Report BHA-2018-09: A SPARQL Query Search Agent for Finding Web Resources and Creating NPDS Semantic Metadata Records Describing Them

Shiladitya Dutta, Adam Craig and Carl Taswell

***Abstract*—Semantic descriptions are a prerequisite for many of the goals laid out for the Nexus PORTAL-DOORS System, however it is labor-intensive to manually annotate resources from the open-web. To remedy this, we have created CoVaSEA: an automated web crawler/query engine that can search and expand NPDS metadata records. With this system, articles from external biomedical databases can have semantic descriptions extracted from them for use in a variety of computational tasks.**

***Index Terms*—Nexus-PORTAL-DOORS System, CoVaSEA, SPARQL, web-crawler, semantic web**

Introduction

The Nexus-PORTAL-DOORS System (NPDS) is a software system that provides the capability to publish both semantic and lexical resources regarding specific target areas. NPDS has inbuilt REST API services via the Scribe Registrar for record search, retrieval, and publication along with a standardized messaging protocol, enabling exchange among client applications and servers. All remote components are organized according to the Hierarchically Distributed Mobile Metadata architectural style**taswell2010distributed**. The structure of the Nexus PORTAL-DOORS System is sub-divided, in a method analogous to IRIS-DNS, into 3 primary parts: Nexus, PORTAL, and DOORS. The PORTAL registry controls entities that have unique labels and their associated lexical meta-data, which can come in the form of tags, vocabulary terms, or cross references. The DOORS directory contains the semantic meta-data which is primarily comes in the form of RDF descriptions. The Nexus diristry (an aggregation of the terms DIRectory and regISTRY) is a single server that serves as a combination of the PORTAL registry and DOORS directory.**taswell2008doors** The NPDS project also has several other features built in such as Hypothesis and Concept-Validating ontologies, each of which serve to avoid large and cumbersome ontologies,**skarzynski2015solomoncraig2017bridging** and also the capability of having individuals maintain their own Nexus, PORTAL, or DOORS servers. Semantic descriptions are a prerequisite for many of the goals laid out for the Nexus PORTAL-DOORS System**taswell2008portal**, however it is labor-intensive to manually annotate resources from the open-web. To remedy this, we have created CoVaSEA: an automated web crawler/query engine that can search and expand NPDS metadata records. With this system, articles from external biomedical databases can have semantic descriptions extracted from them for use in a variety of computational tasks.

Document created June 20, 2018, revised September 30, 2018; typeset December 30, 2018.

Methods

As an expansion to the past section**baeexpanding**, CoVaSEA integrates several features which benefit NPDS including: **(A)** An implementation of SPARQL query based semantic search to allow retrieval and manipulation of DOORS linked data descriptions **(B)** Targeted web-crawling for relevant article metadata from external biomedical literature databases to expand NPDS records **(C)** Automated translation of free-form text abstracts into RDF triples**lassila1999resource** to derive the semantic representations of lexical data. The system

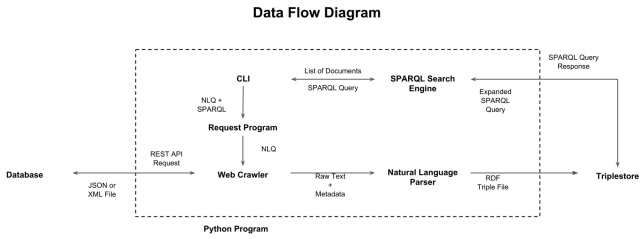

Fig. 1. Data flow diagram of the project and all of its included components

consists of a pipeline in which a web crawler registers articles by converting the abstract text and description to RDF, so that a SPARQL query engine can comb through the retrieved material. The web crawler component searches for articles and their metadata in biomedical literature databases via REST API. Each article has its basic metadata(title, abstract, author(s),

TABLE I
A TABLE OF THE DIFFERENT FEATURES OF THE SEMANTIC AND LEXICAL COVASEA

	Semantic	Lexical	Both
Internal	DOORS Directory	PORTAL Registry	Nexus Diristry
External	Semantic Webcrawler Natural Language to RDF Parser	Lexical Webcrawler	Webcrawler
Both	Semantic Search Engine Natural Language to SPARQL Query UI (User Interface)	Lexical Search Engine UI (User Interface)	UI (Possibly)

etc.) returned. In order to translate the lexical information in the abstract into semantic metadata, CoVaSEA uses Stanford Parser**manning2014stanford** and NLTK**bird2009natural** to develop RDF triples from the unstructured text of the articles' abstracts. First, constituency parsing is performed to create a tree from which the subject(s), predicate, and object(s) are recorded in a method similar to Rusu D et al.**rusu2007triplet**. Then co-reference resolution and pronomial anaphora occur to recognize unique entities and ensure that their references are consistent. Once the logical-form triple post-processing is finished, they are converted from lexical-based triples to valid RDF by identifying each part of the subject-verb-object triples. This is accomplished by using various databases (i.e. MeSH) for field-specific "jargon" and select named entities, word sense disambiguation for standard words, and literals for numerals and names. Finally, the RDF graph is converted to the turtle format, yielding a file that portrays the natural-language information in each abstract in a linked data format. These turtle formatted files are then stored in a DOORS directory via the Scribe API. When a SPARQL query is called the program retrieves the graphs from the database to be queried via the SPARQL query engine. In terms of the structure of the program itself, the web crawler/search engine program consists of a number of interdependent components. Each of these components fulfills a general purpose. The methods are:

- **Web Crawler** : Crawls through a multitude of sites in order to retrieve data. Currently the system can search through the article records on DOAJ, PubMed, Elsevier ScienceDirect, and CORE**knoth2012core**. The webcrawler is similar in structure to Seung-Hong Bae**baeexpanding**. The webcrawler receives a requested query and then it searches through the available databases via REST API. The general method for searching through the databases is first a general lexical search query via the provided natural language query. The database then returns a number of articles. From there, the crawler searches through the list of returned articles and retrieves the basic-metadata from each one. It then passes the basic meta-data to the natural language parser. The basic meta-data constitutes of the abstract, title, author(s), database of origin, DOI(if available), and date of publication. The abstract will have its triples extracted and the rest will be included in the named graph of the article via the DublinCore ontology **weibel1997dublin**. It should be noted that the general search query returns twice the amount of requested queries by the user. This is due to the fact that, in general, the provided lexical search API doesn't have filtering capabilities. This means that we receive many articles that aren't suitable for a variety of reasons (i.e. unsupported format or different language than English). Thus we have to remove those, but in order to retrieve the amount of articles requested we must ensure that we have enough that it is more or less assured that only one general query will retrieve enough valid articles.
- **Natural Language Parser**: Receives the raw text from the web crawler and then parses it into logical form triples. Then from the logical form triples they are parsed into RDF/XML triples and put into a graph.

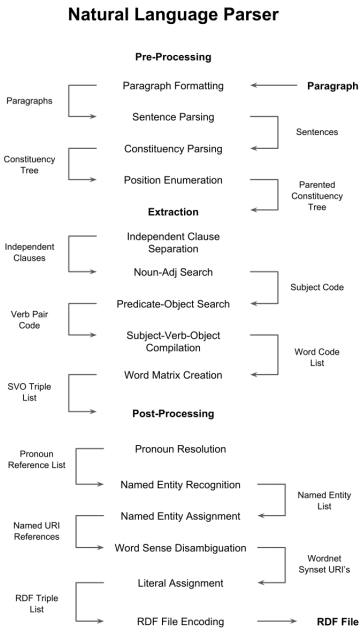

Fig. 2. Data flow diagram of the Natural Language to RDF parser

- Pre-processing: Designed in order to pre-process the raw text to prepare it for parsing and also to perform tasks that depend on the raw texts such as Named Entity Recognition**finkel2005incorporating** and Co-Referencing**clark2015entity**. Pre-processing preparation includes removing troublesome characters, re-spacing the text to ensure homogeneity, and parsing the paragraphs into sentences.
- Extraction: Uses a constituency parse to extract relevant logical form triples. First independent clauses are extracted by separating out the phrases that have a subject and a verb phrase. Note that this system of separation only works on clauses that are completely separate and not clauses that are nested within one another or are interlaced. First a constituency parse occurs on each sentence. Then breadth-first and depth-first search are used to find subject and verb-object phrases. The breadth-first search is used to find the highest noun phrase in the tree. This noun phrase is then split up via conjunctions and any proximal adjectives are linked to the nouns. Then a two stage breadth first, depth first search is used to find the verb. Breadth first search is used to find the highest verb phrase. Then depth first search is used to find the verb specifically so that it can be separated from the prepositional or adjective section of the verb phrase. The verb-object phrases are paired since in general the verb and object need to be specifically related by a link in order to maintain their relative positions. These phrases are then broken into their individual parts and tagged with positions. For the nouns, any proximal adjectives are put into a specialized adjectives list. This is so that they can be directly linked to their representative nouns.
- Post-processing: This step translates the logical form triples into RDF triples. The step primarily consists of graph compression. It does this by assigning URI's to each part of the logical form triples. First "jargon" and some named entities are assigned URI's via databases. Then standard words are assigned via WordNet**miller1995wordnet** synsets using context and part of speech**toutanova2000enriching**. This step is important since it applies to most of the parts of the sentence and is vital to capturing the full meaning behind the sentence. This also is extremely useful in graph compression so that the graph isn't purely lexical-based but rather has references that are broad. The algorithm that is used in order to perform this word sense disambiguation is Lesk**banerjee2002adapted**. Finally named entities and number are assigned to literals. These are all then packaged into a named graph.

- **SPARQL Analysis Engine**: Receives SPARQL query from user and then searches through the triple store to extract relevant data for the user. In order to increase ease-of-use for a user a feature is implemented to allow for constructing a SPARQL query in normal conjunctive form. This is where a series of statements conjoined by AND/OR statements create a query. However, this system doesn't allow yet for the same level of detail that can be achieved by inputting a raw SPARQL query.
- **Triplestore/QuadStore**: This triple store is used in order to store RDF triples that describe documents. The triplestore/quadstore consists of named graphs (in the .ttl format) each describing an article. The graphs consists of two different sections. The first section describes the generic metadata of the article and the second section describes the semantic meaning of the abstract.
- **CLI**: The CLI (Command Line Interface) is how the user interacts with the program. It includes various features to interface with the program. However, its primary objective is to get input and display output. It will take a SPARQL Query and search query for an input and return a list of relevant documents as an output. The CLI interface consists of 6 different options. The first is to perform a full external SPARQL Query search utilizing the web crawler and the SPARQL query engine. The second option is just to expand the stores via the web crawler. The third option is to search the existing stores via SPARQL. The fourth option is to change the environmental variables which will be a necessity for first time setup. Their are three variables to change: the Java installation, StanfordNLP installation, and location of local stores. The fifth command is to reprint the previous commands and the sixth command is to exit the program.

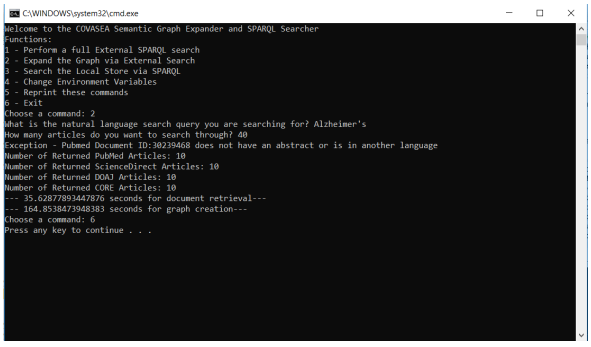

Fig. 3. A screenshot of the CLI interface

- **Interface Component**: The central component is the juncture between the Command Line Interface and the other components. The central component handles interface display and also acts as a center point to pass arguments to the other components.

These items integrate to form a whole system. First the user inputs a Natural Language Query(NLQ) and a SPARQL query into the CLI components. This is then passed to the interface component which calls the webcrawler. The webcrawler goes through the various databases and retrieves the raw text and the metadata which is passed to the natural language parser. On each articles, the natural language parser transforms the text and metadata into semantic triples. These triples are then stored in .ttl files in a triplestore/quadstore. Then the interface components calls the SPARQL query engine. The SPARQL query engine searches through the triplestore/quadstore and returns a list of articles which is outputted to the user via the CLI. The system is built in Python via the Visual Studio Code environment. The format of the project is such that each of the major sections, natural language parser, web crawler, unit test, central interface, and SPARQL query engine, each have their own Python class. The classes, with the exception of the central interface, don't call each other in a pipeline, but rather the central interface calls all of the individual classes and passes data between them. However, the interface doesn't change anything so effectively the flow is between the components. The system uses a variety of external code libraries in order to complete its objectives. NLTK and the Stanford Parser are used in the natural language parser. NLTK is used primarily for the tree manipulation, sentence parsing, Part-Of-Speech(POS) tagging, and the word sense disambiguation. The Stanford Parser is used for constituency parsing, co-reference resolution, and named entity recognition. It should be noted that the Stanford Parser is used from Java by instantiating a server. The system also uses RDFLib in order to manipulate the RDF triples. RDFLib is used in order to extract, build, and search the RDF files. The system currently uses turtle for its RDF format. The web crawler uses requests to make and format the REST API calls and xmljson in order to convert returned XML to JSON for easy searching. The system uses the default unittest system to perform unit testing on all of the components.

Results

Note that all tests are run on a computer with 16 GB of RAM and a six-cored i7-8750H with a base-clock of 2.20 GHz. The process uses only one core. The connection speeds are a download speed of 18.40 Mgbs (Megabytes per second) and an upload speed of 1.68 Mgbs. One result in particular is that the setting for the only key triples and all triples is similar. This is a setting in which if on only the key triples or triples describing subject-object relations are transferred and those triples that refer to subject-adjective relations are filtered out. The run time is probably similar due to the selection process and the added time needed to process the extra triples balancing out.

Unit testing has been performed on all methods in the CoVaSEA library to ensure functionality and to facilitate test-driven development as described in Software Design, Development, and Coding **dooley2017software**.

Fig. 4. Screenshot of Unit Testing Results. Some results are cutoff by the Resource Exceptions

Discussion

The primary point of possible expansion for the system is the Natural-Language Parser. Currently, the parser faces difficulty in 3 main areas: relevance of triples, accuracy of triples, and runtime. The first problem is the relevance of triples and how good they summarize the arguments. There are several methods which we can take inspiration from**leskovec2004learningshang2011enhancing**. Most of these focus on document summarization by extracting the semantic graph structure of the document. Most of these approaches use some form of pattern recognition to identify potentially valuable triples. It is possible to use a system similar to this, though it would require a training set to be effective. The second issue is of sentence format. The system currently can only extract triples from a small set of sentence formats. Thus it is unable to take on sentence formats that are more advanced in nature (i.e. "In spite of a lack of operational knowledge, which could be expected of any animal, the monkey was able to miraculously, and without any training, operate the machine"). In the previous example the parser will not be able to extract the nested statements. The best way to remedy this is to use a semantic frame solution such as FrameNet**baker1998berkeley** or Boxer**bos2008wide**, which was the approach taken by FRED**gangemi2017semantic**. The final problem is that of optimization. The runtime for the parser in its current state is relatively extreme in that it takes 30 seconds to parse a paragraph. This impedes us from performing whole articles parses with reasonable runtime. The best way to solve this is to either use more advanced hardware or rewrite certain sections of code to optimize. The next section of improvement is the SPARQL query engine. The SPARQL query engine relies upon local graph download which could lead to large run times later down the road. This leads to runtime constraints with large graphs. This could be remedied by implementing the capability to access distributed

TABLE II
RUN TIMES AND OUTPUTS OF CoVaSEA

Function	Input	Result	Run Time
Full (Internal + External)	Number Requested = 20 Query = Dementia, SELECT DISTINCT ?author ?title WHERE {?a dc:contributor ?author . ?a dc:title ?title}	20 files 20 results	245.120 seconds
	Number Requested = 100 Query = Dementia, SELECT DISTINCT ?author ?title WHERE {?a dc:contributor ?author . ?a dc:title ?title}	100 files 98 results	1307.42 seconds
SPARQL Internal	Query = See Above # of Articles = 225	220 results	5.731 seconds
Web Crawler Expansion	Number Requested = 20 Query = Parkinson's	20 files	102.324 seconds
	Number Requested = 100 Query = Parkinson's	100 files	483.126seconds
Triple Extraction	Only Key Triples = True Sentence = Parkinson's is a serious brain disorder. It takes multiple years to develop. The cause of Parkinson's is a lack of dopamine receptors in the brain. Research has been done to cure and diagnose Parkinson's.	3 triples	5.388 seconds
	Only Key Triples = False Sentence = Parkinson's is a serious brain disorder. It takes multiple years to develop. The cause of Parkinson's is a lack of dopamine receptors in the brain. Research has been done to cure and diagnose Parkinson's.	7 triples	5.699 seconds

graphs**quilitz2008querying**. In addition SPARQL query expansion should be implemented to increase the breadth of SPARQL searches and to implement a reasoning engine such as RACER**haarslev2003racer** or F-OWL**zou2004f**.

Conclusion

Here we presented a system in which resources from the open-web can be translated into machine-understandable semantic information and be searched via SPARQL. CoVaSEA has the capability to both search "externally" with the web crawler for semantic data to expand the NPDS knowledge base and "internally" with SPARQL to provide a method to navigate the data inside the DOORS directory. With the distinct advantage that the system is automated, thus can furnish large amounts semantic descriptions on a regular basis, CoVaSEA lays the groundwork for a variety of future NPDS applications for which linked data stores are a necessity.

Acknowledgment

I give my thanks to the other members of the Brain Health Alliance 2018 program for their assistance on technical and methodological points.