query ....................................................................................................................... 23
  B.2.2 – Tabular Result ......................................................................................................... 24
B.3 – Use Case 3 ..................................................................................................................... 26
  B.3.1 – Query ....................................................................................................................... 26
  B.3.2 – Tabular Result ......................................................................................................... 27
B.4 – Use Case 4 ..................................................................................................................... 29
  B.4.1 – Query ....................................................................................................................... 29
  B.4.2 – Tabular Result ......................................................................................................... 30
Abstract

For all of the last decade, the volume of online scientific knowledge grows at an exceptional proportion. This scientific overload means a huge challenge for those with the intention of harvesting such knowledge for varied purposes. This thesis aims to present a system aiming to both crawl and index appropriately online research publications for a succeeding semantic analysis. Doctor Inventor Text Mining Framework provides the semantic knowledge extraction from the scientific publication, and this semantic content is indexing by means of Elastic Search. Altogether, this system is able to solve some interesting use cases that might lead to an advance system for analyzing online publications.
Acknowledgements

This thesis describes the project I have been working on from April 2016 until June 2016. The project was done with the Web Research Group (WRG) alongside the TALN (Tractament Automàtic del Llenguatge Natural, or Natural Language Processing in English) department at the Universitat Pompeu Fabra and was part of the master thesis project to complete my education at the master’s program of the Interactive Intelligent Systems program at the Universitat Pompeu Fabra.

I would like to thank both tutor at the Web Research Group, Ana Freire, and my supervisor of the master’s program, Mireia Farrus, for their guidance and support during both the project and the thesis. Also, to Diego Saez, my second tutor, many thanks for the support and help throughout the duration of the master thesis project. And last but not least, a special mention to Francesco Ronzano of the TALN group whose help and faith towards my good self has been instrumental in my being able to keeping up with the demands of the project.

Furthermore, I feel it is my solemn duty to write words of commendation and gratitude to my family, whose unrelenting supportive nature overcomes every obstacle and late-night hour work efforts seem less heavy on the mind.

To each and every one of you, my kindest and most sincere thanks.

Asier Aduriz
**Introduction**

The number of online public scientific documents is getting bigger and bigger at an increased rhythm growing wondrously by ever year. Plus, according to recent studies, a new research document is publicly available every twenty seconds [1].

Several online repositories are mentioned to give some light on the amount of accessible documents available through the web. PubMed\(^1\) comprises, up to the present date, more than 26 million total documents, with a growth rate around 1.370 new articles every day. CiteSeerX\(^2\), developed and hosted by The College of information Sciences and Technology located in The Pennsylvania State University, contains a total corpus of 7M documents. Also, Elsevier’s Scopus\(^3\) and Thomson Reuther’s ISI Web of Knowledge\(^4\) contain more than 57 and 90 million papers respectively. On a less condescending note, the Cornell University Library arXiv initiative provides access to over 1M e-prints from various scientific domains. Furthermore, the Directory of Open Access Journals\(^5\), one of the most authoritative indexes of high quality, open access, peer-reviewed publications, lists more than 10.800 journals and 2.1M papers.

In this scenario of scientific information overload, researchers, as well as any other interested actor, are overwhelmed by an enormous and continuously growing number of articles to consider. The exploration of recent advances concerning specific topics, methods and techniques, peer reviewing, the writing and evaluation of research proposals and in general any activity that requires a careful and comprehensive assessment of scientific literature has turned into an extremely complex and time-consuming task.

In this landscape the availability of text mining tools able to extract, aggregate and turn scientific unstructured textual contents into well organized and interconnected knowledge is fundamental.

**Problem Statement**

As it has been just mentioned in the introduction section, more and more documents are being published, and more of these documents turn out in online repositories with open access policies for public availability. To give some credit to the increased rate of the number of reachable documents, the total amount of the public content of some of the most relevant and important public repositories are therefore shown afterwards.

Overall, the full text of 27% of the articles indexed by PubMed is available online at the cost of zero. As an addition, in 2011, 17% of the articles indexed by Scopus and ISI Web of Knowledge were freely available and this percentage parameter keeps growing considerably. Some where between 2017 and 2021, more than half of the global papers are expected to be published as Open Access articles.

In other words, the document stockpile will keep increasing since documents are being published faster, and what’s also important, these publications will end up in public access domains. Which

---

2. [http://citeseerx.ist.psu.edu/](http://citeseerx.ist.psu.edu/)
3. [http://www.scopus.com](http://www.scopus.com)
4. [http://www.webofknowledge.com](http://www.webofknowledge.com)
5. [https://doaj.org](https://doaj.org)
means, the stockpile will grow higher and wider by the instant generating real trouble. This slight information overload must be dealt with since numbers will keep on growing with little to no intention of slowing down as the mentality for open access is becoming world widely spread.

**Motivation**

Scientific publications are characterized by several structural, linguistic and semantic peculiarities that makes them difficult to analyze using general purpose text mining tools. The contents of the majority of scientific papers (in many case available as PDF documents) includes common structural elements (title, authors, abstract, sections, figures, tables, citations, bibliography) that often requires customized approaches to be properly characterized.

Existing platforms for knowledge extraction in scientific publications mainly focus on structural elements such as title, or authors, or citations and son on, what don’t profit by the semantics of the documents. Few recent approaches have recently integrated a semantic layer; however, semantic scholar\(^6\) and CORE\(^7\) offer a similar feature in providing deeper content analyzing software.

Also, in order to process these large-scaled repositories of scientific documents a lot of computer power, memory and storage is necessary. It is necessary to design the best approaches for indexing and accessing this information in the optimal way.

**Thesis Statement**

We will build a system for managing online scientific documents. The goal of this system is twofold:

- On one hand, to study the best state-of-art solution for indexing scientific publications.
- On the other hand, to include into our repositories not only structural data but also semantic information.

**1 – Background**

Before getting deep into technical terminologies about the proposal, the aim of this section is to plainly explain the various concepts used to forge a proposed system. The most relevant components of the yet-to-explain system are web crawling and data indexing.

**1.1 – Web Crawling**

Web crawling is the discipline in which an agent, or spider as it’s informally known, ravenously searches the World Wide Web and soaks information from the pages it travels through to save valuable data in order to provide a highly accurate response system to user queries.

Mainly, the purpose of having this spider, or spidering, is what web search engines are fed with, they leech onto existing web pages to collect, parse and store information with a very simple objective: optimize speed and performance in finding relevant documents for a search query a user types into the system.

---

\(^6\) [https://www.semanticscholar.org/](https://www.semanticscholar.org/)

\(^7\) [https://core.ac.uk/](https://core.ac.uk/)
This spiders, start with a list of URLs (Universal Resource Locator) to visit, also known as seeds. For every URL, the crawler searches the content of the page in order to find existing links, collecting them as URLs to visit, and depending on the developing guidelines of the spider, it can also save the current webpage for diverse purposes such as web indexing, or feature extraction.

![Web Crawler Architecture](image)

Figure 1 - Web Crawler Architecture

On figure 1, the different elements that compose a high-level web crawler are showcased. The scheduler is the core component of this architecture, because it’s in charge of electing which webpage is the next one to be crawled by the system. Depending on the criteria, the scheduler will decide to crawl first those web pages which deems more important, or it can have a different behavior.

The multi-threaded downloader is a simple component which receives an URL as input, and then the thread is entrusted with that webpage in order to ultimately store relevant information extracted from the input, as well as stacking in the queue which links the input webpage contains.

With the newest advancements on web design and interaction, technologies such as AJAX arose giving the new web pages the label of Rich Internet Applications. Moreover, this labeled web pages could update its content by various events such as dynamic manipulation of the client side, or asynchronous communication with the server terminal.

As a result of the next step on web developing, spidering faced another challenge, and that was to parse web content that was not on display in the DOM (Document Object Model) until the aforementioned events occurred for the element to be on the hierarchy tree [2].

An important fact of the whole crawling structure is the storing part, which I haven’t mentioned yet, but plays a pivotal role as a support component for the crawling agent, and it acts as a memory storage unit, so that the spider saves not only the visited URLs or priorities to these links, but also them content [3].

It’s also worth pointing that instead of storing the whole plain web page, search engines are nowadays capable of saving minimal data to fully harness the meaning of the web page, also known as indexing.
A further work on the non-stop updating web and its technologies forced to adapt the behavior of spidering to semantic web crawling. Currently known as knowledge extraction, agents now must be capable of triggering the system to reveal dynamic knowledge in order to keep with the freshness of the web [4].

1.1.1 – JSoup

JSoup⁸ is a Java library for working with real-world HTML (HyperText Markup Language). It provides a very convenient API for extracting and manipulating data, using the best of DOM (Document Object Model) and several components that create online web pages. JSoup is designed to deal with all varieties of HTML found in the wild; from pristine and validating, to invalid tag-soup; JSoup will create a sensible parse tree.

JSoup makes a fine and fit addition to the concept of web crawling for this thesis since as it has just now been mentioned, the library generates both a structured and organized object model for the user to easily manipulate its contents to its will.

1.2 – Dealing with Big Data

In order to scale, there are two possibilities: vertical scaling (scaling up) and horizontal scaling (scaling out). Vertical scalability is moving to a bigger, more powerful computer or for example replacing the processor and memory with a more powerful model. However, this comes with its downsides as it becomes more expensive to move to the next more powerful model. Furthermore, there are also certain limitations of power a single computer can currently uphold.

The other possibility, horizontal scalability, can be used to distribute these resources over a cluster of computers. This can be a computer cluster consisting of more than a hundred ‘nodes’, which could all be cheap computers, distributed over a wide area. Working with horizontal scalability is more complex so, in order to manage the resources distributed over a cluster of computers, the load is split up in smaller chunks called ‘shards’, and these shards are distributed over the nodes in the computer cluster.

Apache Hadoop⁹ is a framework that can be used to process a large amount of data by processing subsets of the data on each of the nodes within the cluster. Apache Hadoop is based on the MapReduce model for distributing the problem to be solved over a computer cluster. However, a problem with Apache Hadoop is that the MapReduce model only allows data to be processed in batched.

MapReduce is a programming model and an associated implementation for processing and generating large datasets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key [5].

In short, the map function reads a set of records from an input file, does any desired filtering and/or transformations, and then outputs a set of intermediate records in the form of new key/value combinational pairs [6].

⁸ https://jsoup.org/
⁹ https://hadoop.apache.org
In order to process data in real-time, Storm\textsuperscript{10} was developed by Twitter. Storm works with both ‘spouts’ and ‘bolts’, Spouts are units connected to an information source. They stream data real-time into the Storm cluster. These spouts connect to one or more bolts. Bolts are the units that process the data. Later on, the processed data sets need to be stored after it has gone through the Storm cluster [7].

As described before, NoSQL indexing solutions often offer very limited query functionalities and offer limited to no indexing possibilities. So, what to do if you need a data storage solution that can perform all of the following points besides the standard CRUD (Create, Read, Update and Delete) functionalities:

- Perform full text searches
- Perform these searches in (near) real-time
- Maintain and scale a large, say terabytes, number of indices

Elastic Search,\textsuperscript{11} a newly proposed tool, is one such a solution. Elastic Search is a distributed and scalable search engine built on top of Apache Lucene\textsuperscript{12}. Elastic Search is also attractive because it can be used to store the indexed data as well. Furthermore, Elastic Search is in the first place still a search engine and not a database.

The principal of the intelligent search engine of Elastic Search is basically another software project called Lucene. It is possibly easiest to understand Elastic Search as a part of infrastructure built nearby Lucene’s Java libraries. In Elastic Search everything is related to the actual algorithms for matching text and storing optimized indexes of query terms is executed by Lucene. Elastic Search itself provides a more functional and compact API, scalability, and operational tools overhead Lucene’s search implementation [8].

Lucene is ancient in internet years, seeing back to 1999. It’s also extremely widespread and established. Lucene is used by inexpressible numbers of companies, running the scope from huge corporations such as Twitter, to small startups. Lucene is demonstrated, tested, and is widely considered best-of-breed in open-source search software.

Elastic Search is an open source full-text search engine written in Java that is designed to be distributive, scalable, and near real-time capable. The Elastic Search server is easy to install, and the default configuration supplied with the server is sufficient for a standalone use without tweaking, although most users will eventually want to fine tune some of the parameters. A running instance of the Elastic Search server is called a node, and two or more nodes can form the Elastic Search cluster.

While Elastic Search and traditional RDBMSs differ in many ways, at the higher-level many of the core concepts of Elastic Search have analogues in the RDBMSs world (Table 1). All data in Elastic Search is stored in indices. An index in Elastic Search is like a database in a RDBMS: it can store different types of documents, update them, and search for them. Each document in Elastic Search

\textsuperscript{10} http://storm.apache.org/
\textsuperscript{11} https://www.elastic.co/
\textsuperscript{12} https://lucene.apache.org/
is a JSON object, analogous to a row in a table in a RDBMS. A document consists of zero or more fields, where each field is either a primitive type or a more complex structure.

A document has a document type associated with it; however, all documents in Elastic Search are schema-free, which means that two documents of the same type can have different sets of fields. Document type here is similar to the RDBMS notion of a table: it defines the set of fields that can be specified for a particular document.

<table>
<thead>
<tr>
<th>Elastic Search element</th>
<th>SQL element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Database</td>
</tr>
<tr>
<td>Mapping</td>
<td>Schema</td>
</tr>
<tr>
<td>Document Type</td>
<td>Table</td>
</tr>
<tr>
<td>Document</td>
<td>Row</td>
</tr>
</tbody>
</table>

**Table 1 - Elastic Search vs. SQL**

1.2.1 – Indexing in Elastic Search

Elastic Search stores its data in one of more indices. Using similarities from the SQL schema world, indexing is similar to a database. It is used to store the documents and read them from it. Elastic Search uses Apache Lucene library to write and read the data from the index. Elastic Search index may be built on more than a single Apache Lucene Index by using Shards. When a document is added to an index, the Elastic Search server defines the shard that will be responsible for storing and indexing that document. By doing this, Elastic Search balances the loads between available shards and also improves overall performance, since all shards can be used simultaneously.

1.2.2 – Document

Document is the main entity in the Elastic Search world. At the end, all use cases of using Elastic Search can be carried at a point where it is all about searching for documents and analyzing them. Document consists of fields, and each field identified by its name and can contain either none or multiple values. Each document may have different set of fields, hence the advantage of a schemaless architecture built in Elastic Search.

1.2.3 – Type

Each document in Elastic Search has its type defined. This allows the indexing engine to store various document types in one index and different mapping for different documents types.

1.2.4 – Mapping

All documents are analyzed before being indexed or stored. The input text is divided into tokens, which tokens should be filtered out, or what additional processing, such as removing HTML tags, is needed.

In other words, mapping is similar to a schema definition in SQL databases. A mapping is a crucial part of every index in Elastic Search: it defines all document types within the index and how each document and its fields are stored, analyzed and indexed.

Elastic Search can work with either implicit or explicit mapping. If the Elastic Search server has not been provided with a mapping before a document is inserted, the server will try to infer the type
of the document based on the values in the fields of the document and add this type to the mapping.

While implicit mapping might be an adequate solution in some cases, the use of explicit mapping provides an opportunity to create complex document types and to control how the Elastic Search server analyzes each field. Explicit mapping allows the disabling of indexing of some fields in a document whereas by default the Elastic Search server indexes all fields. This reduces the amount of the disk space needed and increases the speed of adding new documents. It also provides a way to store the data that must not be searched but must be quickly accessible through indexed fields. For example, if we have a set of commits in a version control repository, we might want to index fields like author, date etc., but remove the actual changesets from the index. Whilst changesets remain to be instantly accessible through other fields, they neither take additional disk space nor increase the time required to index a document.

On the other hand, Elastic Search by default flattens indexed fields; i.e. undeclared fields lose the uniqueness property with respect to the singular document. For example, if two separate documents contain two keys those being name and surname and document A would contain for name John and for surname Doe, whereas document B would be Jane Doherty for name and surname in order. If the mapping is not prearranged, Elastic Search would not establish the connection between name and surname in their respective documents; on the contrary, a possible surname query will return both surnames if the searching content for the key name is Jane.

1.3 – Doctor Inventor Text Mining Framework

The pivotal part that comes before the Elastic Search is the conversion from plain document content into semantically structured format. Dr. Inventor\textsuperscript{13} provides technologies in information extraction, document summarization, semantics and visual analytics and that will be used in this project to extract semantic information from scientific publications. Dr. Inventor Text Mining Framework\textsuperscript{14} is a Java library that integrates several Document Engineering and Natural Language Processing tools customized to enable and ease the analysis of the textual contents of scientific publications. It enables users to process the contents of papers both in PDF and JATS XML format. Once imported a paper from a local file or a remote URL, the Framework automatically extracts and characterizes several aspects including:

- Structural elements: title, abstract, hierarchy of sections, sentences inside each section, bibliographic entries.
- Bibliographic entries are parsed and enriched by accessing external web services (Bibsonomy, CrossRef, FreeCite, Google Scholar).
- Inline citations are spotted and linked to the respective bibliographic entry.
- The dependency tree is built from each sentence by considering inline citations.
- The discoursive category of each sentence is identified among: Background, Challenge, Approach, Outcome and Future Work.
- BabelNet synsets are spotted inside the contents of each sentence thanks to Babelfy.

\textsuperscript{13} \url{http://drinventor.eu/}
\textsuperscript{14} \url{http://backingdata.org/dri/library/}
- Subject-Verb-Object graphs are built to represent the contents of paper excerpts (the connectedness of these graphs is enhanced thanks to coreference resolution).
- Relevant sentences are selected with respect to several criteria to build extractive summaries of a paper.

1.3.1 – Babelfy
Babelfy\textsuperscript{15} is a unified, multilingual, graph-based approach to Entity Linking and Word Sense Disambiguation based on a loose identification of candidate meanings coupled with a densest subgraph heuristic which selects high-coherence semantic interpretations. Babelfy is based on the BabelNet 3.0 multilingual semantic network and jointly performs disambiguation and entity linking in three steps:

- It associates with each vertex of the BabelNet semantic network, i.e., either concept or named entity, a semantic signature, that is, a set of related vertices. This is a preliminary step which needs to be performed only once, independently of the input text.
- Given an input text, it extracts all the linkable fragments from this text and, for each of them, lists the possible meanings according to the semantic network.
- It creates a graph-based semantic interpretation of the whole text by linking the candidate meanings of the extracted fragments using the previously-computed semantic signatures. It then extracts a dense subgraph of this representation and selects the best candidate meaning for each fragment.

2 – Proposal
Figure 2 depicts the necessary process in order to achieve a meaningful result towards the improvement of a semantic-layered system able to dwell deep in content extraction and cross referencing. Similarly, harvest such a system to provide an automated system for an updated research architecture filled with full semantic meaning with respect to online research publications.

As notified in the previous section, crawling and indexing are two of the major concepts that allow the composition of the proposed system.

2.1 – Overall Process Overview

\textsuperscript{15} \url{http://babelfy.org/about}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{Evolutionary Processes}
\end{figure}
The overall evolution of the system in answer for the main concern is the one depicted in figure 2, where knowledge will undergo four different steps to later on be indexed in the storing engine provided by Elastic Search. Needless to say, each step cannot be executed whilst the previous one has not been finished even if each step is run by different scripts.

2.1.1 – Step 1: Crawling Knowledge
First and foremost, in order to gain a semantic layer with respect to scientifically published knowledge, that information needs to be crawled and then stored. More specifically, data from the ACL repository will be stored containing relevant material concerning online research documentation as shown in the following illustration 3.

2.1.2 – Step 2: Downloading Content
Once information is available as non-downloaded content, online documentation from the repository is targeted for downloading to then storing the PDF document in a local folder.

2.1.3 – Step 3: Content Transformation
In order to make the downloaded files more comfortable to work with, all the inward content is transformed from sentences into a key valued architecture system in which the key set will try and capture every meaning of the processes sentence such as to which section does it belong to, or the semantic concept of the sentence.

In other words, extract all the relevant content and save those chunks in a semantically built-up file generated using the Doctor Inventor explained later on.

2.1.4 – Step 4: Content Indexation
Thus, when a document is processed to become more computer-readable, that document is ready to be indexed in the Elastic Search indexing engine, and that is the main purpose for the fourth step.

16 http://aclweb.org/anthology/
2.2 – Architecture Overview

Following, the different processes shown in figure 4 have been created starting from cero, and these scripts are about to be thoroughly explained. In other words, I make the following claim: the different sub-systems explained later are custom made except for the use of JSoup and Dr. Inventor.

2.2.1 – Database Server

The selected database server plays a big role in the whole architecture, because it is the central pillar of information from which the three Java clients (WebSeek, WebDeliver and DocConvertible) will be fed relevant data pivotal for their functioning.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weburl</td>
<td>String</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
</tr>
<tr>
<td>Abbr</td>
<td>String</td>
</tr>
<tr>
<td>Year</td>
<td>String</td>
</tr>
<tr>
<td>State</td>
<td>Integer</td>
</tr>
<tr>
<td>hasBIB</td>
<td>Boolean</td>
</tr>
<tr>
<td>Size</td>
<td>Integer</td>
</tr>
<tr>
<td>Path</td>
<td>String</td>
</tr>
</tbody>
</table>

*Table 2 - SQL Issue Table Description*

Table 2 shows the separate columns belonging to the MySQL data server corresponding to the meta-information every data provider (WebSeek, WebDeliver and DocConvertible) script is going
to both extract and either insert or update knowledge from. The idea is to mirror the relevant information from the ACL anthology web page and by utilizing few memory space, compress the targeted selection of issue knowledge. All three data provider programs will share the same determined schema database but, they operate from data having different states.

Weburl states the URL of the document soon to be downloaded from the WebDeliver program. Name, abbr, and year are parameters contained in the ACL anthology web URL that represent information about the event being crawled by WebSeek. State is one of the two cornerstone parameters of the database since the Java client programs demand information extraction based on different state values. When state equals to cero, it means that WebSeek has just crawled and extracted the information of the issue. The value of state will be one when WebDeliver has finished downloading and transporting the content of the PDF document to the web service. If DocConvertible has funneled the PDF content using Doctor Inventor framework and stored accordingly through GlassFishWS, the web service will update its state to the numeric value of two.

Boolean value hasBIB only will be true if in the ACL anthology page w.r.t. a specific document contains bibliographic information to crawl upon, otherwise its value will be false.

Size variable belongs to the amount of kilobytes that define the document retrieved from the ACL anthology, and its purpose is to make the work of DocConvertible focus on funneling documents into Doctor Inventor starting from the lightest of documents. Later on Doctor Inventor will be explained in full, but for now in short, the less heavy a document the quicker the semantic content is processed and stored.

And finally, to make things a little easier for the middleware, the local path of the document is saved into the database to quicken and avoid calculating the directory in which the document is contained.

2.2.2 – GlassFishWS
The center of the architecture is the element denominated GlassFishWS, which is a web service coded in java that uses the glassfish open source application server. This component acts as a middleware between the java client elements (WebSeek, WebDeliver and DocConvertible) and the undeveloped ESearch web page, forwarding requests from one sub-system to another. Every request the aforementioned elements call will be captured in the web service, whilst the main purpose is to redirect the requests to the MySQL database or the Elastic Search index engine.

In other words, whenever a program needs to store or extract data from either of the last two mentioned applications (SQL and Elastic Search), the content of these petitions will be funneled into the data stores. Likewise, the glassfish application is in charge of forwarding any of the CRUD solicitations (Create, Read, Update, and Delete) to the SQL data server as well as advancing any Elastic Search request.

Every web service operates on a certain URL and contains several resources available to the clients. In this case, and as it has already been mentioned, the resources have been knowingly prepared to avoid any confusion between sub-components.

11
2.2.3 – WebSeek
A single-threaded system instructed to direct its crawling abilities and latch onto the determined web page specified in its configuration file. With this extra capability, WebSeek becomes fully modular in the sense that its efforts are prearranged in its configuration document.

It was originally designed to crawl and fetch data from the ACL anthology web page\(^\text{17}\), so it will undoubtedly manage to extract data from an online repository other than ACL.

On the other hand, its configuration file contains the following variables that need fulfilling, otherwise WebSeek will not operate. In other words, every variable must contain relevant content in order for the sub-system to even initialize, or the launch will be stopped.

To maintain modularity, WebSeek needs to know which the global URL is so as to be able to afford dropping data into the web service as well as being able to extract meaningful data parts for the correct proceeding of crawling.

Opposite to custom developments, and according to state-of-the-art research in the background set, a crawler ought to compellingly integrate good practices in order to maintain a low profile whilst keepings its crawling impact very little. In other words, a crawler should be able to fetch targeted data as the web page is not crippled. More specifically, a politeness time interval is essential in coping up with the recommendations which specifies the suggested waiting time before the next data extraction is processed.

As a second good practice notice, the freshness element of the crawler is specified in the configuration file as a variable in minutes w.r.t the frequency the crawler should fetch again information in order to be updated. Furthermore, since WebSeek is supposed to be an infinite looped web crawler, every iteration will be fired after waiting the exact quantity of minutes specified by whomever updates the configuration file.

Finally, two more variables populate the configuration file. The first one named crawl will target the name of the events published in the ACL anthology home page where TACL (Transactions of the Association for Computational Linguistics) and NACL (North American Chapter of the ACL). Finally, these events contain several documents which are not relevant to the system that contain information about the issue but will not amount to anything since the content has nothing to do with research but with the outline of the issue, which will be captured in the exclusion filtering variable in the configuration file.

Its only function is to latch onto the ACL anthology web page and traverse through the selected issues in the aforementioned configuration file variable named crawl, and store any non-fetched meta-information about newly uploaded research documents through the previously explained web service GlassFishWS.

2.2.4 – WebDeliver
Also a single threaded script, this program requests information to the middleware about research documents stored in the SQL database to check whether there exists any issue whose document has yet not been downloaded, i.e. whose state equals to cero.

\(^{17}\) [http://aclweb.org/anthology/](http://aclweb.org/anthology/)
Once a valid link is retrieved from the web service, WebDeliver will proceed to fetch the document belonging to the meta-information. Whilst the recently downloaded PDF file maintains temporary, WebDeliver will encapsulate the research document and send it to the web service where GlassFishWS will execute two tasks in sequence.

The first one will transform the encapsulated data chunks back to a PDF document and save it in a specified location, and the second one will update the state in the issue database to set the meta-information of the issue to the value one, i.e. when its document has been successfully stored locally.

And, WebDeliver will keep on requesting more meta-information also dictated by a custom configuration file to maintain its flexibility if any part of the system undergoes any change in its shared variables such as the link to the web service etc.

Its configuration file is very similar to the process described before WebDeliver, WebSeek since they follow the same pattern, that of the web crawler. However, the only difference is that there is no freshness good practice variable in here, as WebDeliver will periodically ask for documents to download instead of seeking information. Yet, a politeness parameter is hardcoded instead of giving the user freedom to incorporate its numeric amount. Instead, the waiting time is provided to be formatted by the user to its will to accommodate for any number that will fit it.

Lastly, the final parameter needs previously checking the ACL anthology home page. WebDeliver provides specific target to download the documents contained in any year and issue the user deems fit to be processes in the system. The format for this parameter is as explained here: contained in block quotes, the abbreviation of the issue goes first followed by any sequence of years separated by a comma and the abbreviation and the years are split by the separating character ‘:’. If more than one issue is deemed relevant enough to be downloaded, another block quote will be appended to the previous one, should there be one, separated also by a comma. An example of this pattern would be this. [Issue abbr1: year1, year2], [issue abbr2: year1] and targeting specific ACL anthology issues [ACL: 14, 15, 16], [TACL: 12].

2.2.5 – DocConvertible

Comparable to the WebDeliver process, DocConvertible will also request meta-information to the middleware to then forward data chunks from a document. The main difference resides in the use of a framework, besides the purpose of the program, that being to transform downloaded PDF documents into a semantically structured key value oriented file type called JSON (JavaScript Object Notation). JSON is mostly known as the XML simplified version removing the need of opening and closing tags replaced with key quotes to specify a new value inside a class, and block quotes to resemble multiple objects of the same class. In other words, a complex but effective and scalable pattern for data exchange.

DocConvertible will, as WebDeliver does, request meta-information about any issue. But in this case, the tiniest of differences causes the biggest of impacts in the process of the whole system. The requested information will be one which complies with the condition of its state being downloaded, i.e. that WebDeliver has previously downloaded and updated its state from zero to one.
Instead of adding more processes to the web service, DocConvertible will be in charge of downloading the file again, but in spite of storing it temporarily, the requested PDF data chunks will be assembled by DocConvertible and funneled straight into the Doctor Inventor framework to extract the semantic content and stick these data parts according to a JSON pattern.

4 – Results

This chapter presents the experimentation performed in order to test the implemented system. First we detail the experimental setup to thereafter give way to four separated use cases to embody different approaches of interests.

4.1 Experimental Setup

Though the setup of the Elastic Search indexed documents is not constant through the years, i.e. all the years do have different amount of indexed documents, the total amount ascends to thirty-one hundred and twenty two (3122 in numbers) to test both the gathered research collection alongside with the built up architecture in order to give a response to the aforementioned main research concern. These documents have been acquired firstly from the ACL anthology web page, and then managed to extract the semantic content using the Doctor Inventor just explained.

4.2 Use Case 1: Most Repeated Concepts

Figure 5 - Most Repeated Concept Graphic

More of a general purpose illustration, it shows the most repeated concepts mentioned all across the document corpus indexed in Elastic Search. Useful though, for ground knowledge concern on the most written and researched areas.
4.3 Use Case 2: Most Repeated Concepts over the years

As shown in figure 6, the most mentioned topics throughout the years differ from the previous use case just because this time we look at the title and not across the whole document content of the total corpus.

This figure illustrates how the mentioning of the dissonant synset ids are plotted over the years between 2005 and 2015 referencing the number of separate repetitions of specific knowledge. Pretty beneficial to see growing or decreasing tendencies, and if preferences begin to tilt or start to rise up.

4.4 Use Case 3: Specific Content information

As mentioned in the Doctor Inventor sub-section, the framework is able to extract the discourse category of the sentence, and Figure 7 plots the different keynote speeches all along every sentence of each document of the whole collection whose title contains the keyword “parsing”.

Seemingly profitable graphic, shows a deeper level of evolution of the parsing concept over the separate years. As per the illustration, it is made clear that parsing area even if is still huge, approach is still predominant in these gathered document collection.
4.5 Use Case 4: Another Specific Content information

Another example on the discourse distribution, but in figure 8, the concept is “deep learning”. As the graphic illustrates, the interest for deep learning has grown over the years where it got a massive spike from 2012 onward.

5 – Conclusion
This thesis aims to present a system with the capabilities of both browsing and analyzing scientific documents. For that purpose, we have researched for state of the art components in order to mesh them together in creating the presented architecture. The deployed system is developed for indexing online publications to a deeper knowledge discovery of the contents of the research publications.

After quite some exhaustive testing, we found out that the proposed system is able to perceive and gather online documentation in real time. Not only this, once the document content has been fetched, the knowledge is then queued for transformation and storing.

The use cases were able to be assembled and performed successfully since every process ended up providing their respective delivery prosperously.

5.1 – Future Work
In short, the future research could be summarized in the following items:

- Extend the crawler to further repositories with the intention of gathering more knowledge.
- Develop a search oriented web page to retrieve information closely similar to the use cases.
References


List of Figures

FIGURE 1 - WEB CRAWLER ARCHITECTURE .................................................................................. 3
FIGURE 2 - EVOLUTIONARY PROCESSES .................................................................................. 8
FIGURE 3 - ACL ANTHOLOGY HOME PAGE .............................................................................. 9
FIGURE 4 - PROCESS INSTANTIATIONS ....................................................................................... 10
FIGURE 5 - MOST REPEATED CONCEPT GRAPHIC .................................................................... 14
FIGURE 6 - MOST REPEATED CONCEPTS OVER THE YEARS GRAPHIC .................................... 15
FIGURE 7 - PARSING DISCOURSE GRAPHIC .............................................................................. 15
FIGURE 8 - DEEP LEARNING DISCOURSE GRAPHIC .................................................................. 16

List of Tables

TABLE 1 - ELASTIC SEARCH VS. SQL ...................................................................................... 6
TABLE 2 - SQL ISSUE TABLE DESCRIPTION ............................................................................ 10
TABLE 3 - USE CASE 1 NUMERIC VALUES ............................................................................... 23
TABLE 4 - USE CASE 2 NUMERIC VALUES ............................................................................... 26
TABLE 5 - USE CASE 3 NUMERIC VALUES ............................................................................... 29
TABLE 6 - USE CASE 4 NUMERIC VALUES ............................................................................... 31
Appendix
A – Mapping
{
  "upfcollection": {
    "mappings": {
      "jsondoc": {
        "properties": {
          "citations": {
            "properties": {
              "citationMarkers": {
                "properties": {
                  "citationId": {
                    "type": "long"
                  },
                  "id": {
                    "type": "long"
                  },
                  "referenceText": {
                    "type": "string"
                  },
                  "sentenceld": {
                    "type": "long"
                  }
                }
              }
            }
          }
        },
        "citationString": {
          "type": "object"
        },
        "id": {
          "type": "long"
        },
        "text": {
          "type": "string"
        }
      }
    },
    "header": {
      "type": "nested",
      "properties": {
        "authorList": {
          "properties": {
            "firstName": {
              "type": "string"
            },
            "fullName": {
              "type": "string"
            }
          }
        }
      }
    }
  }
}
"abstract": {
    "type": "boolean"
},
"babelSynsetsOcc": {
    "type": "nested",
    "properties": {
        "babelURL": {
            "type": "string"
        },
        "coherenceScore": {
            "type": "double"
        },
        "dbpediaURL": {
            "type": "string"
        },
        "golbalScore": {
            "type": "double"
        },
        "id": {
            "type": "integer"
        },
        "inSentenceId": {
            "type": "long"
        },
        "numTokens": {
            "type": "long"
        },
        "score": {
            "type": "double"
        },
        "sentenceldWithTerm": {
            "type": "long"
        },
        "source": {
            "type": "string"
        },
        "synsetID": {
            "type": "string"
        },
        "text": {
            "type": "string"
        }
    }
},
"citationMarkers": {
    "properties": {
        "citationId": {
            "type": "long"
        }
    }
}
B – Use Cases

B.1 – Use Case 1

B.1.1 – Query
GET http://localhost:9200/upfcollection/_search
{
  "size": 0,
  "aggs": {
    "nest_sentences": {
      "nested": {
        "path": "sentences"
      }
    },
    "aggs": {
      "nest_babelSynsetsOcc": {
        "nested": {
          "path": "sentences.babelSynsetsOcc"
        }
      },
      "aggs": {
        "top_synsetIds": {
          "terms": {
            "field": "sentences.babelSynsetsOcc.synsetID"
          }
        }
      }
    }
  }
}

B.1.2 – Tabular Result

<table>
<thead>
<tr>
<th>Synset ID</th>
<th>Total Repetitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>7,755</td>
</tr>
<tr>
<td>Sentence</td>
<td>7,318</td>
</tr>
<tr>
<td>Set</td>
<td>5,938</td>
</tr>
<tr>
<td>Parsing</td>
<td>5,848</td>
</tr>
<tr>
<td>Word</td>
<td>5,376</td>
</tr>
<tr>
<td>System</td>
<td>5,076</td>
</tr>
<tr>
<td>Map</td>
<td>4,511</td>
</tr>
<tr>
<td>Algorithm</td>
<td>4,414</td>
</tr>
<tr>
<td>Training</td>
<td>4,399</td>
</tr>
<tr>
<td>Project</td>
<td>4,236</td>
</tr>
</tbody>
</table>

*Table 3 - Use Case 1 Numeric Values*

B.2 – Use Case 2

B.2.1 – Query
GET http://localhost:9200/upfcollection/_search
B.2.2 – Tabular Result

<table>
<thead>
<tr>
<th>Year</th>
<th>Synset ID</th>
<th>Document Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Parsing</td>
<td>76</td>
</tr>
<tr>
<td>Year</td>
<td>Parsing</td>
<td>Data sets</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>2006</td>
<td>172</td>
<td>120</td>
</tr>
<tr>
<td>2007</td>
<td>102</td>
<td>100</td>
</tr>
<tr>
<td>2008</td>
<td>99</td>
<td>112</td>
</tr>
<tr>
<td>2009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Cognition</td>
<td>2013</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>Parsing</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>Data sets</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>Knowledge</td>
<td>278</td>
</tr>
<tr>
<td></td>
<td>Training</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>Number Theory</td>
<td>333</td>
</tr>
<tr>
<td></td>
<td>Cognition</td>
<td>323</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Cognition</th>
<th>2014</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Parsing</td>
<td>185</td>
</tr>
<tr>
<td></td>
<td>Data sets</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>Knowledge</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>Training</td>
<td>268</td>
</tr>
<tr>
<td></td>
<td>Number Theory</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>Cognition</td>
<td>305</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Cognition</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Parsing</td>
<td>143</td>
</tr>
<tr>
<td></td>
<td>Data sets</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>Knowledge</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>Training</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>Number Theory</td>
<td>206</td>
</tr>
<tr>
<td></td>
<td>Cognition</td>
<td>200</td>
</tr>
</tbody>
</table>

**Table 4 - Use Case 2 Numeric Values**

### B.3 – Use Case 3

#### B.3.1 – Query

```json
GET http://localhost:9200/upfcollection/_search
{
    "size": 0,
    "aggs": {
        "nest_sentences": {
            "filter": {
                "nested": {
                    "path": "sentences",
                    "query": {
                        "nested": {
                            "path": "sentences.babelSynsetsOcc",
                            "query": {
                                "term": {
                                    "sentences.babelSynsetsOcc.synsetID": {
                                        "value": "15790106n"
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```
B.3.2 – Tabular Result

<table>
<thead>
<tr>
<th>Year</th>
<th>Discourse Class</th>
<th>Sentence Repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Approach</td>
<td>8,726</td>
</tr>
<tr>
<td></td>
<td>Outcome</td>
<td>1,234</td>
</tr>
<tr>
<td></td>
<td>Background</td>
<td>1,217</td>
</tr>
<tr>
<td></td>
<td>Unspecified</td>
<td>784</td>
</tr>
<tr>
<td></td>
<td>Challenge</td>
<td>216</td>
</tr>
<tr>
<td></td>
<td>Future Work</td>
<td>165</td>
</tr>
</tbody>
</table>

2006
<table>
<thead>
<tr>
<th>Year</th>
<th>Approach</th>
<th>Outcome</th>
<th>Background</th>
<th>Unspecified</th>
<th>Challenge</th>
<th>Future Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>21,040</td>
<td>3,272</td>
<td>3,123</td>
<td>2,121</td>
<td>566</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td>11,448</td>
<td>1,698</td>
<td>1,658</td>
<td>1,129</td>
<td>328</td>
<td>213</td>
</tr>
<tr>
<td>2008</td>
<td>10,816</td>
<td>1,696</td>
<td>1,872</td>
<td>1,163</td>
<td>257</td>
<td>194</td>
</tr>
<tr>
<td></td>
<td>13,630</td>
<td>2,071</td>
<td>2,027</td>
<td>1,265</td>
<td>404</td>
<td>213</td>
</tr>
<tr>
<td>2011</td>
<td>14,255</td>
<td>2,840</td>
<td>2,865</td>
<td>2,028</td>
<td>512</td>
<td>294</td>
</tr>
<tr>
<td></td>
<td>13,894</td>
<td>2,033</td>
<td>2,319</td>
<td>1,553</td>
<td>341</td>
<td>242</td>
</tr>
<tr>
<td>2013</td>
<td>22,088</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 5 - Use Case 3 Numeric Values

<table>
<thead>
<tr>
<th>Year</th>
<th>Approach</th>
<th>Outcome</th>
<th>Background</th>
<th>Unspecified</th>
<th>Challenge</th>
<th>Future Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014</td>
<td></td>
<td>2,875</td>
<td>3,851</td>
<td>2,273</td>
<td>597</td>
<td>362</td>
</tr>
<tr>
<td></td>
<td>22,245</td>
<td>3,033</td>
<td>3,877</td>
<td>2,063</td>
<td>579</td>
<td>304</td>
</tr>
<tr>
<td>2015</td>
<td></td>
<td>20,573</td>
<td>3,269</td>
<td>2,230</td>
<td>474</td>
<td>306</td>
</tr>
</tbody>
</table>

B.4 – Use case 4

B.4.1 – Query

GET http://localhost:9200/upfcollection/_search

```json
{
  "size": 0,
  "aggs": {
    "nest_sentences": {
      "filter": {
        "nested": {
          "path": "sentences",
          "query": {
            "nested": {
              "path": "sentences.babelSynsetsOcc",
              "query": {
                "term": {
                  "sentences.babelSynsetsOcc.synsetID": {
                    "value": "02601968n"
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```
B.4.2 – Tabular Result

<table>
<thead>
<tr>
<th>Year</th>
<th>Discourse Class</th>
<th>Sentence Repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Approach</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>Outcome</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Background</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Unspecified</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Challenge</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Future Work</td>
<td>1</td>
</tr>
<tr>
<td>2007</td>
<td>Approach</td>
<td>87</td>
</tr>
<tr>
<td>Year</td>
<td>Approach</td>
<td>Outcome</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6 - Use Case 4 Numeric Values