Preliminary Results of Logical Ontology Pattern Detection Using SPARQL and Lexical Heuristics

Ondřej Šváb-Zamazal[1,2], François Scharffe[2], and Vojtěch Svátek[1]

[1]University of Economics, Prague, {ondrej.zamazal,svatek}@vse.cz
[2] INRIA & LIG, Montbonnot, France, {francois.scharffe}@inria.fr

Abstract. Ontology design patterns were proposed in order to assist the ontology engineering task, providing models of specific construction representing a particular form of knowledge. Various kinds of patterns have since been introduced and classes of patterns identified. Detecting these patterns in existing ontologies is needed in various scenarios, for example the detection of the the two parts of an alignment pattern in an ontology matching scenario, or the detection of an anti-pattern in an optimization scenario.
In this paper we present a novel method for the detection of logical patterns in ontologies. This method is based on both SPARQL, as the underlying language for retrieving patterns, and a lexical heuristic constraining the query. It extends our previous works on ontology patterns modeling and detection. We describe an algorithm computing a token-based similarity measure used as the lexical heuristic. We conduct an experiment on a large number of Web ontologies, obtaining interesting measures on the usage frequency of three selected patterns.

1 Introduction

Ontology Patterns turn out to be an important instrument in many diverse applications on the Semantic Web. This is reflected by many different *Ontology Design Pattern* types such as *Logical Patterns*, *Content Patterns*, *Refactoring Patterns*, *Transformation Patterns* or *Alignment Patterns*[1]. Many applications of ontology patterns need to detect patterns at first. In this paper we present preliminary experiments with logical ontology pattern detection using SPARQL and additional lexical heuristics. This work extends the work presented in [7, 8] in terms of detection experiments over real ontologies. Our particular motivation for ontology pattern detection is *ontology transformation* where detection is the first step. In [8] we argue that ontology transformation is useful for many different Semantic Web use-cases such as ontology matching and ontology re-engineering. The remainder of this paper is organized as follows: in the next section we describe three ontology patterns and the way to detect them. Then in Section 3 we present preliminary results of a large scale detection experiment along with illustrative examples of these ontology patterns. We wrap-up the paper with conclusions and future work.

[1] `http://ontologydesignpatterns.org/wiki/OPTypes`

2 Patterns and their Detection

All of the logical ontology patterns introduced in this section have been presented before [8]. We introduce here preliminary results of their detection over a collection of real ontologies. The detection of these patterns has two aspects: structural and naming ones. Our method first detect the structural aspect using the SPARQL language[2]. We currently use the SPARQL query engine from the Jena framework[3]. SPARQL queries corresponding to each detected pattern are detailed in sections below. Then, the method applies the lexical heuristic computed by Algorithm 1.

Algorithm 1 *calculateAverageTokenBasedSimilarityMeasure*

```
MainEntity ⇐ lemmatized tokens of main entity
Entities ⇐ lemmatized tokens of entities
c ⇐ 0
i ⇐ 0
for all u ∈ Entities do
  if |u ∩ MainEntity| ≠ ∅ then
    i ⇐ i + 1
  end if
end for
c ⇐ i/|Entities|
return c
```

Algorihtm 1 computes an average token-based similarity measure c. The particular instantiation of $MainEntity$ and $Entities$ depends on the ontology pattern, see below. This algorithm works on names of entities (a fragment of the entity URI) which are tokenised (See [6]) and lemmatized.[4] Lemmatization can potentially increase the recall of the detection process. The lexical heuristics constraint is fulfilled when c exceeds a certain threshold which is dependent on particular ontology pattern. The motivation of this computation is based on an assumption that entities involved in patterns share tokens. More entities share the same token, the higher probability of occurrence of a pattern. We detail below three patterns that were detected in the experiment described in Section 3.

2.1 Attribute Value Restriction

The AVR pattern has been originally introduced in [4] as a constituent part of an *alignment pattern*, a pattern of correspondence between entities in two ontologies. Basically, it is a class the instances of which are restricted with some

[2] `http://www.w3.org/TR/rdf-sparql-query/`
[3] http://jena.sourceforge.net/
[4] We use the Stanford POS tagger `http://nlp.stanford.edu/software/tagger.shtml`.

attribute value. The SPARQL query for detection of this ontology pattern is the following:

```
SELECT ?c1 ?c2 ?c3
WHERE {
?c1 rdfs:subClassOf _:b.
_:b owl:onProperty ?c2.
_:b owl:hasValue ?c3.
?c2 rdf:type owl:ObjectProperty.
FILTER (!isBlank(?c1)) }
```

In this query we express a value restriction applied on a named class. Furthermore restricting properties must be of the type 'ObjectProperty' in order to have individuals (eg. 'Sweet') as values and not data types (eg. String). Currently we do not consider the naming aspect for this pattern.

2.2 Specified Values

We first considered the SV pattern in [7], but it had been originally presented in a document from the SWBPD group[5]. This ontology pattern deals with 'value partitions' representing specified collection of values expressing 'qualities', 'attributes', or 'features'. An example is given in the next section 3.

There are mainly two ways for capturing this pattern which are reflected by two different SPARQL queries. Either individuals where qualities are instances can be used for the detection:

```
SELECT distinct ?p ?a1 ?a2
WHERE {
?a1 rdf:type ?p.
?a2 rdf:type ?p.
?a1 owl:differentFrom ?a2 }
```

Or subclasses where qualities are classes partitioning a 'feature' can be used:

```
SELECT distinct ?p ?c1 ?c2
WHERE {
?c1 rdfs:subClassOf ?p.
?c2 rdfs:subClassOf ?p.
?c1 owl:disjointWith ?c2
FILTER (
!isBlank(?c1) && !isBlank(?c2) && !isBlank(?p))}
```

We are interested in mutually disjoint named classes (siblings) and we use non-transitive semantics (ie. direct) of 'subClassOf' relation here. Otherwise we would get 'specified value' as many times as there are different superclasses for those siblings. Regarding the initialisation of variables from the Algorithm 1, the $MainEntity$ is either a $?p$ instance (for the first query) or class (for the second query). $Entities$ are all other entities from the $SELECT$ construct. The experimental setting for the threshold is 0.5.

[5] http://www.w3.org/TR/swbp-specified-values/

2.3 Reified N-ary Relations

We have already considered the N-ary pattern in [7]. It has also been an important topic of the SWBPD group [2], because there is no direct way how to express N-ary relations in OWL[6]. Basically, a N-ary relation is a relation connecting an individual to many individuals or values. For this pattern we adhere to a solution introduced in [2]: introducing a new class for a relation which is therefore reified. For examples in the next section 3 we will use the following syntax (property(domain,range)):

$$relationX(X,Y); relationY1(Y,A); relationY2(Y,B)$$

The structural aspect of this pattern is captured using the following SPARQL query:

```
SELECT ?relationX ?Y ?relationY1 ?relationY2 ?A ?B
WHERE {
?relationX rdfs:domain ?X.
?relationX rdfs:range ?Y.
?relationY1 rdfs:domain ?Y.
?relationY1 rdfs:range ?A.
?relationY2 rdfs:domain ?Y.
?relationY2 rdfs:range ?B
FILTER (?relationY1!=?relationY2)}
```

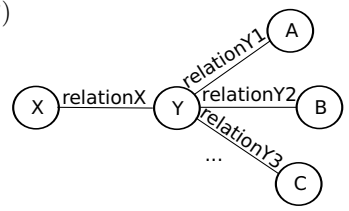

Fig. 1. N-ary relation

The intended structure of this reified N-ary relation pattern is depicted in the Figure on the right. In order to increase the precision of the detection we also apply lexical heuristics introduced above in Algorithm 1: variable $MainEntity$ is initialised with the value ?$relationX$. $Entities$ are all other entities from the $SELECT$ construct. The experimental setting for the threshold is 0.4.

3 Experiment

In order to acquire a high number of ontologies, we applied the Watson tool[7] via its API. We searched ontologies imposing conjuction of the following constraints: OWL as the representation language, at least 10 classes, and at least 5 properties. Alltogether we collected 490 ontologies. However, many ontologies have not been accessible at the time of querying or there were some parser problems. Futhermore we only include ontologies having less than 300 entities. All in all our collection has 273 ontologies.

Table 1 presents overall numbers of ontologies where certain amount of ontology patterns were detected.

We can see that patterns were only detected in a small portion of ontologies from the collection. In four ontologies, the AVR pattern was detected more than 10 times. It reflects the fact that some designers tend to extensively use this

[6] It also holds for OWL 2. The notion of N-ary datatype was not introduced there, except for syntactic constructs allowing further extensions, see `http://www.w3.org/TR/2009/WD-owl2-new-features-20090611/#F11:_N-ary_Datatypes`

[7] `http://watson.kmi.open.ac.uk/WS_and_API.html`

	$\geq 10\times$	$(9-4)\times$	$3\times$	$2\times$	$1\times$	all
AVR pattern	$4\times$	–	$2\times$	$1\times$	$1\times$	$8\times$
SV pattern	–	$4\times$	–	$2\times$	$9\times$	$15\times$
N-ary pattern	–	$5\times$	$4\times$	$16\times$	$25\times$	$50\times$

Table 1. Frequency table of ontologies wrt. number of ontology patterns detected.

pattern. Other two ontology patterns were not so frequent in one ontology (the SV pattern was detected maximally 8 times and the N-ary pattern was detected maximally six times). On the other hand the most frequent pattern regarding a number of ontologies was the N-ary pattern. This goes against an intution that this pattern is quite rare. It can be explained with a low precision detection of this pattern, see below.

In order to obtain raw preliminary precision estimation for the ontology pattern detection we analysed one randomly chosen detected pattern instance from 10 ontologies (in the case of the AVR pattern from 8 ontologies). Although we tried to apply ontology transformation perspective for manual evaluation, we could not fully avoid coarse-grained and subjective evaluation due to soft boundaries between ontology patterns and sometimes unexpected conceptualisations in some Web ontologies.

The overall precision for the AVR pattern is 0.6, for the SV pattern 0.7, and for the N-ary pattern 0.3. For better insight we will look at two examples (one positive and one negative) for each of these ontology patterns.

AVR pattern This ontology pattern was found many times in a wine ontology[8] with high precision. One positive example is the following:

$$\ni hasColor.\{White\} \sqsupseteq Chardonnay$$

Chardonnay wine is restricted on these instances having value 'White' for the property hasColor. On the other hand, one negative example is the following[9]:

$$\ni \_2.\{coordinate\_0\} \sqsupseteq North$$

In this ontology each point of the compass (eg. North) is described using three different relations (_2 is one of them) having coordinates. This cannot be interpreted as an attribute value restriction pattern.

SV pattern The following[10] is one example which we evaluated as positive (a shared token is 'Molecule', $c = 1.0$):

$$Molecule \sqsupseteq AnorganicMolecule; Molecule \sqsupseteq OrganicMolecule$$

This can be interpreted as a collection of different kinds of molecules which is a complete partitioning. Furthermore disjointness is ensured by a query. On the other hand in another ontology[11] a negative example was detected:

[8] http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine
[9] http://sweet.jpl.nasa.gov/ontology/space.owl
[10] http://www.meteck.org/PilotPollution1.owl
[11] http://www.cip.ifi.lmu.de/~oezden/MastersThesisWeb/STCONCEPTS.owl

$TimePeriods \sqsupseteq SocioculturalTimePeriods; TimePeriods \sqsupseteq CalendarDatePeriods$

In this case it can hardly be a complete partitioning, however this is always dependent on domain of discourse. It points out that ontology pattern detection should generally be a semi-automatic process.

N-ary pattern This pattern has the lowest precision. Due to the usage of a relaxed structural condition there are a lot of negative cases. Even if the lexical heuristics constraint improves this low precision, there is still ample space for improvement.

In the *PML* ontology[12] the following positive example was detected:

$hasPrettyNameMapping(InferenceStep, PrettyNameMapping)$
$hasPrettyName(PrettyNameMapping, string)$
$hasReplacee(PrettyNameMapping, string)$

This is the example of N-ary relation where the reified property 'PrettyNameMapping' ('?Y') captures additional attributes ('hasReplacee') describing the relation ('hasPrettyNameMapping'). $c = 0.5$ where shared tokens were 'Pretty' resp. 'has'.

On the other hand in the *earthrealm*[13] a negative example was detected:

$hasUpperBoundary(EarthRealm, LayerBoundary)$
$isUpperBoundaryOf(LayerBoundary, EarthRealm)$
$isLowerBoundaryOf(LayerBoundary, EarthRealm)$

In this case 'LayerBoundary' was detected as a reified N-ary relation ('?relationX') connecting different aspects ('?relationY1', '?A' and '?relationY2','?B') of the same relation; $c = 0.75$ where a shared token was 'Boundary'. But if we look at the classes bound with '?X', '?A' and '?B', we can see that there is an implicit 'inverseOf' relation between 'hasUpperBoundary' and 'isUpperBoundaryOf' resp. with 'isLowerBoundaryOf'. This 'inverseOf' relation cannot be directly found in this ontology. However we could assume this and therefore we could increase the precision considering this specific case in the future work.

Another recurrent negative example is the following[14]:

$concludedby(perdurant, perdurant)$
$startedby(perdurant, perdurant)$
$concludedby(perdurant, perdurant)$

This is a chain of properties connected with the same class in domain/range; $c = 0.66$ where a shared token is 'by'. Thus, at the first sight a detection could be improved with considering this negative example ('?X'='?Y'='?A'='?B'). On the other hand we can also find a counter-example considering true semantics of domain and range in OWL, ie. restrictions are superclasses of possible individuals. As usual, applying this condition we could filter out some current negative examples (getting higher precision), on the other side we could miss other positive examples (getting lower recall). This choice is application-dependant.

4 Related Work

In [3] the authors generally consider using SPARQL expressions for extracting Content Ontology Design Patterns from an existing reference ontology. It is followed by a manual

[12] `http://inferenceweb.stanford.edu/2004/07/iw.owl`
[13] `http://sweet.jpl.nasa.gov/sweet/earthrealm.owl`
[14] `http://neuroscientific.net/bio-zen.owl`

selection of particular useful axioms towards creating new Content Ontology Design Pattern. The Ontology Pre-Processing Language (OPPL) is specialized on pattern-based manipulation with ontologies. It can be used for ontology pattern detection, however there is no such a lexical support which our detection needs. In [1] the authors envision employing OPPL for detecting recurring patterns in ontologies and materialize them as new patterns. This is also one of our long-term effort.

By now, we use the SPARQL language for detecting structural aspect of ontology patterns. But SPARQL is a query language for RDF. Considering ontology patterns as DL-like conceptualisations, it leads to a necessity of expressing DL-like concepts in RDF representations which is rarely 1:1. This can be overcome by using some OWL-DL aware query language, eg. SPARQL-DL [5]. However for now this language does not support some specific DL constructs e.g. *restriction* and it is not fully implemented yet.

5 Conclusions and Future Work

In this paper we presented preliminary results of logical ontology pattern detection for which we use SPARQL and lexical heuristics. We conducted an experiment on a large number of Web ontologies. We manually evaluated 28 ontology pattern instances so as to roughly estimate precision. The performance of this detection must be further improved. Besides future work depicted in Section 3, we have to further work on more sophisticated lexical heuristics constraint. We could also employ head noun detection [6]. Further, we should also try to perform our queries using SPARQL-DL as OWL-DL aware language. We work on a specific FILTER extension (in connection with head noun detection) for the SPARQL language which could include the naming aspect already at the level of the query language.

Acknowledgement The work has been partially supported by the IGA VSE grant no. 20/08 “Evaluation and matching ontologies via patterns”.

References

1. A. R. Luigi Iannone and R. Stevens. Embedding Knowledge Patterns into OWL. In *Proceedings of the 6th European Semantic Web Conference*, 2009.
2. N. Noy and A. Rector. Defining n-ary relations on the semantic web, Apr. 2006.
3. V. Presutti and A. Gangemi. Content ontology design patterns as practical building blocks for web ontologies. In *Proceedings of ER2008*. Barcelona, Spain, 2008.
4. F. Scharffe. *Correspondence Patterns Representation*. PhD thesis, University of Innsbruck, 2009.
5. E. Sirin and B. Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *OWLED2007*, 2007.
6. O. Šváb Zamazal and V. Svátek. Analysing Ontological Structures through Name Pattern Tracking. In *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management*, 2008.
7. O. Šváb-Zamazal and V. Svátek. Towards Ontology Matching via Pattern-Based Detection of Semantic Structures in OWL Ontologies. In *Proceedings of the Znalosti Czecho-Slovak Knowledge Technology conference*, 2009.

8. O. Šváb-Zamazal, V. Svátek, and F. Scharffe. Pattern-based Ontology Transformation Service. In *Proceedings of the 1st International Conference on Knowledge Engineering and Ontology Development*, 2009.