Parsing Text into RDF graphs[*]

Análisis semántico de textos para derivar grafos RDF

Brahim Batouche
CNRS/LORIA
Nancy, France

Claire Gardent
CNRS/LORIA
Nancy, France

Anne Monceaux
Airbus Group Innovation
Blagnac, France

Resumen: Proponemos un método basado en RDF para consultas sobre el contenido de un texto. En primer lugar, el texto es alineado a tuplas RDF usando herramientas de Procesamiento de Lenguage Natural (PLN). Luego, el contenido del texto puede ser interrogado a través de consultas expresadas en SPARQL. Aplicamos la estrategia sobre un conjunto de reglas y mostramos que el sistema permite recuperar aquellas reglas que satisfacen un requerimiento dado.
Palabras clave: La semántica, RDF, SPARQL, extracción de información

Abstract: We propose a RDF-based method for querying the content of a text. Text is first mapped to RDF triples using existing Natural Language Processing tools. The text content can then be queried using SPARQL queries. We apply the approach to a set of rules and show that the resulting system permits retrieving those rules that satisfy a given information need.
Keywords: Semantics, RDF, SPARQL, Open Domain Information Extraction

1 *Introduction*

Semantic parsing maps natural language text into computer interpretable meaning representations. It differs from related tasks such as semantic role labelling or deep parsing in that its output is in formal language and is meant for computers to read. In this setting, evaluation is task based. A semantic interpretation is correct if it yields the correct response with respect to the application making use of it.

Earlier semantic parsing systems include (Woods, 1973) and (Warren y Pereira, 1982) which were both developed to query databases in natural language. Because these systems relied on manually coded lexicons, grammars and semantic interpretation rules, they lack portability and were costly to develop and maintain however.

More recent work on semantic parsing eschews the manual authoring of linguistic resources by exploiting training corpora associating natural language sentences with meaning representations to learn a semantic parser from aligned data (Zettlemoyer y Collins, 2012; Kate y Mooney, 2006). Like the earlier systems though, these new semantic parsers are domain dependent in that they are trained on domain specific corpora of aligned text and formal representations e.g., an air travel information (ATIS corpus), geography data (Geoquery) or robot instructions (robocup). Porting them to a new domain requires building a new training corpus which again is time consuming as it is difficult to obtain semantic representations for sentences.

In contrast, recent work on open domain semantic parsing has focused on mapping natural language utterances into logical forms to be executed on large knowledge bases (Berant et al., 2013; Cai y Yates, 2013; Bordes et al., 2011; Bordes et al., 2011) since it drives applications such as question answering (QA) and information extraction (IE). Most of this work however focuses on factoid questions with a modest amount of compositionality typically, questions whose meaning can be represented by a single RDF triple (e.g., *Who directed TopGun?*).

In this paper, we present a semantic parsing approach which maps text to RDF triples. The resulting RDF database is combined with a domain ontology and can be queried using SPARQL queries. In this way, the text interpretation can be used to support arbitrary queries about the text while the integration of the text semantics with domain knowledge permits a seamless integration with reasoning (e.g., querying the database for information about pipes and retrieving in-

[*] The work reported in this paper was partially funded by the ITEA ModelWriter Project

ATA	Zone	Rule	Object	Auxiliary	Action verb	Prep	Object 2
38	1	1	Pipe	shall be	segregated	from	electrical route

Table 1: Semi-Structured Rule

formation that was given by the text about hose pipe i.e., a subclass of the class being queried).

We develop a proof-of-concept implementation and evaluate the approach in a context where system engineers need to retrieve from a set of documents the information relevant to the task they are working on. Given a large set of documents, system engineers often need to retrieve from these documents all the rules, requirements or specifications that are specific to the task they are addressing. We focus on a case where the documents are regulations describing the installation design principles of a system or a set of systems and where the engineer must retrieve from an existing set of regulation documents all the information relevant to the design of a new system. In this setup, the meaning representations extracted from text must support the information needs of the engineers. For instance, they must allow the user to retrieve all design principle rules that apply to the various parts of the system they are designing. We illustrate by means of examples, how the representations produced support different kind of queries; and how integrating text meaning and domain knowledge permits answering queries whose response is only implicit in the text.

The paper is structured as follows. We start by motivating our approach and explaining the applicative setting in which it is used (Section 2). We then go on to present our approach (Section 3) and illustrate its working using a run through example. In Section 4, we show various types of example queries which can be answered given our interpretation process and the added domain knowledge. Section 6 concludes with pointers for further research.

2 Motivation

Installation design principle documents incorporate the various sets of regulations and directives about how to install a system or a set of systems in a functional area (e.g., electrical and optical system or Water Waste System). For each aircraft project, a set of such documents must be produced to ensure that the resulting system comply with the system requirements and take into consideration applicable regulations and procedures. To facilitate access to the relevant information, the initial approach has been to manually build a rule database which currently contains installation rules normalised in a semi-structured format. For instance, the rule shown in Figure 1 describes a segregation constraint holding between a pipe and an electrical route. This constraint is specified by Rule 1 and applies in Zone 1 of the functional area ATA38 (i.e., the water waste system). While the creation of this relational database facilitates the identification of relevant information, this initial approach has several shortcomings.

First, the database model is fixed and can not easily be changed in case it does not, or no longer satisfy the information needs of the users. RDF has no such limitations.

Second, databases do not integrate easily with knowledge bases and reasoning services. In contrast, RDFS provides a base level integration of RDF triples with knowledge based reasoning by allowing for *class*, *subClass*, *domain* and *range* declarations. OWL based reasoning is also supported which provides additional expressivity and permits describing the relationships and structure of entire domains.

A third drawback is that the manual construction of the database is time consuming and costly.

By converting rules to RDF triples, our approach addresses each of these shortcomings. Because the meaning representations of rules are not constrained by a database schema, they can be freely modified and adapted to suit the information needs of the users. The RDF format permits an easy integration with basic reasoning services. And the use of NLP techniques to automatically convert text to RDF triples avoids the need for manual conversion of the rules. Because we process the normalised rules (from the existing data base) rather than the input documents, we do not fully address this last point. However, by proposing a method for mapping text to RDF graphs, we provide a first step towards converting the documents into a meaning representation (an RDF knowledge

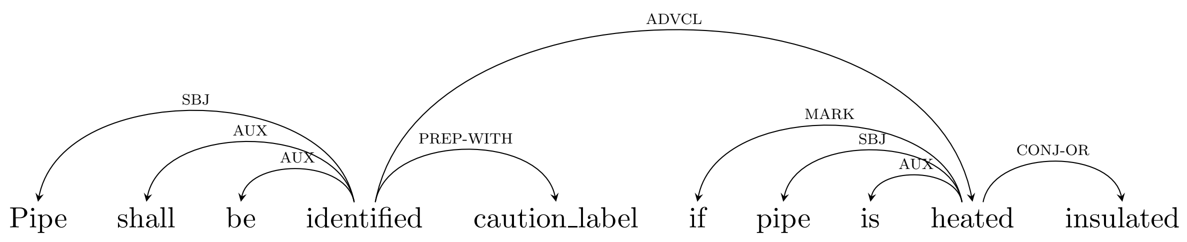


Figure 1: Example Dependency Tree of Rule 14

base) that can be queried, which has a well defined semantics and which can easily be combined with domain specific knowledge.

3 Approach

As mentioned above, in a first step, we process the normalised rules contained in the database rather than the design principle documents themselves. Indeed, extracting rule information from documents involves additional processes such as information extraction (identifying those parts of the document which describe rule related information such as the aircraft zone a rule applies in, the ATA area it belongs to and the rule itself) and anaphora resolution (e.g., resolving pronouns or implicit anaphors). Here we focus on the semantic parsing task (i.e., how to map text to a computer interpretable meaning representation) and leave the question of how to extract rule information from full document open for further research.

To convert text to RDF representations, we combine existing NLP and Semantic Web technology into the pipeline architecture depicted in Figure 3. The remainder of this section explains each of the processing steps involved and illustrates them using a running example.

3.1 Preprocessing

First, the text fragments associated with a design principle rule are extracted from the database and stored into a text file. For instance, the semi-structured rule shown in Figure 1 is extracted from the corresponding row and stored into a text file as follows:

(1) *Rule: Pipe shall be segregated from electrical route*
ATA:38, Zone 1, Rule 1

Note that the rule is stored separately but that the additional relevant information stored in other database attributes (i.e., ATA functional area, zone and rule identifiers) is preserved.

Next rules are tokenised and terms are recognized and grouped into single units. For instance, in Example (1), the bigram *electrical route* is recognised as a term and marked as such so that the parser will treat it as a word. Terms may be recognised in different ways. They may either be listed in a file by a domain expert or extracted automatically from the corpus of rules using term extraction techniques (Nakagawa y Mori, 2002). In addition, synonyms may be grouped together so that variants of a given term can be normalised to a single representative term ideally, the label of the corresponding concept in the domain ontology. In this first experiment, we allow for two ways of detecting terms. We either list them in a file relying on the expert knowledge provided by the industrial or we extract them automatically from the corpus relying on n-gram frequency that is, we assume that frequent n-grams involving content words are domain specific terms. In addition, we use a predefined list of stop n-grams to filter out frequent n-grams that are function words (e.g., *in order to*).

3.2 Parsing

Once rules have been tokenized and terms have been recognised, rules are parsed using the Stanford dependency parser (Klein y Manning, 2003). Figure 1 shows the dependency parse tree of Rule 14. In this tree, each node is labelled with a word of the input sentence and edges between nodes are labelled with grammatical relations (e.g., *sbj* for nominal subject or *prep-with* for a prepositional object introduced by the preposition *with*). Preposition (e.g., *with*) are “folded into” the arc label rather than labelling a node as is the case for all other words in the input sentence.

```
mdwr="http://localhost:8080/Rdf_example#"
<rdf:Description rdf:about="http://localhost:8080/Rdf_example#pipe">
      <mdwr:segregate-from rdf: resource="http://localhost:8080/Rdf_example#electrical-route"/>
      <mdwr:identify-with>"label"</mdwr:identify-with>
</rdf:Description>
```

Figure 2: RDF Triples extracted from the Natural Language triples `(pipe, segregate-from, electrical-route)` and `(pipe, identify-with, label)`.

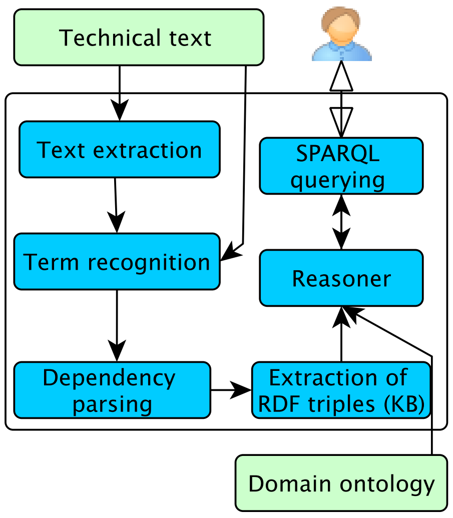


Figure 3: Architecture of the Semantic Parsing and Querying Process

3.3 Extracting triples

Because they directly capture functor/arguments and modifier/modifiee relations, dependency structures are often viewed as approximating semantic representations of a sentence content (Candito y Kahane, 1998). Here we exploit this feature to extract `(s,p,o)` triples based on the dependency relations holding between words and/or terms. We manually specify a set of rules mapping subtrees of dependency parse trees into RDF triples. For instance, if the dependency parse tree contains a subtree of the form:

$$\begin{tikzcd}
S & V \arrow[l, bend right, "\text{SBJ}"'] \arrow[r, bend left, "\text{PREP-WITH}"] & O
\end{tikzcd}$$

we produce an RDF-triple of the form:

(2) `(S,V,O)`

For instance, given the dependency tree shown in Figure 1, the extracted triples will be (the preposition is adjoined to the verb):

```
(pipe,identify-with, caution_label)
(pipe,heat,dummy)
(pipe,insulate,dummy)
```

Note that the extracted triples only capture the predicate/argument dependencies expressed by the natural language rule. In particular, both the disjunction relation between the second and third triples and the condition relation between the first and the latter two triples are not captured. In other words, the semantic representations obtained are more similar to those produced by semantic role labellers (Carreras y Màrquez, 2005; Palmer, Gildea, y Xue, 2010) than to deep parsing semantics. To produce a more complete semantic analysis of the input text, additional mapping rules would need to be defined. Alternatively these could be learned from an annotated corpus. We leave the question of a more complete RDF interpretation of natural language open for further research and focus instead on showing how RDF representations can be used to provide a computer interpretable meaning representations of natural language which can support querying and reasoning about the meaning of a text.

3.4 Constructing RDF Graphs

To support SPARQL queries and RDFS reasoning, the triples extracted from the dependency trees are further converted into XML formatted, RDF statements as illustrated in Figure 2.

An RDF statement consists of a subject, a property and an object. The subject of an RDF statement is either a uniform resource identifier (URI) or a blank node, both of which denote resources. They are not directly identifiable from the RDF statement. The predicate is a URI which also indicates a resource, representing a relationship. The object is a URI, blank node or a string literal.

Given a triple `(s,p,o)` extracted from the dependency tree, we build an RDF statement by mapping `s` to a subject URI, `p` to a property and `o` to either a URI or a literal depending on whether `o` is the subject of another statement: if it does, `o` is mapped to a URI, otherwise, it is mapped to a literal.

Recall though (cf. Figure 1), that the normalised rules we start from contain additional information e.g., information about the ATA functional area or the zone introducing a rule. This additional information is also converted to RDF triples as illustrated in Figure 5. E.g., an ATA area is related to a zone by an *hasZone* relation, a zone to a rule by an *hasRule* relation and a rule to a serie of elements by an *element* relation. In this way, the information contained in the additional database fields can be taken into account when querying the OWL knowledge base. The complete KB schema is shown in Figure 5. Figure 4 shows the RDF interpretation of Rule 14.

```
(ATA-38,has_rule, rule-14)
(rule-14,has_zone, zone-14)
(rule-14,element, pipe)
(pipe,identify-with, caution_label)
(pipe,heat,dummy)
(pipe,insulate,dummy)
```

Figure 4: Full interpretation of Rule 14

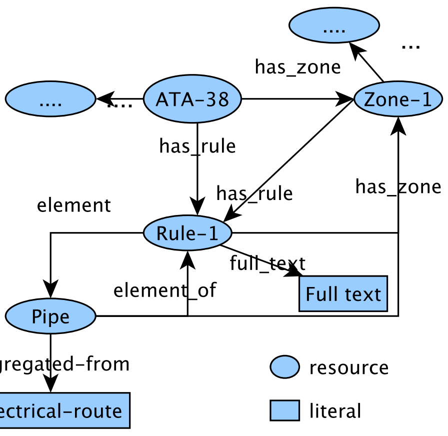


Figure 5: Graph of knowledge base

3.5 Adding Domain Knowledge

While the translation of the normalised rules into RDF triples provides a computer interpretable representations of these rules content, complementing these representations with domain knowledge permits extending the set of queries that can be answered by supporting e.g., taxonomical reasoning. As mentioned in Section 1, the addition of such domain knowledge permits, for intance, retrieving a rule about *hose pipes* while querying for rules about *pipes*.

RDF data can be annotated with semantic metadata (e.g., classes and subclasses information) using two principal syntaxes: RDFS (Pérez, Arenas, y Gutierrez, 2006) and OWL (McGuinness, Van Harmelen, y others, 2004) with RDFS a sublanguage of OWL. We use OWL to complement the RDF data extracted from the textual rules with a taxonomic classification provided by the experts (in a different setting, the required classification could be retrieved e.g., from the web or automatically constructed from relevant textual or formal data). For instance, we define *hose pipe, electrical pipe* and *water pipe* to be subclasses of the *pipe* class.

4 *Querying the Knowledge Base*

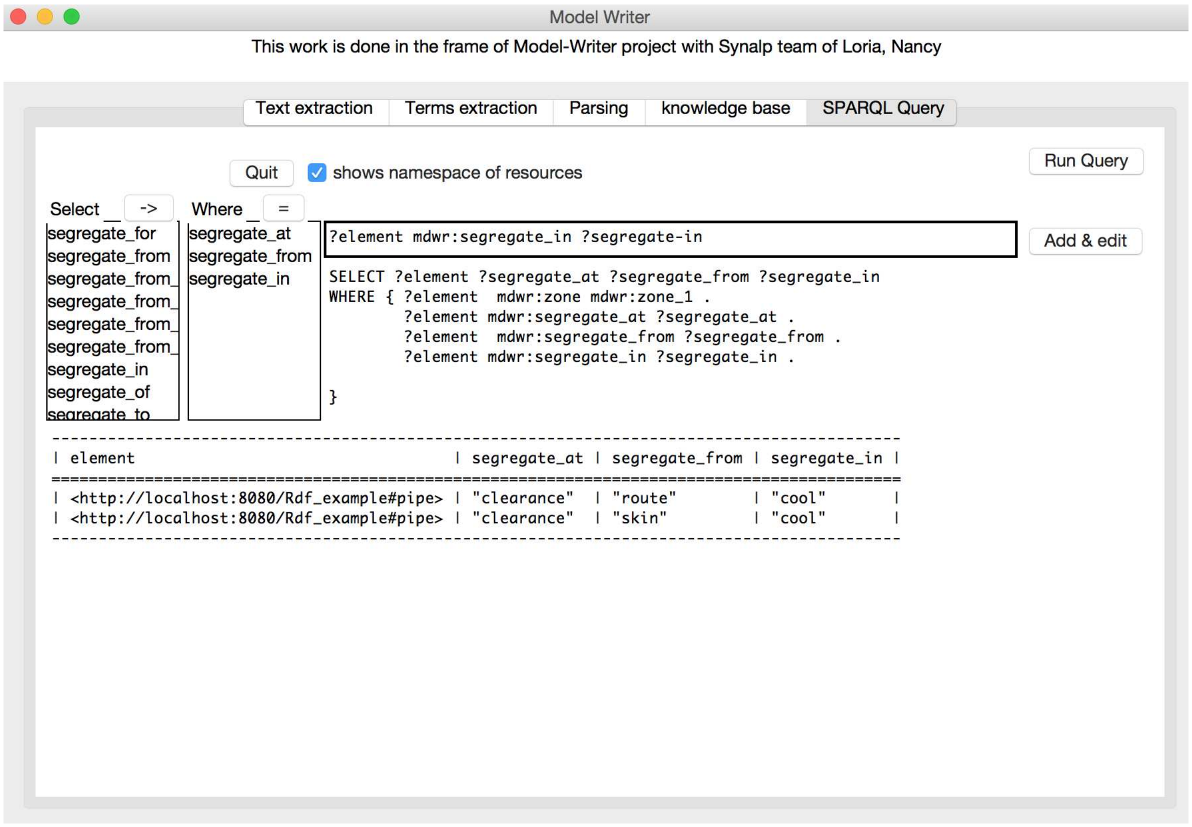
RDFS and OWL databases can be queried using the SPARQL query language (Pérez, Arenas, y Gutierrez, 2006). Like SQL, SPARQL selects data from the query data set by using a SELECT statement to determine which subset of the selected data is returned. The WHERE clause is used to define graph patterns to find a match for in the query data set. Figure 2 shows some example queries and their results. As the example shows, the RDF model of textual content naturally supports complex and varied queries on the combined content of the rules, the additional information contained in the database (ATA area, Zone, etc) and the domain ontology. For instance, query 6 describes how to select an element and its properties values, where the element is related to `zone-1` and `rule-1` and the properties are: `segregate-from`, `segregate-at` or `segregate-in`.

5 *Experiment*

The approach described in the preceding section was implemented and evaluated on a small sample of 115 normalised rules. Figure 6 shows a screenshot of the system graphical interface. Using this interface, the user can run the different subprocesses involved namely, extraction of text from a database, sentence tokenization, term extraction and recognition, dependency parsing, construction of the RDF database and SPARQL

	Query	Gloss	Example Answer
1	SELECT *?element* WHERE {*?element* :has-zone :zone-26 .}	*Elements related to zone 26*	*movement, hose*
2	SELECT *?element* WHERE {*?element* :element-of :rule-1. }	*Elements related to rule 1*	*pipe*
3	SELECT *?element ?rule* WHERE {*?element* :has-zone :zone-1 ; :element-of *?rule*.}	*Rules applying to zone 1 and their elements*	*(pipe, (rule-6, rule-15, rule-5, rule-4, rule-13, rule-11 ...))*
4	SELECT *?element ?zone* WHERE {*?element* :element-of :rule-21 ; :has-zone *?zone*}	*Zones of objects mentioned in rule 21*	*(hose, (zone-21, zone-77, ...))*
5	SELECT *?element ?has-zone* WHERE {*?element* :has-zone *?has-zone* . FILTER (*?has-zone* = :zone-1) }	*Elements belong in zone 1*	*pipe*
6	SELECT *?element ?segregate-at ?segregate-in ?segregate-from* WHERE {*?element* :segregate-at *?segregate-at*; :segregate-in *?segregate-in* ; :segregate-from *?segregate-from*; :element-of :rule-1 ; :has-zone :zone-1 . }	*properties of element belong in zone-1 and related to rule-1*	*(pipe,(segregate-at(clearance), segregate-in(cool), segregate-from(route, skin)))*

Table 2: Example SPARQL Queries. Namespaces are omitted for brevity



This work is done in the frame of Model-Writer project with Synalp team of Loria, Nancy

```
?element mdwr:segregate_in ?segregate-in
```

```
SELECT ?element ?segregate_at ?segregate_from ?segregate_in
WHERE { ?element  mdwr:zone mdwr:zone_1 .
        ?element mdwr:segregate_at ?segregate_at .
        ?element  mdwr:segregate_from ?segregate_from .
        ?element mdwr:segregate_in ?segregate_in .

}
```

```
-------------------------------------------------------------------------------------------
| element                                   | segregate_at | segregate_from | segregate_in |
===========================================================================================
| <http://localhost:8080/Rdf_example#pipe> | "clearance"  | "route"        | "cool"       |
| <http://localhost:8080/Rdf_example#pipe> | "clearance"  | "skin"         | "cool"       |
-------------------------------------------------------------------------------------------
```

Figure 6: Screenshot of the application

querying.

The overall architecture is fully generic and could be used for any text independent of its domain. The application is implemented with Python 3.4, where the used packages are:

- Term extraction: Nltk (v.3.0.1) and Xlrd (v.0.2.1)
- Parsing: Stanford-parser (v.3.3.9)
- Reasoning: Jena (v.2.13.0)

For our proof-of-concept experimentation, we used a dataset of 115 rules. In average, each rule is mapped by the semantic parsing process to 5.91 triples (min:1, max:7). To support reasoning we complemented the RDF graph extracted from the rules with a domain ontology which we created manually. The ontology is composed of 24 classes, of which 17 are subclasses. We tested the resulting RDFS knowledge base using 20 SPARQL queries of which 6 are shown, together with their result, in Figure 2.

6 Conclusion

While RDFizers are commonly used to convert various data format to RDF (Hwang et al., 2014), little work has been done on how to map natural language to RDF and on how to exploit such a mapping. In this paper, we have presented an end-to-end proof-of-concept process illustrating how natural language can be mapped to RDF triples and how the existing link between RDF, RDFS and OWL could be exploited to support reasoning about text content using knowledge not contained in text (e.g., ATA information provided by an external database and a taxonomy of the domain being considered). The resulting approach permits querying text content while taking into account this external knowledge i.e., permits merging knowledge from different sources.

One distinctive feature of our approach is that we use dependency trees as a basis for constructing RDF triples. This is a natural move as dependency tree encodes functor/argument and modifier/modifiee relations. Thus constructing a RDF triple `(s,p,o)` from a dependency subtree `p(s,o)` amounts to encoding the dependency tree syntactic/semantic functor `p` as an RDF property and its children `s,o` as RDF resources related by the `p` property. That is, a syntactic functor/modifiee is mapped to a property and its arguments/modifiers to resources related by that property. Exploiting this natural mapping between dependency trees and RDF triples allows for a simple and linguistically principled semantic parsing process. Modification and addition to the base semantic construction procedure (i.e. to the mapping between dependency trees and RDF triples) can be defined at will, based on linguistic information and on the requirements of the application domain. For instance, although we did not process determiners, these might be relevant in another context. Their semantics could be captured by specifying an additional mapping rule mapping e.g., a det-noun subtree to an RDF triple of the form `(n, det, d)` where `n,d` are the noun and the determiner and `det` is the property relating them. The interpretation of such a triple would then be determined by the reasoner used to interpret the RDF graph. Similarly, unary modifiers could be encoded as datatype properties.

As mentioned in Section 3.3, the RDF meaning representations currently extracted by our approach are limited to functor/argument dependencies. To derive a more complete representation of the meaning of natural language sentences, the set of mapping rules used to convert dependency subtrees to RDF triples need to be extended. This could be done either manually (using e.g., graph rewriting techniques (Bédaride y Gardent, 2011)) or automatically. It would in particular be interesting to explore how a parallel or comparable corpus of text and RDF triples could be built from which mapping rules could be learned.

Further directions for future work include conducting a large scale evaluation of the meaning representations obtained (Do they support the queries required by the end application namely the retrieval by engineers of the information that is relevant to a given task ?) and designing a robustness mechanism to counter parse errors. Indeed if the parse tree is incorrect so are the RDF triples. Although the Stanford Parser has a high accuracy on newspaper text (92%), accuracy decreased on out of domain data. Another interesting direction for further research would be to explore ways of automatically detecting and correcting parsing errors in order to improve the semantic representations obtained.

Bibliografía

Bédaride, P. y C. Gardent. 2011. Deep semantics for dependency structures. En *Computational Linguistics and Intelligent Text Processing*. Springer, páginas 277–288.

Berant, J., A. Chou, R. Frostig, y P. Liang. 2013. Semantic parsing on freebase from question-answer pairs. En *EMNLP*, páginas 1533–1544.

Bordes, A., J. Weston, R. Collobert, y Y. Bengio. 2011. Learning structured embeddings of knowledge bases. En *Conference on Artificial Intelligence*, numero EPFL-CONF-192344.

Cai, Q. y A. Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. En *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics (ACL)*, páginas 423–433. Citeseer.

Candito, M.-H. y S. Kahane. 1998. Can the tag derivation tree represent a semantic graph? an answer in the light of meaning-text theory. En *Proceedings of the Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks*.

Carreras, X. y L. Màrquez. 2005. Introduction to the conll-2005 shared task: Semantic role labeling. En *Proceedings of the Ninth Conference on Computational Natural Language Learning*, páginas 152–164. Association for Computational Linguistics.

Hwang, J., H. Jung, S. Yoo, y S. Park. 2014. Design of an rdfizer for online social network services. En *Future Information Technology*. Springer, páginas 449–454.

Kate, R. y R. Mooney. 2006. Using string-kernels for learning semantic parsers. En *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, páginas 913–920. Association for Computational Linguistics.

Klein, D. y C. Manning. 2003. Accurate unlexicalized parsing. En *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, páginas 423–430. Association for Computational Linguistics.

McGuinness, D., F. Van Harmelen, y others. 2004. Owl web ontology language overview. *W3C recommendation*, 10(10):2004.

Nakagawa, H. y T. Mori. 2002. A simple but powerful automatic term extraction method. En *COLING-02 on COMPUTERM 2002: second international workshop on computational terminology-Volume 14*, páginas 1–7. Association for Computational Linguistics.

Palmer, M., D. Gildea, y N. Xue. 2010. Semantic role labeling. *Synthesis Lectures on Human Language Technologies*, 3(1):1–103.

Pérez, J., M. Arenas, y C. Gutierrez. 2006. Semantics and complexity of sparql. En *The Semantic Web-ISWC 2006*. Springer, páginas 30–43.

Warren, D. y F. Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122.

Woods, W. 1973. Progress in natural language understanding: an application to lunar geology. En *Proceedings of the June 4-8, 1973, national computer conference and exposition*, páginas 441–450. ACM.

Zettlemoyer, L. y M. Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*.