Building ontologies from textual resources: A pattern based improvement using deep linguistic information

Sami Ghadfi, Nicolas Béchet and Giuseppe Berio

IRISA, UMR 6074, Université de Bretagne-Sud, 56017 Vannes, France
sami.ghadfi@gmail.com, nicolas.bechet@irisa.fr, giuseppe.berio@univ-ubs.fr

Abstract. Ontologies are a key component for several applications. Ontologies are often built by hand, but automatizing the process of ontology building has been and is even more recognized as very important for scaling and speeding up this process. However, several difficulties have been identified, some of them are quite fundamental. In this paper, we present our work for overcoming some of the fundamental difficulties. Our work resulted in improvements of an existing ontology building tool (*Text2Onto*). The contribution of our work consists in the creation of a flexible language (DTPL—Dependency Tree Patterns Language) for expressing patterns as syntactic dependency trees to extract semantic relations, and making an existing ontology building tool (*Text2Onto*) able to use them. DTPL allows to exploit deep linguistic information (related to co-reference resolutions, conjunctions, appositions, passive verbal phrases, etc.) provided by deep syntactic analysis of the text, and also (in order to improve the accuracy of patterns) to express the exclusion of some dependency bindings in patterns.

Keywords: ontology building, semantic relation extraction, dependency tree patterns, deep linguistic information, Text2Onto, DTPL.

1 Introduction

Ontologies are a key component for several applications. Ontologies are often built by hand, but automatizing the process of building ontologies has been and is even more recognized as very important for scaling and speeding up this process. Indeed, humans employ texts for providing information directly or indirectly, through the Web for instance. However, unstructured or semi-structured texts do not provide a well-defined semantic structure to be used by machines for reasoning tasks. Ontologies play therefore the key role for representing more explicitly the knowledge hidden in texts. As a consequence, ontologies can be made available for further applications.

Unfortunately, several difficulties concerning automatic ontology building have been identified, some of them are quite fundamental.

Additional arguments suggesting the need for developing complete "Ontology Building Support Systems" (OBSS) can be mentioned. Despite the fact that humans can recognize *ontology artifacts* from terms and sentences (which is enabled by their knowledge of the domain and the contexts on which terms are put together in sentences, suggesting semantic relations between terms), OBSS can supply the frequent

terms and the contexts in which they appear, and systematically apply rules for suggesting how they are related to ontology artifacts. The magnitude of these terms and contexts makes their identification a task more suitable for machines than humans. In addition, ontologies evolve and these evolutions should be supported by automated systems.

For ontology building, there are two main challenges to be taken into account, which correspond to the basic building blocks of any ontology:

- The extraction of concepts and their possible instances: it is a task in which we further distinguish between the extraction of the concept/instance itself, and naming it;
- The extraction of semantic relations (hierarchical and non-hierarchical): it is a task in which we distinguish between identifying the relation occurrence (for example, identifying the relation occurrence "*lion,animal*"), and then identifying the semantic relation to which it belongs ("*lion,animal*" is an occurrence of a hyponymy relation, the whole relation occurrence can be rewritten as *is-hyponym-of(lion,animal)*).

Even if these two challenges are partially connected (i.e. the extraction of relations may impact on already extracted concepts and instances or may lead to additional concepts and instances), in this paper, we concentrate on the second one, i.e. semantic relation extraction. However, as better explained in Section 2, concept/instance extraction and relation extraction can be treated separately. This is also confirmed by the fact that tools used or usable for concept/instance extraction are developed independently for performing well identified tasks such as terminology extraction (possibly comprising disambiguation) and entity identification.

Semantic relation extraction methods can be categorized into two approaches: Pattern based (mainly employing linguistic patterns), and Clustering based (mainly employing clustering and statistical methods). We consider that linguistic patterns are natural and concrete (because close to what humans (can) apply when they manually build ontologies – by following methodologies and design patterns) for improving the overall ontology building process, thus, we have focused on pattern based approaches for relation extraction for the following detailed reasons:

- Patterns represent frequent contexts in which term-pairs related by a given semantic relation tend to appear—the reason for this observation is the way patterns are constructed; very often, this construction begins by specifying seed examples (term-pairs related by a given semantic relation), then looking for the contexts –in sentences– in which they tend to appear together (these contexts can be sequences or sets of words [1,10], or dependency paths in syntactic dependency trees [12], [11]), and then generalizing/merging the most similar contexts or keeping only the most accurate ones (i.e. contexts relating at least a given number of instance examples)—; for instance, the Hearst pattern " X(NP[1]) such as Y(NP) " [5] induces the relation "Y is-hyponym-of X", where the context in this case is the sequence of words "such as".
- Patterns fall into two categories: 1. Reliable patterns (they possess high precision and low recall), 2. Generic patterns (they possess low precision and high recall). One can use the advantages of one category to overcome the drawbacks of the oth-

[1] NP represents a noun phrase.

er. For instance, in [8], the authors have used reliable patterns as a reference to evaluate the relevance of relation occurrences extracted by generic patterns.
- Any extraction method and technique that does not use predefined patterns takes more processing time, because it needs to identify the (frequent) contexts in which terms related by a given relation do appear (for instance, these contexts can be syntactic dependency links that bind individual words in the text—as used in [2] and [9]—, etc.). These methods are based on the distributional hypothesis [4] and its derivations [15], [6], [7].

However, effective usage of patterns within an OBSS remains an open research question. In Section 2, we present the existing methods for pattern-based relation extraction, and also their inherent difficulties (or limitations) preventing to get acceptable ontologies. Section 3 presents the contributions of the paper, i.e. (I) A method for enhancing OBSS with the ability to use deep linguistic information for relation extraction, (II) Making the generation of relations (including how relations can be named) through patterns very flexible, and (III) Implementing this method within an existing ontology building tool (*Text2Onto*). We finally conclude by summarizing the contributions and presenting perspectives in Section 4.

2 Difficulties

Willing to semi-automatically build ontologies (or to support ontology building as best as possible) starting form texts, improvements can be concentrated on:

- Improving the input text by modifying (substituting) the employed terms (e.g. for adopting a more standard terminology) and sentence structures, resolving ambiguities and co-references and so on;
- Improving the quality of the final ontology by performing a quality assessment (e.g. using reasoning, if applicable, similarity (and other) measures) followed by relevant modifications;
- Improving the process of building the ontology by improving the efficiency and effectiveness of the required tasks (i.e. relation extraction, concept/instance extraction, etc.).

In this paper, we focus on the third line of improvements, and more specifically, (as said in the Introduction) on relation extraction, because, as explained in section 2.1 below, concept/instance extraction can be performed independently from relation extraction. Section 2.2 presents the inherent difficulties in using pattern-based approaches for extracting semantic relations and related work.

2.1 Reasons for processing relation extraction and concept/instance extraction separately

Although concept/instance extraction and relation extraction are two partially dependent tasks, they can be treated separately. A formal justification can be presented as follows, on the top of a hypothetical ontology Description Logics formalization; whenever a relation (role) R is newly introduced, additional axioms involving existing concepts can be added. Generally speaking, introducing R can result in 3 situations:

- Additional specification for an existing concept C e.g. $C \sqsubseteq \exists R.\top \sqcap \forall R.\top$ or $C \sqcap \exists R.\top \sqcap \forall R.\top \neq \bot$;
- Splitting an existing concept in subconcepts C', C'' such as, for instance, $C \sqsubseteq C' \sqcap C''$, $C' \sqsubseteq \exists R.\top \sqcap \forall R.\top$;
- Creating a new concept C' such that $C' \sqsubseteq \exists R.\top \sqcap \forall R.\top$.

These few arguments should convince the reader that the extraction of relationships can be modularly managed as well. As a consequence, addressing only the difficulties concerning relation extraction is not a limitation; it even contributes in a well-defined modular way to improve concept/instance extraction.

2.2 Relation extraction methods using flat patterns and the inherent difficulties

Pattern-based relation extraction methods often concern hyponymy and part-of relations [8]. These methods often use patterns expressed as *flat regular expressions* (Flat patterns), which contain basic syntactic information (like part of speech tags, lemmas, affixes, etc.). The most known and successful example of using flat patterns is Hearst patterns [5], which are used for extracting the *hyponymy relation* (or IS-A/subsumption relation when using the standard ontology terminology). Because of their high precision, Hearst patterns have been used even in clustering-based relation extraction methods: for instance, in [2], Hearst patterns have been used to name the clusters of a hierarchy of terms based on the hyponymy relation (a hyponymy hierarchy is close to an IS-A taxonomy). In [10], a similar approach has been used for naming the clusters of a hyponymy hierarchy.

Another successful use of flat patterns is using reliable patterns to correct the extraction results of less accurate patterns [8].

Java Annotation Patterns Engine (JAPE), a language of the open-source platform General Architecture for Text Engineering (GATE[2]), has been the key language for expressing flat patterns. With JAPE, flat patterns are expressed as transducers (using macros, input and output annotations) to annotate sentences in the text that match the pattern. Transducers are organized in queues corresponding to sentences in which, the results (output annotations or macros) of one phase can be used as inputs by the next one. A relevant usage of JAPE can be found in *Text2Onto* [3] (an ontology building tool), where GATE is used as the key library for preprocessing. *Text2Onto* preprocessing tasks involve some of GATE's components such as the Part Of Speech (POS) tagger, the named entity extractor, and also patterns made by the user.

Using flat patterns has been successful for extracting semantic relations, but such patterns suffer from two major limitations that we point out hereafter.

The absence of deep syntactic information in flat patterns leads to misinterpretations when these patterns are matched to the text. Consider the following sentences: (s1) "*The semantic formalization of knowledge has been achieved by the use of several tools such as ontologies, semantic networks and expert systems.*"; (s2) "*Euclid, a great mathematician in his own right, showed to a king that there is no royal road to geometry.*". In these sentences, the comma can play two roles, i.e. a conjunction in (s1), or an introducer of apposition in (s2) (in (s2) the apposition is "great mathematician"). Another example of cases leading to misinterpretations is when the syntactic

[2] A full documentation on GATE can be found at http://gate.ac.uk/documentation.html

structure of the text (having impact on its semantic interpretation) cannot be efficiently and effectively captured by flat patterns. This includes cases like verb phrases expressed in active or passive form, or discontinuity cases (topicalization, etc.).

Flat patterns contain often unnecessary symbols for relation extraction, which often reduce the patterns coverage. It is the syntactic information conveyed by symbols that should be identified: in the example above (sentences (s1) and (s2)), what is interesting is to know whether the comma symbol represents a conjunction or an apposition. Another example is in the Hearst pattern *P*: " <Hypernym>(NP) **including** <Hyponym>(NP) ". The flat pattern *P* can be applied successfully to extract the hyponymy relation instance "specie is-hyponym-of organism" (r1) from the sentence (s3) "*Organisms including species like flies, yeast, monkeys and worms have previously been put on diets and shown to have their life spans extended by 30 to 200%.*". However, if we insert the adjective *diverse* between *including* and *species* in the sentence (s3) (which results in the sentence (s4) *"Organisms including **diverse** species like flies, yeast, monkeys and …"*), then *P* does not match anymore. However, the semantic relation (r1) should have been extracted from both sentences. Adding an adjective between the word *including* and the hyponym in *P* is not necessary (from the semantic view) to identify the hyponymy relation.

2.3 Dependency tree patterns

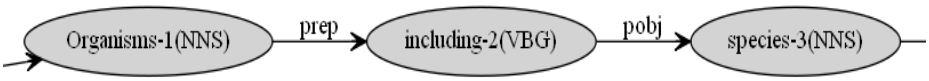

Fig. 1. (t3) A sub-tree of the syntactic dependency tree of the sentence (s3)

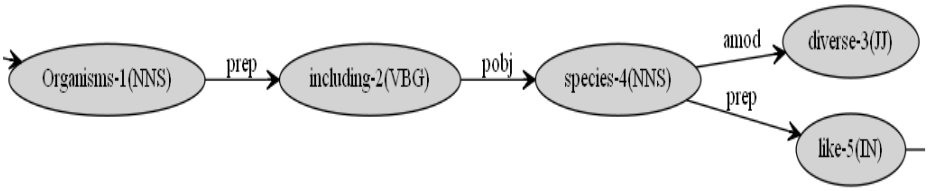

Fig. 2. (t4) A sub-tree of the syntactic dependency tree of the sentence (s4)

The limitations of flat patterns mentioned in Section 2.2, can be overcome by using patterns that take into account deep linguistic information, i.e. **syntactic dependency links**[3]. We call these patterns Dependency Tree patterns (DT patterns). For example, the limitation involving the Hearst pattern *P* and sentences (s3) and (s4) mentioned in Section 2.2 can be overcome by the following DT pattern *P2* " <Hypernym>(NP) --prep--> **including(VBG**[4]**)** --pobj--> <Hyponym>(NP) ". A matching between *P2* and

[3] The dependency links (*mwe, prep, pobj, amod,* etc.) mentioned in this paper are described in [13].

[4] VBG is a part of speech tag corresponding to a gerund or the present participle of a verb.

the dependency tree t3/t4 (of the sentence s3/s4) in Figures 1,2 above allows the extraction of the same relation (r1).

Using dependency tree patterns for relation extraction has been proposed in [12] in which the authors presented an algorithm for discovering patterns expressed as dependency paths. Those patterns allowed the authors of [12] to construct (what they called) a "hypernym-only classifier" showing a dramatic improvement compared to previous classifiers: their best logistic regression classifier showed a 132% improvement of average maximum F-score over the Hearst patterns based classifier. In [11] the authors followed a similar approach which they used to compare dependency tree patterns to flat patterns in terms of precision and recall (the patterns they used are for extracting hyponymy relations from Dutch texts) but their result is in contradiction with the work of [12]; the authors of [11] concluded that using deep syntactic information does not produce substantial improvement in the precision and recall of the extracted results. An explanation for this gap can be identified in the section 4.3 (the *error analysis* section) of [11] where one can see that most of the errors are due to the syntactic analyzer.

However, in the works mentioned above, the usage of dependency tree patterns has not been made neither systematic nor user-oriented. Indeed, in those works, there has been no specification of a **formal language** (in the same way that JAPE allows to express flat patterns to be used modularly by extraction tools) for expressing DT patterns that can be used by users for programming and experimenting DT patterns.

The most similar work to ours is [16] which presents a new ontology learning system (*OntoCmaps*) intended to overcome the drawbacks of tools using flat patterns which contain only shallow linguistic information (such as *Text2Onto*). The patterns used in *OntoCmaps* [17] contain deep linguistic information and are expressed in a language syntactically different than ours. Both languages are meant to make patterns use deep linguistic information for extracting knowledge from the text through the usage of regular expressions. Some of the differences between the language used to express patterns in *OntoCmaps* and DTPL (defined in Section 3) is that the later allows to use many POS tags for a node, it also allows to express properties for patterns (as the JAPE language does) that extraction tools could use modularly. Another difference between the two languages is that each pattern expressed in DTPL is meant to extract only **one kind** of relations, the reason is that each time that a pattern identifies more than one kind of relations it indicates nested patterns to differentiate by specifying dependency bindings (each dependency binding consists of a dependency link, the governor and the dependent) that should not exist when a match occurs (by adding the symbol *'!'* to the dependency binding to exclude from the matching, in [18] we present an example for such use); this distinction is needed because the extracted results of one pattern could be erroneous for another one. Another difference is that *OntoCmaps* uses collapsed dependencies [13] while we use uncollapsed ones.

3 Improvements in ontology building by using DT patterns

The key contribution of this paper consists in giving OBSS the ability to modularly use **patterns expressed as dependency trees** (Dependency Tree patterns—DT patterns) to take into account deep syntactic information found in texts. This will be achieved by (I) Specifying a formal language for expressing DT patterns to be matched with syntactic dependency trees of sentences, and (II) Creating and integrat-

ing a new algorithm in an ontology building tool (*Text2Onto*) to extract semantic relations by using DT patterns.

3.1 DTPL, a language for expressing DT patterns

In this section we are going to define DTPL (Dependency Tree Patterns Language), a language for expressing patterns represented as dependency trees, each DT pattern helps to extract a semantic relation. In order to extract a given relation from a sentence *S*, a DT pattern must be matched with the syntactic dependency tree of *S*.

Dependency trees (both patterns and sentences) comprise nodes and arrows. Table 1 provides the reader with the relevant definitions (the 3rd column concerns patterns—DT patterns— only).

Tree component	Components of the tree component	Optional or Mandatory
Node	*NodeValue*: it represents either a non-terminal symbol (output annotation) or a terminal symbol (word)	mandatory
	PosTags: the part(s) of speech of the symbol represented by this node	mandatory (the wildcard character * can be used, it matches with any POS tag)
	Index: the index of the symbol in the sentence in which the pattern is to be matched	optional
Arrow	*DependencyLink*: the name of the dependency link that exists between the two connected nodes	mandatory
	SourceNode: the node from which the arrow is departing	mandatory
	ArrivalNode: the node to which the arrow is aiming at	mandatory

Table 1. Inner components of a dependency tree

In DT patterns, each node must possess only one parent, with one exception for any node linked by the *ref* dependency link (i.e. in a sentence containing a co-reference, the *ref* link binds a relative pronoun with the noun it refers to) with its governor, the reason is that such nodes have more than one parent (for more detail on co-reference links used in this paper we refer the reader to [13]). In other terms, without the occurrences of the *ref* link, a DT pattern must have a tree structure. In [18] the reader can find examples that illustrate how the use of co-reference links in DT patterns allows to extract semantic relations.

In DT patterns, each **node** has to be expressed in the form **NodeValue-Index(PosTags)** (e.g. the nodes " as(IN) ", " as-5(IN) ", " as(*) "), see the example at the end of this subsection. Each **arrow** must be expressed in the form " **DependencyLabel(SourceNode,ArrivalNode);** ". The only imposed constraint is that there must be no spaces between the closed parenthesis ") " and the " ; " character for expressing each arrow (for instance, in the arrow " mwe(as(IN),such(JJ)); " of the DT pattern (dtp1) at Figure 3, we have *DependencyLink*=mwe, *SourceNode*=as(IN) and *ArrivalNode*=such(JJ)). The output annotation labels (which correspond to non-

terminal symbols) are in the form **<annotationLabel>**. For instance, *<firstHyponym>*, *<domain>*, *<range>* and *<relationName>* are nonterminal symbols in the pattern (dtp1).

DT patterns can possess properties. Each property corresponds to a non-terminal symbol. Each pattern property is defined between two *'#'* characters in the form *#propertyName=regularExpression#*, where *propertyName* is the name of the property, and *regularExpression* is a regular expression combining terminal and non-terminal symbols except pattern properties (for instance, in the pattern (dtp1) it's not allowed to define the *<relationName>* property as follows *#<relationName>=<domain>_to_<range>#* because the non-terminal symbols *<domain>* and *<range>* are also properties of the pattern).

A DT pattern allows to extract a relation (unary, binary, or having any other non-null arity). The idea is that each argument of a relation can be pointed out by a pattern property. For instance, to identify the hyponym and hypernym of a hyponymy relation, one can use the annotations **<hyponym>** and **<hypernym>**. For binary relations (which are quite important because –for instance– any Description Logics formalization of an ontology comprises only binary relations), we can use the properties *<domain>* to represent the Domain of a relation and *<range>* to represent its Range.

The expressions written in a DT pattern are either defining properties (e.g. the 1st three lines in the patterns of Figure 3) or defining dependency links between nodes. The order on which POS tags are mentioned for each node isn't important (for instance, in (dtp2), the nodes *<verb>(VBN|VBZ|VBD)* and *<verb>(VBZ|VBD|VBN)* are the same).

For extracting binary relations for ontologies, DT patterns have to contain the three properties *<relationName>*, *<domain>*, and *<range>*.

#<**relationName**>=is-hyponym-of#
#<**domain**>=<conjDep>#
#<**range**>=<prepositionGov>#
mwe(as(IN),such(JJ));
pobj(as(IN),<firstHyponym>(NN|NNS|NNP));
prep(<prepositionGov>(NN|NNS),as(IN));
conj(<firstHyponym>(NN|NNS|NNP),<conjDep>(NN|NNS|NNP));
	#<**relationName**>=<verb>_<directObject>_<preposition>#
	#<**domain**>=<depNoun> <subject>#
	#<**range**>=<prepositionalObject>#
	nsubj(<verb>(VBN|VBZ|VBD),<subject>(NNP|NN|NNS));
	pobj(<preposition>(IN|TO),<prepositionalObject>(NNP|NN|NNS));
	dobj(<verb>(VBZ|VBN|VBD),<directObject>(NNP|NN|NNS));
	prep(<verb>(VBZ|VBD|VBN),<preposition>(IN|TO));
	nn(<subject>(NNP|NN|NNS),<depNoun>(NNP|NN|NNS));

(dtp1) DT pattern similar to Hearst's *such as* pattern	(dtp2) DT pattern for extracting semantic relations based on intransitive verb phrases (verb phrases of which the verb is intransitive) containing prepositions

Fig. 3. The DT patterns (dtp1) and (dtp2)

In Figure 3, in the DT pattern (dtp1), the output annotation *<domain>* represents the hyponym, while *<range>* represents the hypernym. In (dtp2), the output annotation *<domain>* represents the subject of the verb annotated by *<verb>*, while *<range>* represents the prepositional object. The tree (tdtp1) in Figure 4 is a way to visualize the DT pattern (dtp1). For visualizing (dtp2) we refer the reader to [18].

The Domain and Range also have to be written as regular expression. For details on the syntax of DTPL we refer the reader to [18].

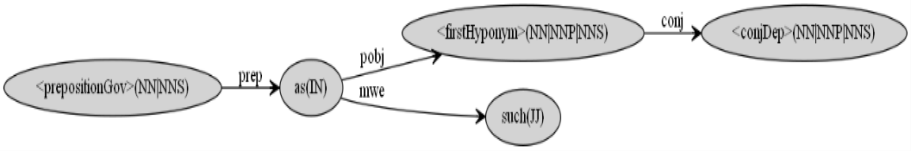

Fig. 4. (tdtp1) A visual representation of the DT pattern (dtp1)

Matching (dtp1) with the dependency tree (t5) in Figure 5 (the syntactic dependency tree of the sentence (s5) *"Carmakers such as Maruti, Hyundai, Tata, Toyota, Ford, GM & Mercedes put brakes on price hikes despite margin pressures"*) allows to extract the relations ***is-hyponym-of(mercedes,carmaker)***, ***is-hyponym-of(ford,carmaker)***, ***is-hyponym-of(toyota,carmaker)***, ***is-hyponym-of(tata,carmaker)***, ***is-hyponym-of(hyundai,carmaker)***, ***is-hyponym-of(gm,carmaker)***. While the pattern (dtp2) allows to extract from the tree (t6) in Figure 6 (the syntactic dependency tree of the sentence (s6)*"The Ebola virus causes internal bleeding to its victims"*) the relation ***cause_bleeding_to(ebola virus,victim)*** (r2).

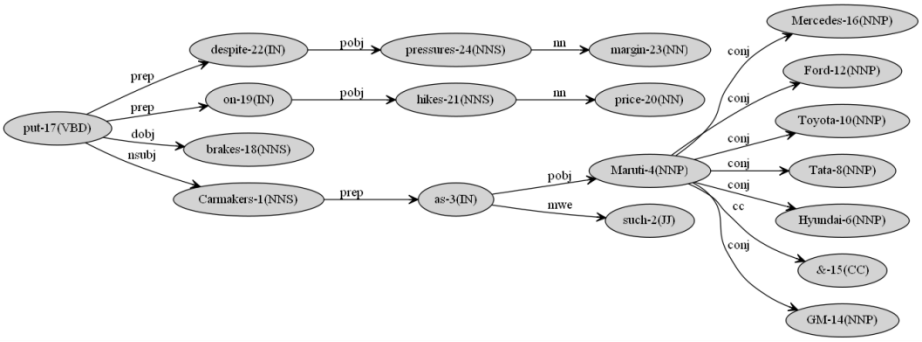

Fig. 5. (t5) The syntactic dependency tree of the sentence (s5)

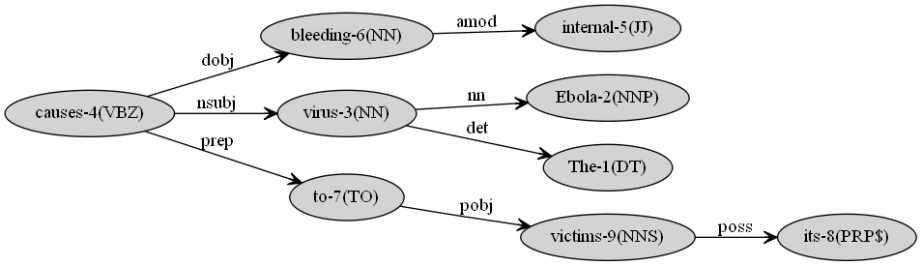

Fig. 6. (t6) The syntactic dependency tree of the sentence (s6)

Other examples (including the exploitation of co-reference resolutions, and also examples of expressing the exclusion of some dependency bindings in DT patterns — to improve the accuracy of patterns—) can be found at [18].

3.2 Text2Onto enhancement and improvement by introducing DTPL

For relation extraction purposes, *Text2Onto* has been used as target for testing DT patterns expressed by the language DTPL.

Text2Onto comprises various algorithms for ontology extraction tasks (such as relation extraction, concept/instance extraction). Given our interest on relation extraction, we will only present *Text2Onto*[5] native algorithm for relation extraction, named *SubcatRelationExtraction*.

Difference	DTP_BinaryRelationExtraction	SubcatRelationExtraction
The usage of deep linguistic information	Uses deep linguistic information for extracting semantic relations.	Uses shallow syntactic information for relation extraction.
The language used for expressing patterns	Uses patterns expressed in DTPL.	Uses patterns expressed in JAPE.
Extracting multiple instances of a relation	Extracts several instances of a relation without restrictions and provides a frequency, indicative of the relevance of a relation instance.	Extracts only one instance of a relation formed by the most frequent element of the Domain and the most frequent element of the Range (which is a source of errors).
Allowing more modularity for relation extraction	Allows the explicit naming of the extracted relation (is-a relations, part-of relations, verb phrase based relations, etc.).	• Constrained to extract only relations based on verb phrases (for instance, for transitive verb phrases, it generates relation instances in the form ***Verb(Subject,Object)***). • It does not allow the explicit naming of the relation.

Table 2. Differences between *DTP_BinaryRelationExtraction* and *SubcatRelationExtraction*

SubcatRelationExtraction is a pattern-based relation extraction algorithm using flat patterns expressed in JAPE (here, patterns are called JAPE rules) located in *Text2Onto*'s */3rdparty/gate/english* directory. *SubcatRelationExtraction* takes into account the information that JAPE rules possess (like the output annotations *TransitiveVerbPhrase*, *Subject* and *Object*) to generate relations. However, another limitation of *SubcatRelationExtraction* (other than using flat patterns) is the exclusive usage of verb phrases (transitive/intransitive, etc.) for extracting and naming relations. For instance, from the sentence (s6), *SubcatRelationExtraction* extracts (given the **right** JAPE rule) ***cause_to(ebola virus,victim)*** (which is not as meaningful as (r2)). Indeed, despite the fact that numerous semantic relationships can be identified from verb phrases, this is not always the case (as for the hyponymy relation in sentence (s5)).

We implemented the new relation extraction algorithm *DTP_BinaryRelationExtraction*, which uses DT patterns expressed in DTPL for binary relation extraction. *DTP_BinaryRelationExtraction* uses a JAVA library (*TreeMatcher*) which generates relations by using the regular expressions that define the pattern properties *<domain>*, *<range>* and *<relationName>*.

To parse the input texts, *TreeMatcher* uses the *Stanford Full Parser* version 3.3.1 (which can be found at [14]) and its parsing model *englishPCFG.ser.gz*. *TreeMatcher*

[5] *Text2Onto* version 2007-11-09, it can be found at https://code.google.com/p/text2onto/.

is tolerant to DT patterns containing empty lines and multiples space characters as well. Each pattern has to be specified in DTPL in a distinct file (textual file) within *Text2Onto*'s */3rdparty/gate/english* directory. The name of each file containing a DT pattern has to start with "dtp-". For example, the DT pattern (dtp1) can be named "dtp-SuchAsPattern".

DT patterns can be added, removed, modified and used modularly by *Text2Onto* like any flat pattern expressed in JAPE (e.g. the JAPE rules *SubclassOfRelation1*, *SubclassOfRelation2*, etc. of the JAPE file *ontological_relations.jape* in *Text2Onto*'s */3rdparty/gate/english* directory).

TreeMatcher allows *DTP_BinaryRelationExtraction* to process relation extraction in four steps: (I) Reading the DT patterns in *Text2Onto*'s */3rdparty/gate/english* directory, (II) Producing syntactic dependency trees from the input corpus by using the *Stanford Full Parser*, (III) Performing matching between dependency trees extracted from the corpus and the DT patterns, each matching can produce many relations (the generation of relations uses the regular expressions attached to the properties <*domain*>, <*range*> and <*relationName*> (see Section 3.1)), and each relation contains the frequency of its occurrence on the corpus, (IV) Producing the result as a list of relations (each relation instance possess an index indicative of its relevance).

Table 2 above summarizes the differences between the algorithm *DTP_BinaryRelationExtraction* and *Text2Onto*'s native algorithm *SubcatRelationExtraction*.

4 Conclusion and perspectives

We have presented in this paper a method giving ontology building tools the ability to use deep linguistic information in patterns called DT patterns. Specifically, we have first defined a new language (DTPL) to express these patterns, and, accordingly, enhanced and improved an existing ontology building tool (*Text2Onto*).

The work that we described in this paper is a piece of a bigger scheme aiming at:

- The integration of a new parsing strategy (especially made for relation extraction algorithms) assuring the accuracy of the extracted relations (because of the deep syntactic analysis) while maintaining a reasonable computational cost;
- Introducing two weakly supervised algorithms for pattern discovery, one for DT patterns and the other for flat patterns. The patterns to be learned are for extracting sematic relations (including IS-A and part-of).

Using deep linguistic information needs deep syntactic analysis of the text which takes longer runtime than shallow parsing. This may be overcome by using a strategy for relation extraction which consists in parsing only sentences that contain at least two Terms Representative of the knowledge Domain of the corpus (TRD), the methods for extracting such terms need only shallow parsing. This strategy is expected to enhance the precision of the extracted results, but the gain in computational time depends on how many sentences contain at least two TRDs in the corpus (i.e. if such sentences are too frequent, then there would be no significant gain).

REFERENCES

1. Sergey Brin. 1998. *"Extracting patterns and relations from the world wide web"*. WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT '98.
2. Sharon A. Caraballo. 1999. *"Automatic acquisition of a hypernym-labeled noun hierarchy from text"*. In Proceedings of ACL-99. pp 120-126, Baltimore, MD.
3. Philipp Cimiano, Johanna Völker. 2005. *"Text2Onto: a framework for ontology learning and data-driven change discovery"*. In Proceedings of the 10th international conference on Natural Language Processing and Information Systems, June 15-17, 2005, Alicante, Spain [doi>10.1007/11428817_21].
4. Zellig S. Harris. 1954. *"Distributional structure"*. Word 10 (23): 146–162.
5. Marti A. Hearst. 1992. *"Automatic Acquisition of Hyponyms from Large Text Corpora"*. In Proceedings of ACL-92. Nantes, France.
6. Maayan Geffet, Ido Dagan. 2005. *"The distributional inclusion hypotheses and lexical entailment"*. In Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, p.107-114, June 25-30, 2005, Ann Arbor, Michigan [doi>10.3115/1219840.1219854].
7. Maayan Zhitomirsky-Geffet, Ido Dagan, Idan Szpektor, Lili Kotlerman. 2010. *"Directional distributional similarity for lexical inference"*. Natural Language Engineering, 16(04): 359–389.
8. Patrick Pantel, Marco Pennacchiotti. 2006. *"Espresso: leveraging generic patterns for automatically harvesting semantic relations"*. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, p.113-120, July 17-18, 2006, Sydney, Australia.
9. Patrick Pantel, Deepak Ravichandran. 2004. *"Automatically labeling semantic classes"*. In Proceedings of HLT/NAACL-04. pp. 321-328. Boston, MA.
10. Patrick Pantel, Deepak Ravichandran, Eduard H. Hovy. 2004. *"Towards terascale knowledge acquisition"*. In Proceedings of COLING-04. pp. 771-777. Geneva, Switzerland.
11. Erik T. K. Sang, Katja Hofmann. 2009. *"Lexical patterns or dependency patterns: which is better for hypernym extraction?"*. In Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL '09). Association for Computational Linguistics, Stroudsburg, PA, USA, 174-182.
12. Rion Snow, Daniel Jurafsky, Andrew Y. Ng. 2004. *"Learning syntactic patterns for automatic hypernym discovery"*. In NIPS 2004.
13. Marie-Catherine de Marneffe, Christopher D. Manning. 2008. *"Stanford typed dependencies manual"*: http://nlp.stanford.edu/software/dependencies_manual.pdf
14. *"The Stanford Parser: A statistical parser"*: http://nlp.stanford.edu/software/lex-parser.shtml
15. Julie Weeds, David Weir, Diana McCarthy. 2004. *"Characterizing Measures of Lexical Distributional Similarity"*. In Proceedings of Coling-04. Geneva, Switzerland.
16. Amal Zouaq, Dragan Gasevic, Marek Hatala. 2011. *"Towards open ontology learning and filtering"*. Information Systems, v.36 n.7, p.1064-1081, November, 2011 [doi>10.1016/j.is.2011.03.005].
17. Amal Zouaq, Dragan Gasevic, Marek Hatala. 2012. *"Linguistic Patterns for Information Extraction in OntoCmaps"*. In Proceedings Of the 3rd Workshop on Ontology Patterns - WOP2012, in conjunction with the 11th International Semantic Web Conference, Boston, USA.
18. *"Grammar of the DTPL language, and examples"*: http://people.irisa.fr/Nicolas.Bechet/WOP2014/